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

COMPUTER NETWORKS (CO3093)

Assignment 1

"DEVELOP A NETWORK APPLICATION: PEER-TO-PEER FILE-SHARING"

Lecturers: Nguyễn Phương Duy
Nguyễn Lê Duy Lai
Students: Nguyễn Huy Hoàng – 1852382
Đinh Xuân Quang - 2053359
Nguyễn Lê Anh Tuấn - 1852837

Ho Chi Minh City, 12/2023

Table of Content

Member list and contributions

No.	Fullname	Student ID	Workload	Contribution
1	Nguyễn Huy Hoàng	1852382	Client.py, Validation	25%
2	Đinh Xuân Quang	2053359	Server.py, Validation	25%
3	Nguyễn Lê Anh Tuấn	1852837	Client.py, Server.py, Writing report	50%

1. Introduction

1.1. Defintion

Peer-to-peer (P2P) computing or networking is an architectural framework for distributed applications that divides tasks or workloads among peers. Peers, which are nodes within the network, enjoy equal privileges and collectively forming a network based on peer-to-peer principles.

The distribution and exchange of digital media through peer-to-peer (P2P) networking technology define peer-to-peer file sharing. Utilizing P2P file sharing involves accessing media files like books, music, movies, and games through specialized software. This program searches for the desired content by connecting to other computers on the P2P network. The nodes in these networks, representing end-user computers and distribution servers, operate without the need for the latter.

1.2. Principles

The basic principle of peer-to-peer (P2P) computing or networking is the decentralized sharing of resources and responsibilities among interconnected and equally privileged participants, referred to as "peers." In a P2P system, each node (or peer) has the ability to both contribute resources and utilize resources from other peers. This contrasts with traditional client-server models, where there is a clear distinction between clients (requesters) and servers (providers).

In a P2P network, tasks, data, or services are distributed across the peers, and each peer has a degree of autonomy. This decentralized architecture promotes collaboration, scalability, and fault tolerance. Peers in a P2P system can act both as consumers and providers of resources, creating a more fluid and dynamic network where no single node has complete control.

The fundamental idea is to leverage the collective power of distributed nodes, allowing them to share and exchange resources directly without relying on a central server. This approach is often associated with file sharing, content distribution, and other collaborative applications, where users can access and contribute to the network's resources without a centralized authority.

1.3. P2P Architecture

There are 3 different types of peer-to-peer network:

- Unstructured P2P networks
- Structured P2P networks
- Hybrid P2P networks

Unstructured P2P networks:

Unstructured peer-to-peer (P2P) networks lack a defined arrangement of nodes, leading to random communication between nodes. This characteristic makes unstructured P2P networks well-suited for applications characterized by high activity levels, such as social platforms, where users frequently join or leave the network.

However, unstructured P2P networks have a downside. They require substantial CPU and memory resources for effective operation, necessitating hardware support for maximum network transactions to ensure seamless communication among all nodes. This can pose challenges, especially for large networks or those experiencing high levels of activity.

Structured P2P networks:

Structured peer-to-peer (P2P) networks differ from unstructured ones by providing organized interaction among nodes. Enabled by a welldefined architecture, these networks facilitate efficient file location and utilization, eliminating the need for random searches. Hash functions often play a role in database lookups within structured P2P networks.

While generally more efficient, structured P2P networks exhibit a degree of centralization due to their organized setup. This may result in higher maintenance and setup costs compared to unstructured P2P networks. Nevertheless, structured P2P networks offer greater stability than their unstructured counterparts.

Hybrid P2P networks:

Hybrid peer-to-peer (P2P) networks are a combination of P2P and client-server models. This fusion introduces a central server with P2P capabilities, proving advantageous in specific network scenarios.

Hybrid P2P networks provide numerous advantages over structured and unstructured networks, including strategic approaches, enhanced performance, and other benefits. Overall, these networks present a compelling choice for scenarios that aim to leverage both P2P and client-server architectures.

1.4. Applications

P2P architecture is most effective when a considerable number of active peers participate in a dynamic network, facilitating easy connections for new peers entering the network. The resilience of the system is demonstrated when numerous peers leave the network, as a sufficient number of remaining peers can compensate for the decrease in participation. Conversely, with only a limited number of peers, the overall availability of resources diminishes. For instance, in a P2P file-sharing application, the speed of downloading a file is accelerated when the file is popular, indicating widespread sharing among peers.

Optimal functionality of P2P is achieved when the workload is divided into smaller units that can be reassembled later. This approach allows a multitude of peers to collaborate simultaneously on a task, distributing the workload more evenly. In the context of P2P file-sharing, a file is segmented, enabling a peer to download multiple chunks concurrently from various peers.

Various applications leverage P2P architecture for diverse purposes, including:

- File sharing
- Instant messaging
- Voice communication
- Collaboration
- High-performance computing

Several notable examples of P2P architecture include:

- Napster, which was discontinued in 2001 due to its reliance on a centralized tracking server
- BitTorrent, a widely used P2P file-sharing protocol often associated with piracy
- Skype, initially utilizing a proprietary hybrid P2P protocol but transitioning to a client-server model following Microsoft's acquisition
- Bitcoin, a P2P cryptocurrency operating without a central monetary authority.

1.5 Risks

Security Issues: The inherent openness of P2P networks exposes them to security threats such as unauthorized access, data breaches, and the distribution of malware.

Varied Quality and Reliability: Resources shared on P2P networks can exhibit significant variations in quality and reliability. It becomes crucial to verify the legitimacy and integrity of files to ensure their trustworthiness.

Complex Network Management: Maintaining a P2P network proves to be more intricate compared to traditional networks, necessitating specialized protocols and mechanisms for effective coordination.

Legal and Copyright Concerns: P2P networks have become linked with copyright infringement due to the ease of sharing copyrighted material. This association has resulted in legal complexities for both users and network operators.

Absence of Centralized Control: While the decentralized nature of P2P networks is advantageous, it also presents challenges in enforcing rules, policies, and governance uniformly across the entire network.

2 Peer-to-peer file-sharing application:

2.1 Requirements analysis:

Functional Requirements:

- Server Tracking: The server should keep track of connected clients and the files they store.
- Client File Information: A client should inform the server about the files in its local repository.
- File Request and Fetching: When a client requires a file not in its repository, it should send a request to the server. The server should identify other clients storing the requested file and send their identities to the requesting client. The client should then fetch the file directly from the identified client.
- Multithreading: The client code should be multithreaded to allow multiple clients to download different files from a target client at a given point in time.
- Command Shell: Both the client and the server should have a simple command-shell interpreter to accept commands.

Non-Functional Requirements:

- Concurrency: The system should support multiple clients downloading different files from a target client simultaneously.
- Performance: The system should quickly respond to file requests and efficiently handle file transfers.
- Reliability: The system should reliably track client connections and file locations, handle file requests, and transfer files.
- Usability: The command-shell interpreters in the client and server should be simple and easy to use.
- Network Protocol: The system should use the TCP/IP protocol stack for communication.

2.2 Functions description:

Class name	Function	Parameter	Description
Client	__init__	self, host, port	Initiate the client
	start_server	self	Start the server on the client
	accept_connections	self	Accept the connections from clients
	handle_client	self, client	Handle the message from the client
	send	self, msg	Send a message to the server
	receive	self	Receive messages from the server
	publish_all_files	self	Publish all files in the local repository
	fetch_file	self, fname, addr	Fetch a file from another specified client
	send_file	self, fname, client	Send a requested file to a client
	command_shell	self	Command-shell interpreter
Server	__init__	self, host, port	Initate the server
	handle	self, client	Handle the message received from the client
	command_shell	self	Command-shell interpreter
	run	self	Run the server

N/A	get_open_port	N/A	Retrieve a free port

2.3 Communication protocols:

The application uses the TCP/IP protocol for communication between the server and clients and between the client and client. This is evident from the use of socket.AF_INET (which stands for Internet Address Family) and socket.SOCK_STREAM (which stands for TCP) when creating the socket objects in both the server and client. This ensure a reliable data transmission between the server and clients and between the client and clients.

2.2.1 Between server and client:

Publish

When a client wants to publish a file, it sends a ‘publish’ command to the server. The command is in the format **publish <localname> <filename>**. Here, **<localname>** is the local path of the file on the client’s machine, and **<filename>** is the name under which the file should be published on the server. The server then adds the file to its list of available files and then send a reply back to the client as a confirmation message.

Discover

When the server want to discover the list of local files of a client, it send the message ‘list’ to the client. When the client received this message, it will response back with a list of the files in the local repository. The server then use this provided information to display onto the command line.

Ping

The server can ping clients to check their availability by sending a ‘ping’ command to the client, and the client responds with a ‘ping’ command. The server then calculates the round-trip time.

Fetch

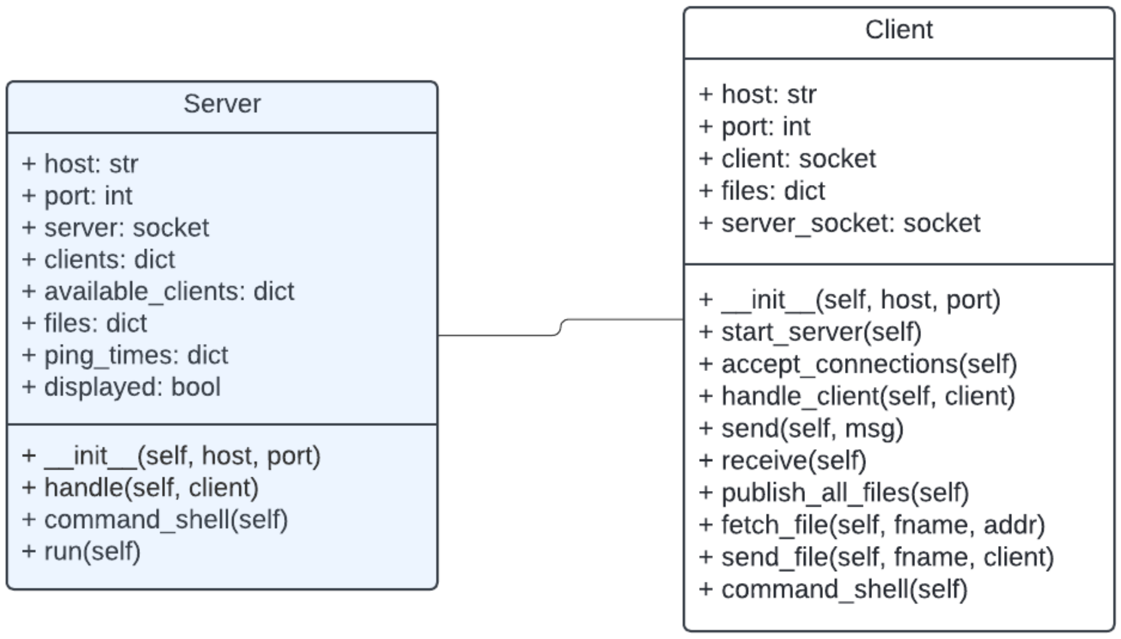
Although fetching is being done directly between two clients, the requested client still need to send a request to the server, the main job of the server in this case is to identify some other clients who store the requested file and sends their identities to the requesting client. So when a client want to fetch a file, it first need to send a 'fetch' command to the server. The server responses to this request by searching its own dictionary that contains all the available files for fetching and send these information to the client.

2.2.2 Between client and client

Fetch

After the client has receive a list of clients that have the requested file, the client can choose an appropriate client by using the 'fetch' command again but in this case the client also has to specify the ip address and the port of the client that its want to fetch the file from. After a client is chosen, a socket will be created using the TCP/IP protocol, this socket is used to connect to the client's server that owns the file. A request then is sent to the client that owns the file, the client will reply to this by sending the requested file to the other client.

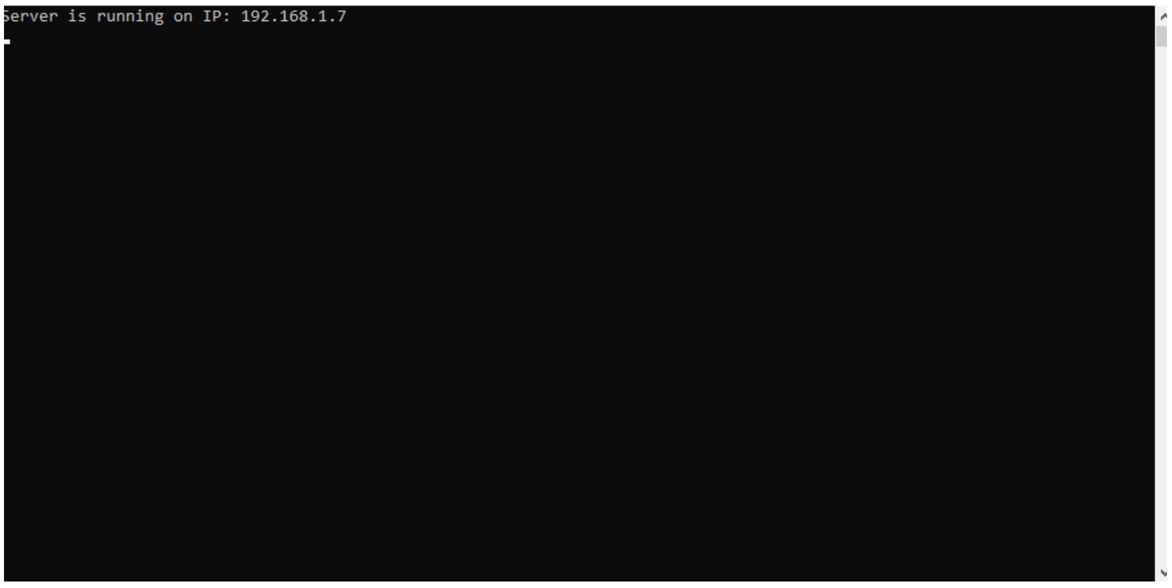
2.4 Class diagram



2.5 Validation

2.5.1 Server connection

The server is able to start and wait for connection. The server also prints its ip address on the command prompt.

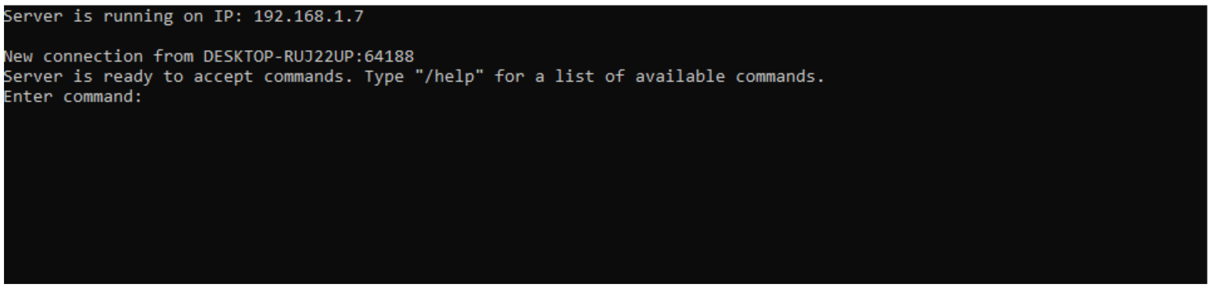


```
Server is running on IP: 192.168.1.7
```

Server CLI

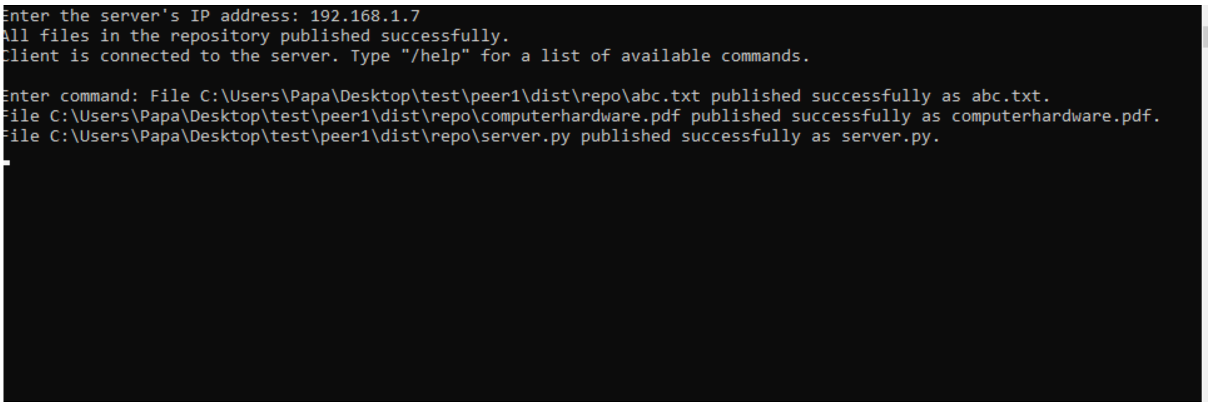
2.5.2 Client Connection

The client can connect to the server by typing the correct ip address of the server. If the client successfully connect to a server, both the server and the client will notify this to the user.



```
Server is running on IP: 192.168.1.7

New connection from DESKTOP-RUJ22UP:64188
Server is ready to accept commands. Type "/help" for a list of available commands.
Enter command:
```

Server CLI



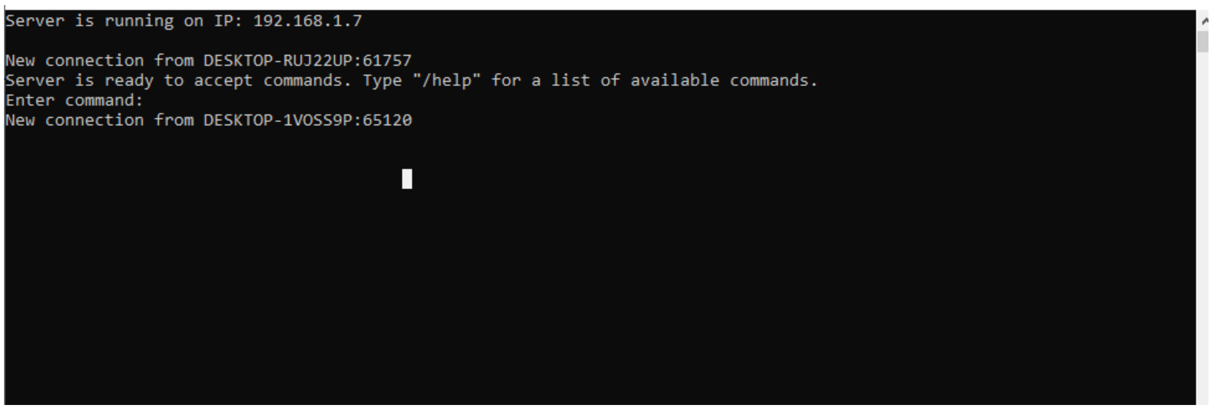
```
Enter the server's IP address: 192.168.1.7
All files in the repository published successfully.
Client is connected to the server. Type "/help" for a list of available commands.

Enter command: File C:\Users\Papa\Desktop\test\peer1\dist\repo\abc.txt published successfully as abc.txt.
File C:\Users\Papa\Desktop\test\peer1\dist\repo\computerhardware.pdf published successfully as computerhardware.pdf.
File C:\Users\Papa\Desktop\test\peer1\dist\repo\server.py published successfully as server.py.
```

Client CLI

You can also notice that when connection to the client is successful, the client will also automatically publish all the files in its local repository.

2.5.3 Concurrency

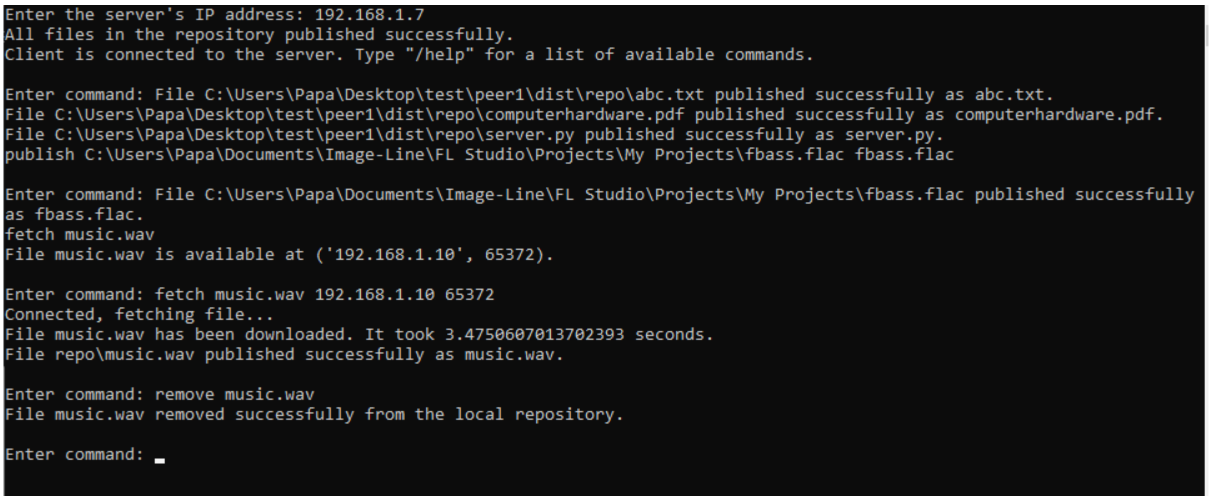
The server is able to handle multiple clients at the same time.



```
Server is running on IP: 192.168.1.7

New connection from DESKTOP-RUJ22UP:61757
Server is ready to accept commands. Type "/help" for a list of available commands.
Enter command:
New connection from DESKTOP-1VOSS9P:65120
```

2.5.4 Basic operation on the client

On the client, some basic operation can be performed: publish, fetch and remove.

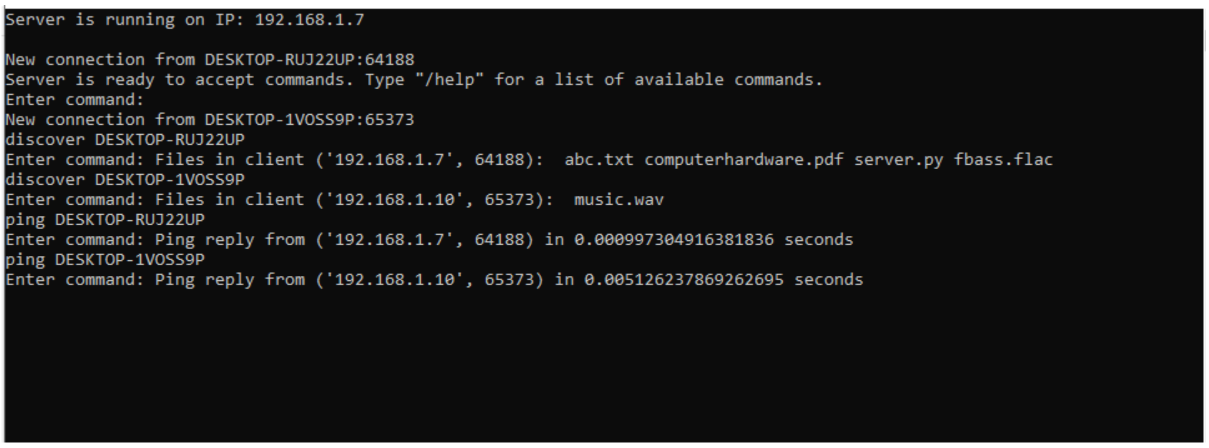


```
Enter the server's IP address: 192.168.1.7
All files in the repository published successfully.
Client is connected to the server. Type "/help" for a list of available commands.

Enter command: File C:\Users\Papa\Desktop\test\peer1\dist\repo\abc.txt published successfully as abc.txt.
File C:\Users\Papa\Desktop\test\peer1\dist\repo\computerhardware.pdf published successfully as computerhardware.pdf.
File C:\Users\Papa\Desktop\test\peer1\dist\repo\server.py published successfully as server.py.
publish C:\Users\Papa\Documents\Image-Line\FL Studio\Projects\My Projects\fbass.flac fbass.flac

Enter command: File C:\Users\Papa\Documents\Image-Line\FL Studio\Projects\My Projects\fbass.flac published successfully
as fbass.flac.
fetch music.wav
File music.wav is available at ('192.168.1.10', 65372).

Enter command: fetch music.wav 192.168.1.10 65372
Connected, fetching file...
File music.wav has been downloaded. It took 3.4750607013702393 seconds.
File repo\music.wav published successfully as music.wav.

Enter command: remove music.wav
File music.wav removed successfully from the local repository.

Enter command: _
```

Client CLI

2.5.5 Basic operation on the server

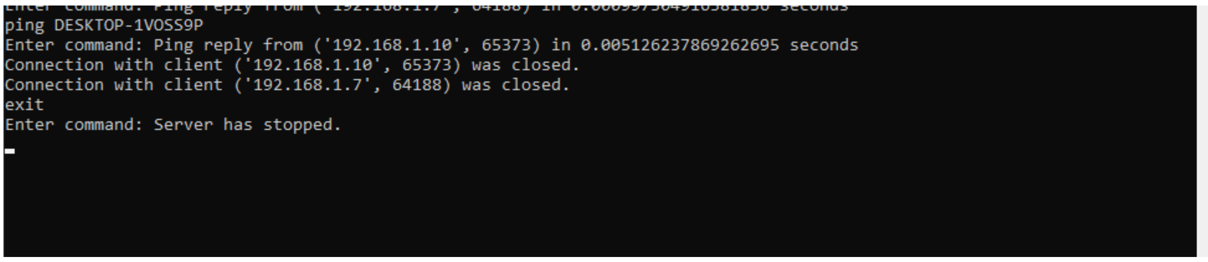
On the server, some basic operation can be performed: discover, ping



```
Server is running on IP: 192.168.1.7

New connection from DESKTOP-RUJ22UP:64188
Server is ready to accept commands. Type "/help" for a list of available commands.
Enter command:
New connection from DESKTOP-1VOSS9P:65373
discover DESKTOP-RUJ22UP
Enter command: Files in client ('192.168.1.7', 64188):  abc.txt computerhardware.pdf server.py fbass.flac
discover DESKTOP-1VOSS9P
Enter command: Files in client ('192.168.1.10', 65373):  music.wav
ping DESKTOP-RUJ22UP
Enter command: Ping reply from ('192.168.1.7', 64188) in 0.000997304916381836 seconds
ping DESKTOP-1VOSS9P
Enter command: Ping reply from ('192.168.1.10', 65373) in 0.005126237869262695 seconds
```

Server CLI

2.5.6 Closing the client and stop the server



```
Enter command: Ping reply from ('192.168.1.7', 64188) in 0.000997304916381836 seconds
ping DESKTOP-1VOSS9P
Enter command: Ping reply from ('192.168.1.10', 65373) in 0.005126237869262695 seconds
Connection with client ('192.168.1.10', 65373) was closed.
Connection with client ('192.168.1.7', 64188) was closed.
exit
Enter command: Server has stopped.
_
```

Server CLI

2.6 Performance evaluation

On the server side, basic operation like discover and ping can be executed very quick.

On the client side, basic operation like publish, remove can be performed really quick. On the other hand, operation like fetch can takes some times depend on the connection speed and the file size.

Here's the summary of some of the test on file fetching:

File name	File Size(MB)	Time(s)
music.mp3	2.29	0.26556
computerhardware.pdf	10.5	1.09348
music_large.wav	41.6	4.20213
large_zip.zip	165	17.52713

2.7 Extension function

Beside some of the required functions like discover, ping, publish and fetch. Some extension functions was also implemented.

The client can remove a file and delete from the client's repository by the command **remove <filename>.** This function has the same protocol as publish: it will first remove the file from the self.files dictionary on the client, after that it will send a 'remove' command to the server along with the filename. The server then removes the file from the self.files dictionary., after that the server will send a message back to the client as a confirmation.

The client will also automatically publish all files in the local repository when it connects to the server, this is achieved by the function **publish_all_files(),** this function will also check whether the folder for the repository exists or not and will create one if the folder can not be found.

The function **get_open_port()** is for getting a free port, this port is then being used in creating the server on the client. This makes no different when the application is run on different machines as different machines will have different ip addresses so you can have the same port number, but if you want to run this application on the same machine, each time you open a new client you will need a different port for the server on the client.

2.8 Summary of achieved results

The application is able to run on Windows, other operating systems have not been tested yet.

2.8.1 Server

- The server is able to keep track of clients, availables clients and the files they store.
- The server can quickly execute the commands **discover <hostname>** and **ping <hostname>**, the server can also be closed by typing **exit** or **stop**

2.8.2 Client

- The client is able to keep track of the files they store in the local repository, this information can also be conveyed to the server.
- The server can quickly execute the commands **publish <lname> <fname>**, **fetch <fname>**, **fetch <fname> <ip> <port>**, and **remove <fname>,** the client can also stop the server on the client and quit by typing **exit** or **stop.**
- The server on the client can send files to the other requested clients.

2.8.3 Command-shell interpreter

- Simple and easy to use, the user can also use a command to list out all available commands to the command-shell.

- The printed results on the command-shell are not always perfect in term of presentation, sometimes multiple results may be displayed on the same line. For example, a result may be displayed on the same line as the line asking for command input.

2.9 User manual

The application will have two executables: Client.exe and Server.exe. To use the application, follow these steps:

1. **Start the server**: this can be done by simply open the server executable, the server must be started first before the client.
2. **Start the client**: this can be done by simply open the client executable. Remember to do this after the server has already been started
3. **Input the server address on the client**: The server will print its address to the command prompt, use this address to connect the client to the server.
4. **Input the desired command on eithr the client or the server**: The users can type **/help** for a list of available commands.

   Available commands for the server:

   - **discover <hostname>** - Discover files on the specified host'
   - **ping <hostname>** - Ping the specified host'
   - **exit, stop** - Stop the server'

   Available commands for the client:

   - **publish <localname> <filename>** - Publish a file to the server'
   - **fetch <filename>** - Search whether a file is available to be fetched'
   - **fetch <filename> <ip> <port>** - Fetch a file from a specified client'
   - **remove <filename>** - Remove a file from the server'
   - **exit, stop** - Stop the program'

3 Conclusion

In conclusion, the assignment involved designing and implementing a simple file-sharing application using the TCP/IP protocol stack. The application comprised of a centralized server and multiple clients, with the server maintaining a record of connected clients and the files they store.

Clients were designed to inform the server about the files in their local repository without transmitting the actual file data. When a client required a file not present in its repository, it sent a request to the server, which then identified other clients storing the requested file and relayed their identities to the requesting client. The file was then fetched directly from the identified client, without requiring any server intervention.

The system was designed to handle multiple clients downloading different files from a target client simultaneously, necessitating the client code to be multithreaded. Both the server and the client were equipped with a simple command-shell interpreter to accept commands.

The project provided valuable insights into the workings of peer-to-peer file sharing systems, network programming, multithreading, and the TCP/IP protocol stack. It also highlighted the need for robust error handling and the importance of efficient communication protocols in networked applications. The system could be further improved by adding features like encryption for secure file transfers, and a graphical user interface for ease of use. Overall, it was a challenging yet rewarding experience that offered practical exposure to network programming concepts.