Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

Department of Computer Engineering - Semester 231

ASSIGNMENT REPORT

Computer Network (C03093)

Name:	*Student's ID:*
Nguyen Thanh Thao Nhi	2152840
Le Thanh Binh	2152897
Phan Le Khanh Trinh	2151268
Nguyen Phuc Toan	2153902

Instructor:

Nguyen Manh Thin

Contents

1 Introduction

Peer-to-peer file sharing systems have become more popular for sharing, exchanging and files transferring among many edge computer network devices over the Internet. In peer-to-peer network, a central point, in other word is server, is not necessary. This network is being invested with an enormous amount of financial and human resources around the world. For instance, Napster, Gnutella, and Freenet are some of the most popular peer-to-peer networking. Most of the Internet traffic is due to file sharing systems, Napster uses a server to communicate between the users, and each users should contact the server in order to get the data. Gnnutella, however, depends on the client/server approach where the peer sends a query to all peers inn the network.

In the scope of this paper, a simple peer-to-peer network file-sharing application will be implemented, utilizing the TCP/IP protocol for communication between peers within the local network. The paper will be divided into four parts: Theoretical Foundation, Application Design, Function Description, and a User Manual, which will be provided to readers.

2 Theoretical Foundations

2.1 Socket programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. The concept of sockets emerged as part of the Berkeley Software Distribution (BSD) in the early 1980s, integrated into the Unix operating system. This was a significant development in network programming and has since become a core component of many operating systems and programming frameworks.

Here's a basic overview:

1. **Sockets:** These are the endpoints in a network communication. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to.

2. **Server Socket:** This is set up to listen for incoming client connections. It typically runs on a specific computer and has a port number associated with it.

3. **Client Socket:** This is created by the client to establish a connection to the server socket.

4. **IP Address and Port:** For network communication, a distinct address is necessary. The IP address pinpoints a specific network interface, while the port number denotes

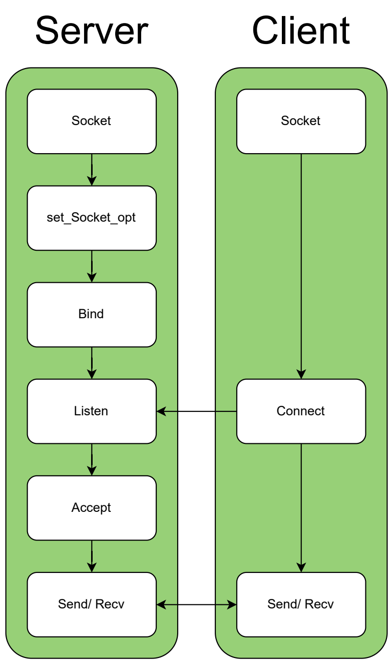

Figure 1: The entire process of socket programming.

the application or service on that computer.

5. **Establishing Connection:** The client initiates a connection request to the server socket, which the server acknowledges, thus creating a new socket on the server side for this connection.

6. **Transferring Data:** After establishing the connection, data can be exchanged between the connected sockets.

7. **Closing the Connection:** Once the data exchange is complete, it's essential to close the sockets at both ends to terminate the communication.

2.2 Multi-thread programming

Multithreaded programming is a programming concept used to enable multiple threads to execute concurrently within a single program. A thread is the smallest unit of processing that can be scheduled by an operating system.

1. **Simultaneous Execution:** This approach allows different parts of a program to run simultaneously, leading to more efficient use of the CPU by minimizing idle time.

2. **Threads:** A thread is a sequence of instructions within a program that can be executed independently of other code. Imagine a thread as a worker assigned to perform a specific task in a factory.

3. **Benefits:** The primary benefit of this method is better performance and quicker response times. For example, a web server using multiple threads can handle many requests at once, improving overall efficiency.

4. **Complexities:** Multithreading can be complex to implement correctly. Issues such as deadlocks (where two threads are waiting for each other indefinitely) and race conditions (where the outcome is dependent on the sequence or timing of threads) need careful handling.

5. **Applications:** It's commonly used in scenarios like server applications, intensive computational tasks, real-time processing, and any application needing quick responsiveness.

6. **Programming Support:** Many programming languages, including Java, C#, and Python, offer support for multithreading, often through specialized libraries that help manage threads and deal with synchronization.

7. **Closing the Connection:** Once the data exchange is complete, it's essential to close the sockets at both ends to terminate the communication.

3 Application design

3.1 System design

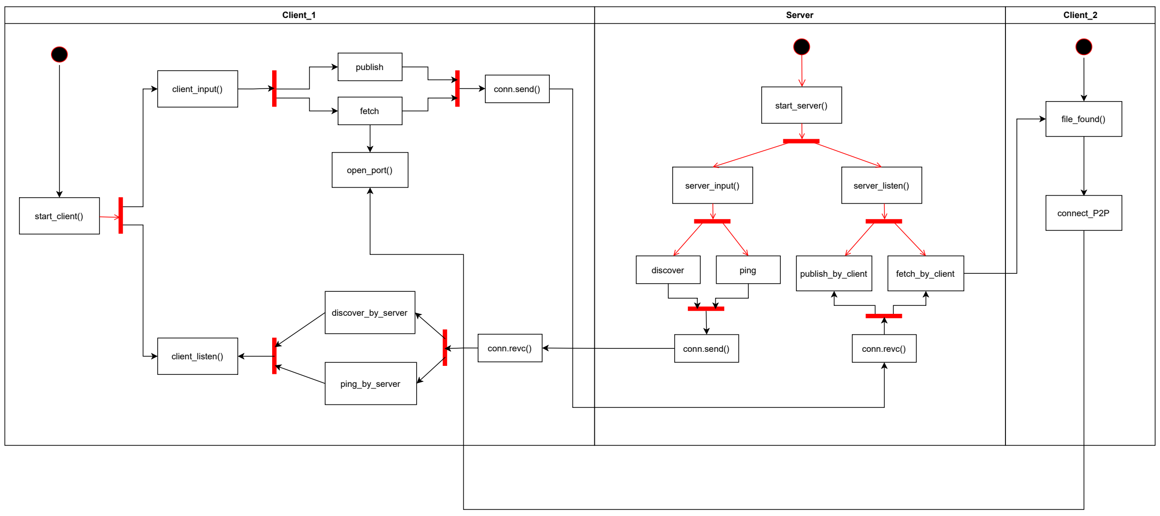

Figure 2: The activity diagram for the whole system.

The diagram above illustrates how the application system running, each use-case scenario can be implemented by several function in the source code.

There are three entities: **server**, **client1** and **client2**. Because the application also perform the P2P protocol so it is necessary to have another client corrupts to the diagram, despite the fact that server handles **multiple** client.

Although this is a P2P application, this network require server online due to the centralized data such as: list of published files, number of clients online in the server, kept in server storage.

3.2 Peer-to-Peer architecture

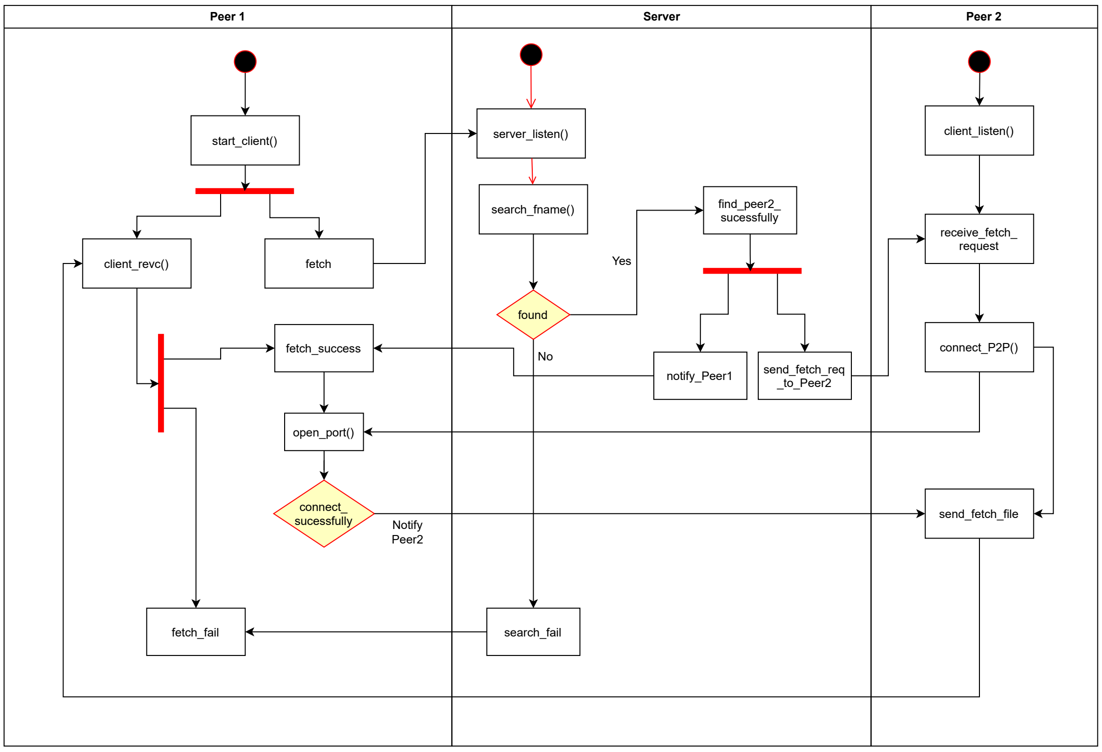

Figure 3: How peer connect to peer for fetch.

The above diagram indicates that when a client requests to fetch a file, it sends the filename (fname) to the server and opens a port, waiting for the server to store the desired file. Then, the server will check in the public folder, send a message to the client that is requesting to fetch that file, and the requesting client will connect to the one that is waiting.

Clients are also coded as multithread for multiple peer connect and fetch files at the same time.

4 Manual Document

Following these step-by-step procedures of this procedures to deploys the file-sharing application.

4.1 For server set-up and guideline

Step 1: Create a public folder

Create a folder name ***public*** on the server computer. Then copy the ***public*** folder path to *public_path* in ***server.py*** file. This folder will keep track of which client has publish what file name. Be aware that this folder does not content the file data that client has published.

Step 2: Find server IP-v4 address

In order for clients connect to the sever, check IP-v4 address in Termial and give it to clients which want to connect to that server. To check the IP-v4 address, on Windows, open Terminal and type: ***ipconfig***.

Note: Make sure all the server and clients are connected on a same Wifi network. Because our server static IP address is not public yet on the Internet so these instruction to make sure the application works well.

Step 3: Deploy server

Compile and run **server.py** file. Then wait for clients connect to the server.
If server is created successfully, it will display:

```
Server is listening on localhost: 12345
```

When a new client connected to the server, it will have the IP-address and port number of the client connect via socket.

```
Connected to ('127.0.0.1', 58953)
```

Step 4: Server command:

- **discover *hostname***: discover the list of local files of the host named hostname.
- **discover *all***: discover all the local files that every user has published
- **ping *hostnname***: live check if the hostname is still in connection or not.

hostname in this case is the client's IP address, it will appear to the server when a new client connect to the server.

Step 5: To disconnect the server Kill the terminal and all the client will be disconnected.

4.2 From client set-up and guideline

Step 1: Enter server IP address

Enter IP address given by server to establish TCP/IP connection between client and server.

Step 2: Connect success

Deploys the ***client.py*** file. If the client connect to the server successfully, it will show for instance

'Connected to ('192.168.1.4', 58883)'

, which is the IPv4 address and port number of the host running server. Then the server will get the information of the new client. This application allow multiple clients connect to the server so feel free to setup and run as many as possible number of client connect to the same server.

Step 3: Client command: fetch publish

- **publish *lname fname***: to send to the server client's file name ***(fname)*** and where it is located in client's files system ***(lname)***.
- **fetch *fname***: to fetch the file-name ***fname*** in the network and save it in to the local repository.

When the same client publishes a file that was previously published, the server will delete the old file and retain the newly published one. If multiple clients attempt to publish a file with the same filename, the server will automatically append a counter at the end of the file-name and store it in the server's data storage. For instance: *filename(1).txt, filename(2).txt*

During fetching process, if the connection is interrupted, the client will have to fetch the file at the beginning.

Note:

- fname: must inlcude the valid file name and file extention.
- lname: local repository
- **Client and server do not deploy on a same computer.**

4.3 Correct command example

- **ping** 127.0.0.1
- **discover** 127.0.0.1
- **discover** all

- **fetch** output.txt
- **publish** C:\User\Documents\ output.txt

5 Source code

Visit this link for source code: https://github.com/kthkhanhtrinh/File_Transfer-P2P_Application