Workshop in Computers Network
(20588)

Final Project Portfolio

P2P Application

Sagi Chafetz – 203867148
Gabriel Shapiro – 326980331

Table of Contents

Table Of Figures

Objective:

The objective of the Project is to allow users to download media files such as music, pdf, and music using a P2P software client that searches for other connected computers, handled by one central server. Multiple clients, also called as peers, are computer systems that may join the file sharing system by connecting to the tracker and declaring the list of the files that wish to be shared. The tracker keeps a list of the files which are shared among the network. The files being distributed are divided into chunks and for each file, the tracker handles the list of chunks each peer has.

Once a peer receives a new chunk of the file it becomes a source for that chunk to be shared among other peers. When a peer tries to download a file, it will initiate a direct connection to the peers which have the file (or chunks of it) to download the file and will be able to download different chunks of the file simultaneously from several peers.

Both peer and tracker acts as server and client:

- **The peer** connects to tracker to get the information and connects to other peers to get the data. The connections between tracker and peer are always alive, while the connections between peers are disconnected when the file transfer is done.

- **The tracker** only listens for peers to connect to acquire information about the current files and the peers who have the file.

When downloading, the peers are pinging other peers to start seeking content from the lowest-latency-peer, while that peer will download chunks rarest-first (which fewest peers have it) and from the fastest available peer.

Hardware & Software Requirements:

Input Requirements:

The Point-To-Point sharing system will have a minimum set of following input requirements:

- **Login**: Whenever the user tries to connect to the server, it must provide a valid IP address to access the server. Based on server address, either the user is connected to the server or denied the access.
- **Online-clients:** Once the client and server get successfully connected, the server shows the list of the clients that are active that time. It is important since some of the clients may be down so the user must not keep accessing the offline clients.
- **Listing:** The peer has a possibility to send a request in order to see the available files.
- **Downloading:** While the downloading starts, progress bar keeps on indicating that the file has been downloaded successfully or not.

Output Requirements:

The Point-To-Point sharing system has a minimum set of the following output requirements:

- **Space:** The peers must have a surplus space at their HDD
- **Message:** Successful downloading of a particular file is indicated by a message at the monitor screen of the downloading peer client.

Software Requirements:

- Windows, Linux, Macintosh
- The Point-To-Point sharing system was developed on Python 3.9 IDE and is using several packages which have to be installed beforehand: math, json, time, asyncio, logging, struct, msgpack, abc, os.path, hashlib, aioconsole, beautifultable, coloredlogs, , aioconsole.stream

Hardware Requirements:

- Any modern CPU
- 4 GB RAM
- High speed internet connections (DSL/Cable)

Software Architecture:

General:

The system is designed in object-oriented programming and is based on initializing the Tracker once while each peer has to connect to it in order to publish or download a file from the list by using specific commands on the terminal.

The system is using several technologies:

- Send and receiving messages using the TCP protocol, by transforming the data in streaming between the peers within the tracker's network. This technology provides reliable stream delivery of data while keeping the order of the messages.

- The system is using an asynchronous networking in order to maximize the system's response. Both the tracker and the peer are using A-sync sockets, provided by the "asyncio" library (additional information below).

- The system is using JSON to transfer messages between the tracker/peer and pee/peer. Each message has JSON format and type field while messages with different types have different fields. The system is using a logger from the logging library in order to log any message within the peer/tracker.

- The file system is being monitored simultaneously by the peer for transforming the file into chunks and download/publish them and the tracker for providing the list of the available chunks to download based on their rareness and keep it updated according to the peers' availability.

- The system is using SHA256 for digest of the binary data and BASE64 for encoding of the binary data.
- The system is designed to be as fail-safe as possible: peers may shutdown abruptly at any time (during file transfer/when idling, etc.), and tracker can shutdown abruptly at any time. The list below describes what the system does when unexpected shutdown happens:
    - Tracker shutdown: Peers won't be able to retrieve file/peers' information until it connects to the tracker again (need to manually use the command connect <tracker_ip> <tracker_port>). Current file transfers won't be affected (as the peers' information has already been downloaded).
    - Peer shutdown: Current file transfers which the crashed peer doesn't have won't be affected and for the ones that the crashed peer is involved, the downloading peer will seek other peers for the content. If no other peers have the file, the downloading peer will report an error for incomplete transfer and the tracker will delist the file on its record.

Classes Structure:

The system is constructed from multiple classes.
The figure below describes these classes and the connection between them:

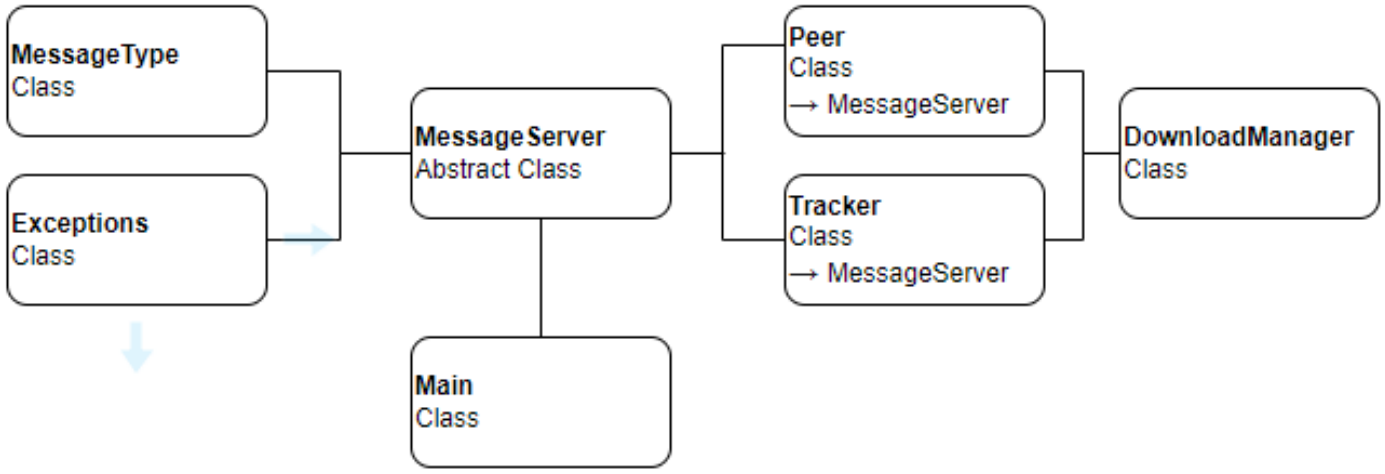

Figure 1 - Classes Structure

Classes Description:

Class Name	Description
Main	The main program of the system in which the users may run the system and choose which server they wish to initiate (Peer/Tracker)
MessageServer	Abstract class for the async TCP server which provides basic start and stop methods.
Tracker	Based on the Server class, this class provides the tracker's functionally methods and implementation to the connection method.
Peer	Based on the Server class, this class provides the Peer's functionally methods and implementation to the connection method.
DownloadManager	This class handles the downloading process of the peer, updates the chunks availability and peer's RTT (Round-Trip Time)
MessageType	Based on the Enum class, this class provides the JSON messaging system in which the peer/peer and the tracker/peer are transferring messages.
Exceptions	This class includes 5 exception classes (DownloadIncompleteError, AlreadyConnectedError, TrackerNotConnectedError, InProgressError, ServerRunningError) which are used within the Peer-to-Peer system.

Asynchronous Programming:

The Peer-to-Peer system is using the asyncio python package which enables us to execute tasks asynchronously and await for incoming data between function calls.

By using the word "async" before each function call, we insert coroutines in the stack to be executed once the tasks that were called beforehand were completed. The word "await" is being used as well during function calls to be able to use incoming data from other tasks and use it.

The AsyncIO package is used in order to work in parallel with the programs command executions and to not have to crash the whole program if data is taking time to be transferred or a client is disconnected. Instead of a full routine having to crash, only the coroutine will crash, and the rest of the program will work without issue.

This package is a pivotal package to be used in networking applications because of its multi-layer processes and multi-user parallel usage.

Overview of the Client-Server Model:

The client/server model is a computing model that acts as distributed application which partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system (such as it is on our project with peer and tracker). A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

Figure 2 - Client-Server Model

The P2P Architecture:

The P2P architecture has several unique features:

- All the peers are both clients and servers which provide and consume data.
- Any peer can initiate a connection to a tracker.

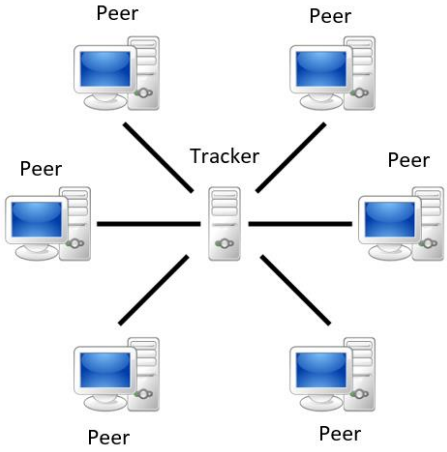

Figure 3 - P2P Architecture

P2P Network Characteristics:

- Peers are also server and routers, which contribute content, storage, memory, CPU.
- Once the peers are connected to the tracker, the peers are autonomous while the tracker provides a level of support for connectivity between them.
- Network is dynamic: peers enter and leave the network “frequently”

Instructions:

Activating the Program:

The file that the program is activated from is called __main__.py. Once the project is open you will have to first install all the required packages to your system to execute the program. Once the main file is running, you will have to go to the terminal and input certain commands to activate either a tracker terminal or a peer terminal.
To run the program as a Peer you must first start the tracker server to connect to first. The command in the python terminal will look like this:
python __main__.py tracker

After running that command, you will be able to immediately run any functionality of the tracker or type a '?' to get a set of instructions on what commands you can execute. Once the tracker user is input to the terminal, you must start the tracker with a host IP address and a port number that peer users will connect to. The command looks like this (for example):
start 127.0.0.1 3000

After this you will have started the tracker server that will maintain everything that has to do with the file sharing program. Now we will do the same thing to connect a peer to the server in order to start using the program.

To do so you will put in the command in your python terminal:
python __main__.py peer
And then run the command:
connect 127.0.0.1 3000
To connect to the server and be visible to the tracker as an online peer.

Publishing a File to The Program for Download:

Once you have connected to the tracker server, you will be visible as an online peer unless you exit the program or close the terminal. If you are connected, you can go ahead and publish your file to the server so everyone can see it. To do so you will type in the command:
publish <file location>
After doing so, the program will go through a sequence of operations to ensure that the publishing of the file will happen with no issues. If everything checks out, your file will be published as a file on the server and will be visible to other peers for download.

Search for a File to Download:

Once a peer has successfully published a file to the server, it can be visible by other peers by typing in the command in the terminal (only after the peer connected to the server):
list_files
Following the activation of this command you will be prompted with a table that will include the name of the file, the total number of kilobytes and the number of chunks that file takes. We have set that a chunk is 524 KB or
0.5 MB.
Next, we would like to download the published files to our own system, let's see how to do that:

Downloading a Published File:

After seeing the files that have been published by other users, those files are ready to be downloaded to your system. Once you have done so, that same file will be published in your name automatically for others to download from. To download the file you would like, you should type in the command:
download <name of file> <name of file in destination>

Once you press enter after typing this command, the program will initiate several inspections and tests to ensure from who, what parts, and if there is the possibility to download the entire file or not. The program will also inspect the round trip time and rarity of each chunk that will need to be downloaded in accordance with the people who own the file on the server, to enable the fastest and most organized download possible.

Looking Up Files With Chunk info:

Once you have downloaded the file you wanted to download whether partially or fully, you can now connect as the tracker again and see who owns what files and what chunk of each file. To do so you must run the command:
list_chunkinfo
Executing this command will produce a list of every file in the system as well as who owns it and exactly what chunk it owns. This is important in cases for example that a peer has disconnected while downloading from the system or the "server" peer that has published the file has disconnected in between.

Shut down the Tracker/Peer:

In order to shut down the peer/tracker you must run the command:
exit
Once the peer is shut down it will be removed from the available peers' list by the tracker.
In case the tracker is shut down, connected peers will be able to proceed with their active downloads.

Processes:

Starting the program:

In the start of the main program, a task loop is started and is waiting for incoming tasks to be placed on the asynchronous command stack. Once we open the terminal of our program, we must give our program an argument of either peer or tracker. Once that is done, the program shifts from the main to start dealing with a Tracker or Peer object and their respective terminals. After that, a terminal prompt will greet you and wait for your next command to be selected.

If a tracker is started, an IP address and a port will be given to the tracker server so others can connect to it. In addition, a list of all files will be created and ready to be modified, a list of peers will be created to keep track who is online and connected to the server at all times, and also a list of chunks and their owners will be initiated to keep track who owns what files and what chunks of that file.

If a peer is started, it will put in what tracker server IP and port it wants to connect to. A file map that holds the files that the peer has successfully published to the server is created as well. The server will send back a message of type REQUEST_REGISTERY when the peer is registered successfully.

Publishing a File to the server:

When a peer would like to publish a file to the server, it gives the location of the file on his local PC and the connection between the peer and the main server starts. First, the peer sends out a message to the server of type REQUEST_TO_PUBLISH and fills in the file info and sends a request packet through TCP to the server. The server checks if there is a file of the same name already in the server and if so, doesn't go through with the publishing. Otherwise, the file is published to the server and the file list is updated as well as the chunk info.
Afterwards, a message is sent back as a reply to the peer as a json message (every message between computers is built this way) that the file has been published successfully.

File Search:

When a peer has published a file, either that peer or a different peer will be able to lookup the files the server has that are published and their owners are online. It does so by sending a message of type REQUEST_FILES_LIST to the main server and waiting for a reply of type REPLY_FILES_LIST to be able to receive the file names and file info.

Downloading a File from the Server:

Once a file has been successfully published to the main tracker server, every peer that is connected to said server will be able to see these files as well as download them to their own system as long as the source peer is still connected. When a peer would like to download a file, it must first lookup what files are available for download with the “list_files” command. Then it sees the name of the file in the server and sends the command to download it.

The server then initiates an algorithm to determine the most efficient and time saving way to download the file from one or several peers. The way it does this is by updating the chunk info of the file in accordance with rarity of owners of each chunk as well as round trip time of each peer who owns the file, so that the chunks who are owned by several people, will be downloaded from the fastest connected peer.
Once a chunk has been determined from who it will be downloaded from, the peer sends a message to the server of type PEER_REQUESTS_CHUNK to the peers that has the chunk and waits for a reply of type PEER_REP_CHUNK. After that it downloads the chunk and then updates the server that it downloaded the chunk by sending a message of type REQUEST_CHUNKS_REGISTER to the server.

At the end of this sequence of events, the client peer finishes downloading either the file entirely or a part of the file. After that the server updates itself with regards to who owns what chunks of what file, and the peer who just finished downloading the file now becomes a server peer for the rest of the peers who wish to download that file.

Peer\Server Disconnection:

If the peer is in the middle of a download of files, file transfers which the crashed peer doesn't have won't be affected. For the ones that the crashed peer is involved with, the downloading peer will seek other peers for the chunks it has not downloaded yet. If no other peers have the file, the downloading peer will report an error for incomplete transfer and the tracker will delist the file on its record.
On the other hand, if the server shuts down, peers won't be able to retrieve file/peers’ information until it connects to the tracker again. Complete file transfers won't be affected (as the peer’s information has already been downloaded) and the state of the files will be incomplete but not corrupted.

Screenshots:

Start and connection of Tracker:

```
Welcome to Tracker terminal.    Type help or ? to list commands.

Tracker> ?
Available command list:
  -  exit
  -  help
  -  list_chunkinfo
  -  list_files
  -  list_peers
  -  start

Tracker> start 127.0.0.1 3000
Tracker started listening on ('127.0.0.1', 3000)
Tracker>
```

Figure 4- Start and connection of Tracker

Start and connection of Peer:

```
Welcome to Peer terminal.    Type help or ? to list commands.

Peer> ?
Available command list:
  -  connect
  -  download
  -  exit
  -  help
  -  list_files
  -  publish
  -  set_delay

Peer> connect 127.0.0.1 3000
Successfully connected!
Peer>
```

Figure 5- Start and connection of Peer

List of online peers:

```
Tracker> list_peers
+----------------------+
|     Peer Address     |
+----------------------+
|    ["::1", 55453]    |
| ["127.0.0.1", 55460] |
+----------------------+
Tracker>
```

Figure 6- List of online peers

Peer publishing file to server:

```
Peer> publish H:\testPublish.txt
File H:\testPublish.txt successfully published on tracker.
Peer>
Peer> publish H:\testMoviePublish.mkv
File H:\testMoviePublish.mkv successfully published on tracker.
Peer>
```

Figure 7- Peer publishing file to server

Tracker's list of files:

```
Tracker> list_files
+----------------------+-----------+----------------+
|       Filename       |   Size    | Total_chunknum |
+----------------------+-----------+----------------+
|   testPublish.txt    |    41     |       1        |
| testMoviePublish.mkv | 839430586 |      1602      |
+----------------------+-----------+----------------+
Tracker>
```

Figure 8 - Tracker's list of files

Tracker Chunk info:

```
Tracker> list_chunkinfo
{'testPublish.txt': {'["::1", 55453]': [0]}, 'testMoviePublish.mkv': {'["127.0.0.1", 55460]': [0, 1,
 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
10, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 1
 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,
7, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 23
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290,
, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344
78, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 3
 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451,
5, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 50
```

Figure 9 - Tracker Chunk info

Peer downloading File and renaming to local machine:

Figure 10- Peer downloading

```
Downloading ...: 801MB [00:04, 187MB/s]
File testMoviePublish.mkv successfully downloaded to Movietest.mkv.
Peer>
```

Figure 11 - Download complete

Figure 12 - Completed file

Peer exit and updated file list:

```
Peer> exit
```

Figure 13 - Peer exit command

```
+----------------------+-----------+----------------+
|       Filename       |   Size    | Total_chunknum |
+----------------------+-----------+----------------+
| testMoviePublish.mkv | 839430586 |      1602      |
+----------------------+-----------+----------------+
```

Figure 14 - Updated file list

Tracker Disconnect:

```
Tracker> exit
```

Figure 15 - Tracker exit command

```
Peer> list_files
Tracker is not connected, try 'connect <tracker_ip> <tracker_port>' to connect.
Peer>
```

Figure 16 - Peer exit message