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

COMPUTER NETWORK (CO3094)

Assignment 1

"Develop a network application"

Lecturer:	Mr. Nguyễn Mạnh Thìn
Group:	CC04
Students:	Nguyễn Hoàng Long - 2153530
	Bùi Đăng Khoa - 2152668
	Lê Trọng Kiên - 2153494
	Nguyễn Anh Khang - 2053097

HO CHI MINH CITY, Dec 2023

Contents

1 Requirement Analysis

Centralized server: keeps track of which clients are connected and storing what files. In other words, the application will need a server have a feature of knowing the lively connecting clients and the files that they publish on the server.

Client-server interaction :

- A client informs the server as to what files are contained in its local repository but does not actually transmit file data to the server. This means clients will announce the server the location or name of the file that he/she is keeping on his/her own computer. They cannot upload the whole file data on to the server in this app.
- When a client requires a file that does not belong to its repository, a request is sent to the server. The server identifies some other clients who store the requested file and sends their identities to the requesting client. The client will select an appropriate source node and the file is then directly fetched by the requesting client from the node that has a copy of the file without requiring any server intervention. This mean client will have the right to choose any available peer to get the file downloading.

Multithreading code : Multiple clients could be downloading different files from a target client at a given point in time. This shown that the each comming request should be assigned to a single thread to run their own downloading process or in others word, the client code will be multithreaded.

Client's command-shell interpreter :

- ***publish lname fname :*** On the client's computer, the file has the original name is *lname*, then they want to publish their file to the server with another file name *fname*. Another client can ask for fetching the file with name is *fname*.
- ***fetch fname :*** When a client want to download the file with the name *fname*, they ask for the server who is keeping this file, the server then return a list of available peers for the client to contact and download.

Server's command-shell interpreter :

- ***discover hostname :*** discover the list of local files of the host named hostname. The server can keep track all published files of the client name "*hostname*".
- ***ping hostname :*** live check the host named hostname. The server can see if the client name "*hostname*" are still connecting with the server or not.

In this implementation, we choose Python as the programming language with support library "Tkinter" to make the GUI (Graphic User Interface).

2 Application Design

2.1 Architecture

The our designed file-sharing Peer-to-peer application :

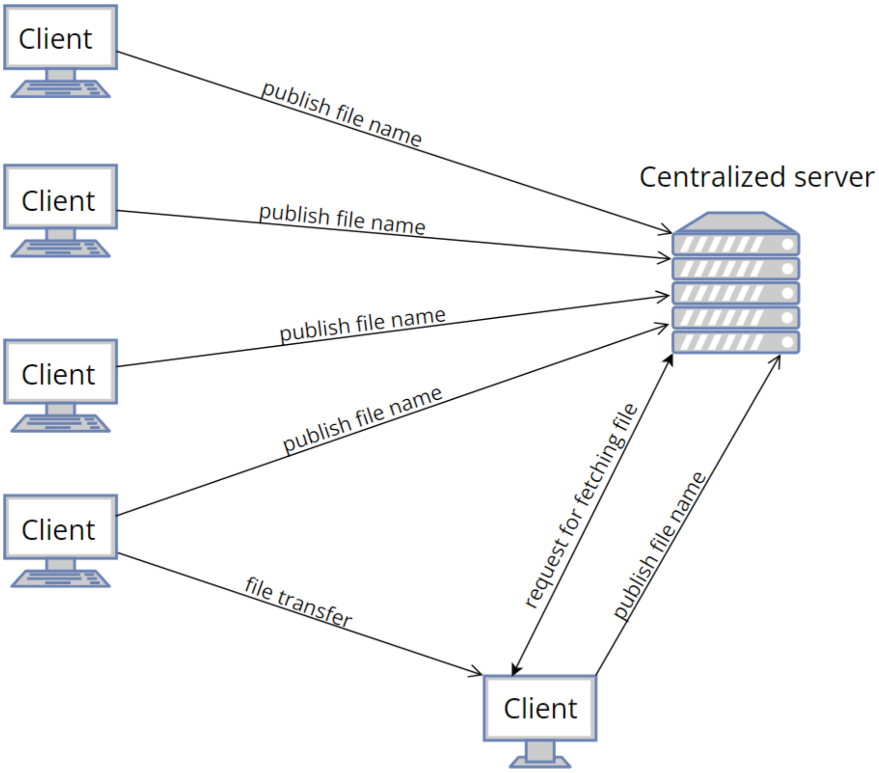


Figure 1: The Peer-to-peer network model

There are two actors in this model : Centralized server and client

Working flow:

- **publish file name** : The clients publish their file's name to server (they can also replace with another name if they want). The server will store that information.
- **Request for downloading file** : The clients request the server for file. Since the server has all the information of its peers, so it returns the IP addresses of all the peers having the requested file to the client. Then let the client chooses one of them .
- **Interaction between two clients** : The transfer file process take place between 2 peers : the requesting file client connect to the one that server feedback. Then the later will send the file data to the former.

2.2 Class diagrams

The our class diagram :

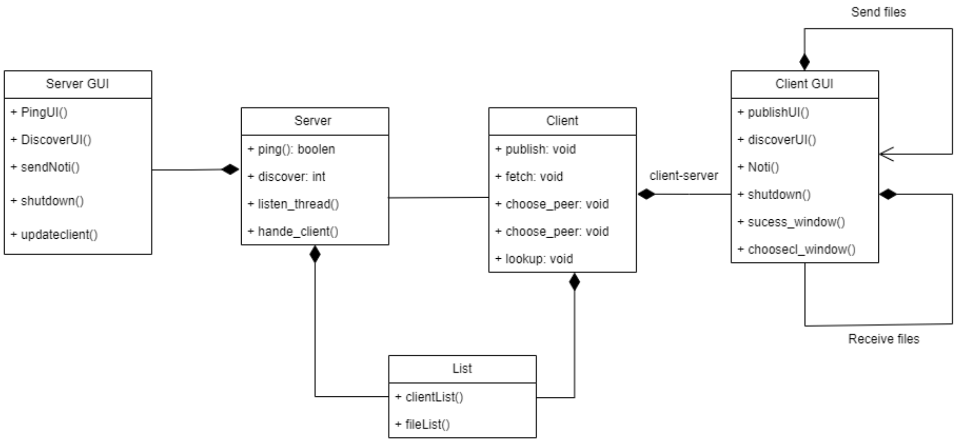


Figure 2: Class diagram of the system

3 Describe functions

3.1 Server functions

3.1.1 Some supporting functions

3.1.1.a *Init* function

The following code segment is for setting some global variables to use for others functions.

```
    def __init__(self, output_text, host='', port=9999):
        #take host name
        self.hostname = host if host else socket.gethostname()
        #set the port to listen on
        self.port = port
        #this peer_records is to save the peer's list of published
 files
        self.peer_records = defaultdict(set)
        #this rfc_records is to save the list of peers can share the
 file
        self.rfc_records = {}
        #create a threading lock
        self.lock = threading.Lock()
        #save the online clients
        self.connecting_clients = []
        self.output_text = output_text  # Store the output_text widget

```

3.1.1.b *start* function

The following code segment is to create the server socket (using IPv4 address and TCP protocol), and then using threads to serve client and run its own command (ping,discover) at a time:

```
def start(self):
  try:
      #Create a listening socket using IPv4 (AF_INET) and TCP (SOCK_STREAM)
      self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      #Bind the socket to its hostname and port
      self.socket.bind((self.hostname, self.port))
      #Listen for up to 5 client at once (setting the max number for serving)
      self.socket.listen(5)
      print(f'The server is listening on port {self.port}')
      #Use 2 worker threads :
      with ThreadPoolExecutor(max_workers=2) as executor:
          #server_loop here is function to listen each incomming connecting client
          executor.submit(self.server_loop, print_to_ui=True)
          #Another thread if to run for ping and discover command
          executor.submit(self.interactive_shell, print_to_ui=True)
  except KeyboardInterrupt:
      print('\nShutting down the server..\nGoodbye!')
      self.socket.close()
  except SystemExit:
      os._exit(0)
```

3.1.1.c *server_loop* function (multithreaded code)

The following code segment shows the multithread code: for each *while* loop, there is 1 created thread assigned to 1 client for serving (it will get to *handle_client* function). Also, when client connect to the server, the server will record connecting_clients for that online client (this client will be remove from the record if he/she is offline - shown in the *handle_client* function)

```
def server_loop(self, print_to_ui=False):
      while True:
          #listen to connecting client, accept and save its info
          client_socket, client_address = self.socket.accept()
          hostname, _, _ = socket.gethostbyaddr(client_address[0])
          #record live client info
          self.connecting_clients.append({'hostname': hostname, 'port': client_address[1], 'socket': client_socket})
          if not print_to_ui:
              print(f'\nClient hostname:{hostname}\nThrough port:{client_address[1]} connected\nEnter your favorite command (discover,ping,shutdown) on the next line:')
          thread = threading.Thread(
              target=self.handle_client, args=(client_socket, client_address, print_to_ui))
          thread.start()
```

3.1.1.d *handle_client* function

The following code segment shows: The server receives the request from client like publish file: save file info and corresponding publisher info to peer_records and rfc_records (through *add_record* function); or fetch file: respond a list peers keeping that file to client to connect with whom he/she likes. Besides, when the server try failed to connect with client, it will remove the online client from the aforementioned record above and delete the relating client's information also.

```python
def handle_client(self, client_socket, client_address, print_to_ui=False):
    peer_host = None
    peer_port = None
    try:
        hostname, _, _ = socket.gethostbyaddr(client_address[0])
        self.connecting_clients.append({'hostname': hostname, 'port': client_address[1], 'socket': client_socket})
        message = f"\nClient hostname: {hostname}\nThrough port: {client_address[1]} connected \n"
        if print_to_ui:
            self.update_output(message)  # Display notification in UI

        while True:
            request = client_socket.recv(2048).decode()
            if print_to_ui:
                self.update_output(f'\nReceived request:\n{request}\n')  # Display notification in UI
            lines = request.splitlines()
            method = lines[0].split()[0]
            if method == 'PUBLIC':
                peer_host = lines[1].split(None, 1)[1]
                peer_port = int(lines[2].split(None, 1)[1])
                file_name = lines[3].split(None, 1)[1]
                real_file_name = lines[4].split(None, 1)[1]
                self.add_record(client_socket, (peer_host, peer_port), real_file_name, file_name)
                if print_to_ui:
                    self.update_output("\n")  # Display file published notification in UI
            elif method == 'FIND':
                file_name = lines[3].split(None, 1)[1]
                self.get_peers_of_rfc(client_socket, file_name)
                if print_to_ui:
                    self.update_output("\n")  # Display file fetched notification in UI
            else:
                raise AttributeError('Method Not Match')
    except ConnectionError:
        if print_to_ui:
            print(f'{hostname} via port {client_address[1]} left the server\n')
            if peer_host and peer_port:
                self.clear_data(peer_host, peer_port)
            message = f'{hostname} via port {client_address[1]} left the server\n'
            self.remove_connecting_client(client_socket)
            self.update_output(message)
    except BaseException:
        try:
            client_socket.sendall(str.encode(f'Status  400 Bad Request\n'))
        except ConnectionError:
            hostname, _, _ = socket.gethostbyaddr(client_address[0])
            if print_to_ui:
                print(f'Host {hostname} through {client_address[1]} port left')
                self.remove_connecting_client(client_socket)
                if peer_host and peer_port:
                    self.clear_data(peer_host, peer_port)
                message = f'Host {hostname} through {client_address[1]} port left\n'
                self.update_output(message)
    finally:
        self.remove_connecting_client(client_socket)

```

3.1.2 "*discover_command*" function

The following code segment shows checking in the peer_records to see list of file the client has published.

```
    def discover_command(self, hostname):
        output = f"File available on {hostname} is:\n"
        i = 1
        file_list = []
        for (host, port), files in self.peer_records.items():
            if host == hostname:
                for (_, file_title) in files:
                    file_list.append(f"{i}. {file_title}\n")
                    i += 1
        if i == 1:
            file_list.append("Not any file at all\n")
        return output + "".join(file_list)

```

3.1.3 "*ping_ command*" function

The following code segment shows the checking in the connecting_clients record to see whether he/she is still online or not.

```
def ping_command(self, hostname):
        check = any(client['hostname'] == hostname for client in self.
    connecting_clients)
        if check:
            return f"{hostname} is live now\n"
        else:
            return f"{hostname} is off now\n"

```

3.2 Client functions

3.2.1 Some supporting functions

3.2.1.a *Init* function

The following code segment is for setting some global variables. Also create for client a folder to work with (file publish and sharing space).

```
def __init__(self, server_host='0.0.0.0', directory='File_Sharing')
    :
        self.SERVER_HOST = server_host
        self.SERVER_PORT = 9999
        self.DIRECTORY_FILE = directory
        Path(self.DIRECTORY_FILE).mkdir(exist_ok=True)
        self.MY_PORT = None
        self.SHARING_MODE = True
        self.ui = None

```

3.2.1.b *start* function

The following code segment is to create a socket (using IPv4 and TCP also) to connect with server. Similarly to the server, it will have the thread to address for another peer's sending file request (*init_ sending* function) and another thread is to work with server :

```
def start(self):
        print('Connecting to the server %s:%s' % (self.SERVER_HOST,
    self.SERVER_PORT))
        self.server_socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
        try:
            self.server_socket.connect((self.SERVER_HOST, self.
    SERVER_PORT))
        except Exception:
            print('Server is not available.')
            return
        print('Connected to the server !')
        uploader_thread = threading.Thread(target=self.init_sending)
        uploader_thread.start()
        while self.MY_PORT is None:
            pass  # Wait until upload port is initialized
        print('Listening on the port %s' % self.MY_PORT)
        self.interactive_shell()

```

3.2.1.c *init_ sending* function (multithreaded code)

The following code segment shows creating listening socket and the multithread code: for each *while* loop, there is 1 created thread assigned to 1 peer for serving sending file request (it will get to *handle_ sending* function).

```
    def init_sending(self):
        self.uploader_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.uploader_socket.bind(('', 0))
        self.MY_PORT = self.uploader_socket.getsockname()[1]
        self.uploader_socket.listen(5)
        while self.SHARING_MODE:
            requester_socket, addr = self.uploader_socket.accept()
            handler_thread = threading.Thread(target=self.handle_sending, args=(requester_socket, addr))
            handler_thread.start()
        self.uploader_socket.close()

```

3.2.1.d *handle_ sending* function

The following code segment shows: The client receives the message containing the local file path, he/she will check whether this information is valid or not, then it calls the *sending_file* function to send the file to the requesting peer.

```
    def handle_sending(self, requester_socket, addr):
        header = requester_socket.recv(2048).decode().splitlines()
        try:
            realname = header[0].split()[2]
            method = header[0].split()[0]
            file_path = f'{self.DIRECTORY_FILE}/{realname}'
            chekc = '%s/%s' % (self.DIRECTORY_FILE, realname)
            # print(chekc)
            if not Path(file_path).is_file():
                self.after(0, lambda: self.ui.update_output(f'Status 404 Not Found'))
            elif method == 'GET':
```

```
            response_header = self.create_response_header(file_path)
            requester_socket.sendall(response_header.encode())
            self.sending_file(requester_socket, file_path)
            # Restore command line
            self.after(0, lambda: self.ui.update_output('\n'))# Enter your command (e.g., "fetch fname," "publish lname fname," "shutdown"):
        else:
            raise P2PClientException('Bad Request.')
    except Exception:
        self.after(0, lambda: self.ui.update_output(f'Status 400 Bad Request\n'))
    finally:
        requester_socket.close()

```

3.2.1.e *sending_file* function

The following code segment shows: The client will send all file data to the peer. (By each iteration of the loop, there are 2048 bytes to read and send to the peer)

```
def sending_file(self, requester_socket, file_path):
    try:
        print('\nSending file...')
        send_length = 0
        with open(file_path, 'rb') as file:
            to_send = file.read(2048)
            while to_send:
                send_length += len(to_send)#.encode()
                requester_socket.sendall(to_send)#.encode()
                to_send = file.read(2048)
    except Exception:
        raise P2PClientException('Sending Failed')
    print('Sending Completed.')

```

3.2.1.f *choose_peer_to_download* function

The following code segment shows: when the server responds the file message which contains all available peers can share their file (the lookup_request function is call in fetch_command function), this function allows client to choose peer for fetching that file.

```
def choose_peer_to_download(self, lines):
    print('Available peers can share the file: ')
    for i, line in enumerate(lines[1:]):
        line = line.split()
        print(f'{i + 1}: {line[-2]}:{line[-1]}')
    try:
        idx = int(input('Choose one peer to download: '))
        realname = lines[idx].split()[0]
        title = lines[idx].split()[1]
        peer_host = lines[idx].split()[-2]
        peer_port = int(lines[idx].split()[-1])
        print(realname,title,peer_host,peer_port)
    except Exception:
        raise P2PClientException('Invalid Input.')

    if (peer_host, peer_port) == (socket.gethostname(), self.MY_PORT):
        raise P2PClientException('Do not choose yourself.')

    self.download_request(title, realname, peer_host, peer_port)

```

3.2.1.g *download_request* function

The following code segment shows: after client chooses peer to download, this function will first send request for the chosen peer to fetch the file and then create, save the fetched file to the local file directory.

```
def download_request(self, title, realname, peer_host, peer_port):
    try:
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if soc.connect_ex((peer_host, peer_port)):
            raise P2PClientException('Peer Not Available')

        msg = f'GET file {realname} Request\n'
        msg += f'Host: {socket.gethostname()}\n'
        msg += f'OS: {platform.platform()}\n'
        soc.sendall(msg.encode())

        header = soc.recv(2048).decode()
        print(f'Receive response header: \n{header[0]}')
        header = header.splitlines()

        if header[0].split()[-2] == '200':
            path = f'{self.DIRECTORY_FILE}/{title}'
            print('Downloading...')
            try:
                with open(path, 'wb') as file:
                    content = soc.recv(2048)
                    while content:
                        file.write(content)#.decode()
                        content = soc.recv(2048)
            except Exception:
                raise P2PClientException('Downloading Failed')

            total_length = int(header[1].split()[1])
            if os.path.getsize(path) < total_length:
                raise P2PClientException('Downloading Failed')

            print('Downloading Completed.')
    finally:
        soc.close()

```

3.2.1.h *lookup_request* function

The following code segment shows: When client calls fetch function, this function will be called to send request to the server with the purpose that receiving the list of available peers keeping the required file response from the server.

```
def lookup_request(self, file):
        msg = f'FIND file request\n'
        msg += f'Host: {socket.gethostname()}\n'
        msg += f'Port: {self.MY_PORT}\n'
        msg += f'Title: {file}\n'
        try:
            self.server_socket.sendall(msg.encode())
            response = self.server_socket.recv(2048).decode()
        except Exception:
            #print('\nServer is not available.')
            response = "no"
        return response

```

3.2.2 "*fetch_command*" function

The following code segment shows the steps to fetch the file. First, it calls the lookup_request function to take the list of available peers keeping the required file name. Second, the choose_peer_to_download function will let client to choose peer and fetch that file.

```
def fetch_command(self, fname):
        lines = self.lookup_request(fname).splitlines()
        if lines[0].split()[1] == '200':
            self.choose_peer_to_download(lines)
            messagebox.showinfo("Fetch Success", f"Successfully fetched {fname}")
        elif lines[0].split()[1] == '400':
            raise P2PClientException('Invalid Input.')
        elif lines[0].split()[1] == '404':
            raise P2PClientException('File Not Available.')
        else:
            raise P2PClientException('\nServer is not available.')

```

3.2.3 "*publish_ command*" function

The following code segment shows the steps to publish the file. It check the file's directory to see it exists or not and after that send the message containing file's name and its replaced name to the server to collect that information.

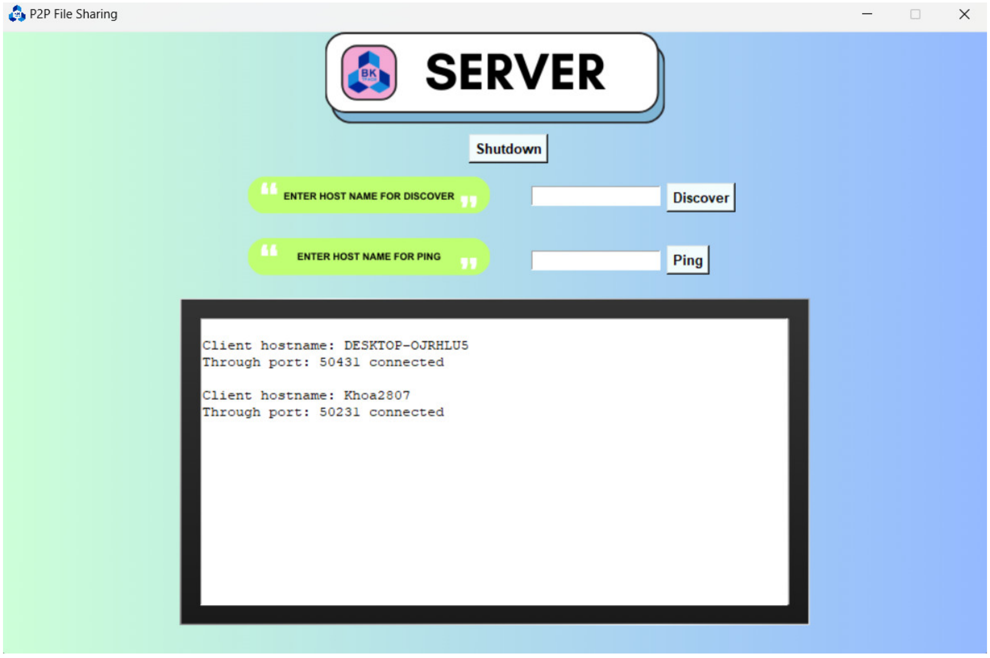
```
def publish_command(self, lname, fname):
        file_path = Path(f'{self.DIRECTORY_FILE}/{lname}')
        if not file_path.is_file():
            raise P2PClientException('Local file does not exist!')
        msg = f'PUBLIC file {fname} Request\n'
        msg += f'Host: {socket.gethostname()}\n'
        msg += f'Port: {self.MY_PORT}\n'
        msg += f'Title: {fname}\n'
        msg += f'real-Name: {lname}\n'
        try:
            self.server_socket.sendall(msg.encode())
            response = self.server_socket.recv(2048).decode()
            print(f'Receive response: \n{response}')
            messagebox.showinfo("Success",f"Successfully")
        except Exception:
                print('\nServer is not available anymore !')

```

4 Describe testing flow

4.1 Server functions

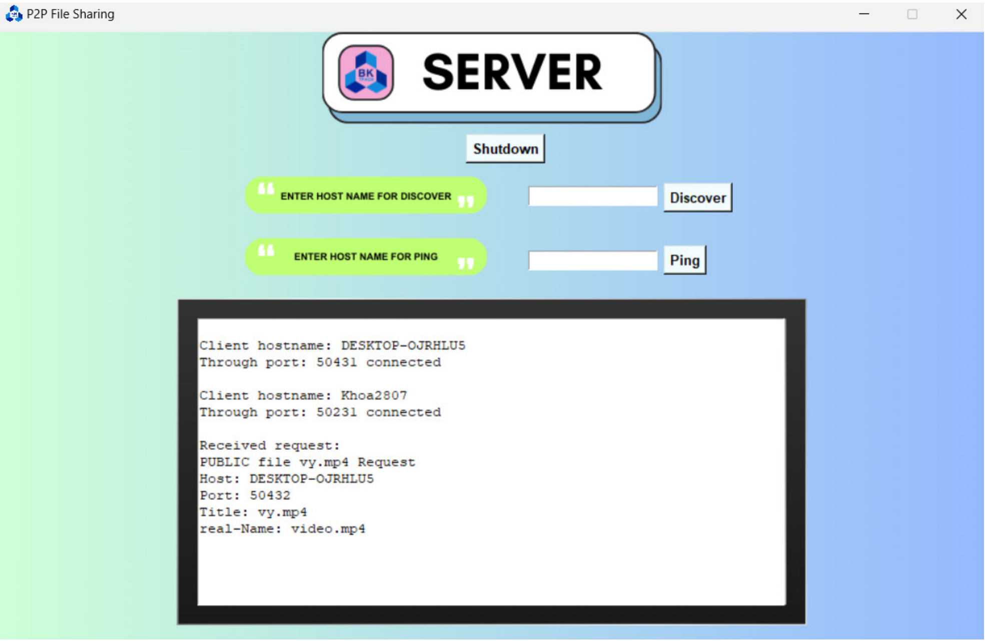
4.1.1 "*Display* notifications"

The server interface has the function of notifying system activities



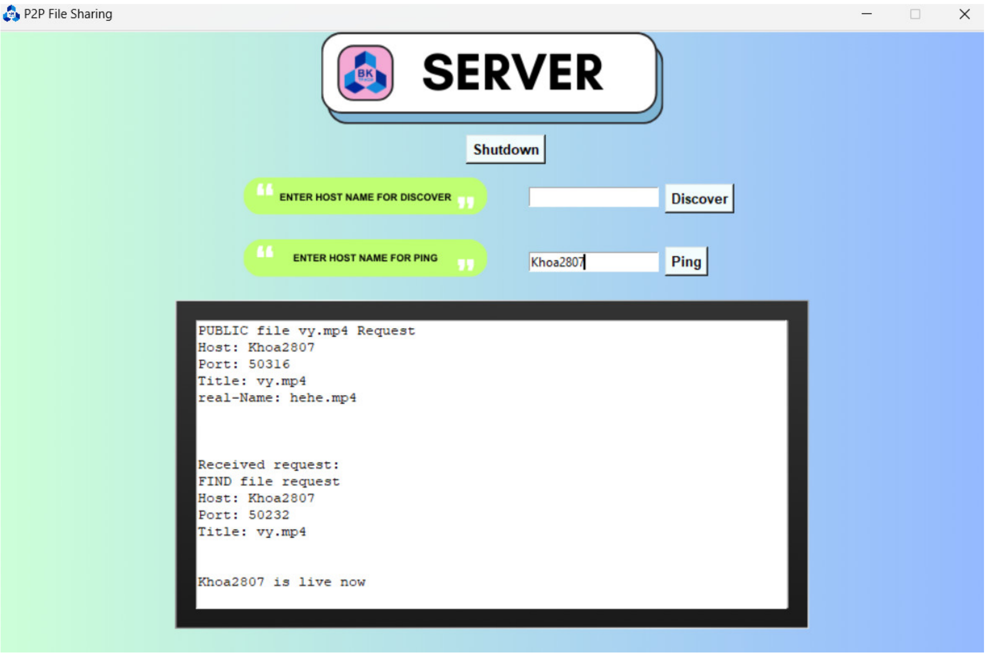
```
Client hostname: DESKTOP-OJRHLU5
Through port: 50431 connected

Client hostname: Khoa2807
Through port: 50231 connected
```

Figure 3: When a client connects to the server, it will be notified



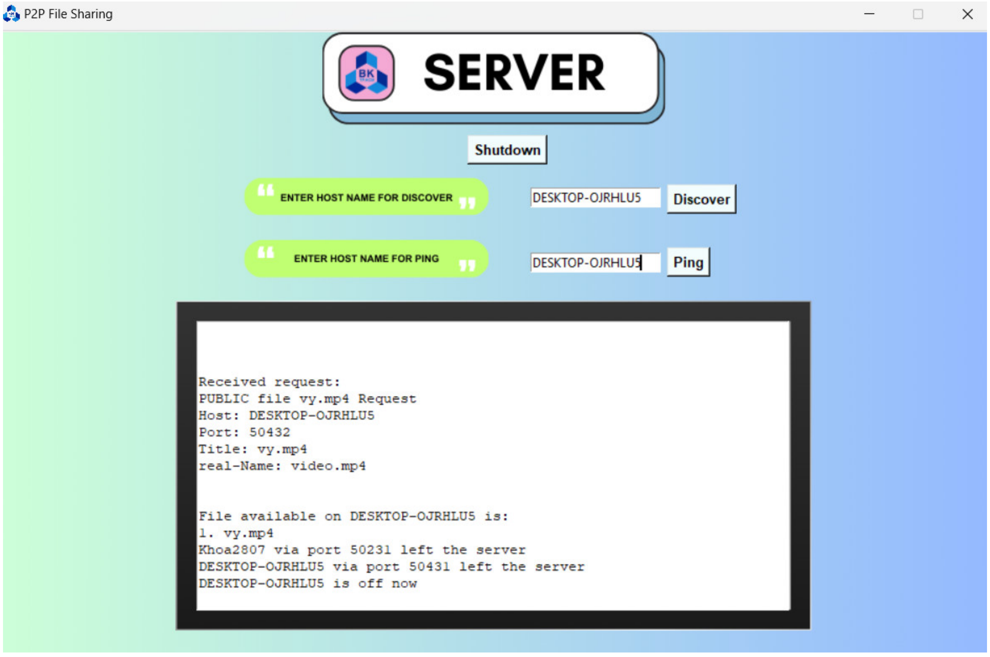
```
Client hostname: DESKTOP-OJRHLU5
Through port: 50431 connected

Client hostname: Khoa2807
Through port: 50231 connected

Received request:
PUBLIC file vy.mp4 Request
Host: DESKTOP-OJRHLU5
Port: 50432
Title: vy.mp4
real-Name: video.mp4
```

Figure 4: The system will notify when the client publishes a file



```
PUBLIC file vy.mp4 Request
Host: Khoa2807
Port: 50316
Title: vy.mp4
real-Name: hehe.mp4


Received request:
FIND file request
Host: Khoa2807
Port: 50232
Title: vy.mp4


Khoa2807 is live now
```

Figure 5: The system will notify when the client fetches a certain file

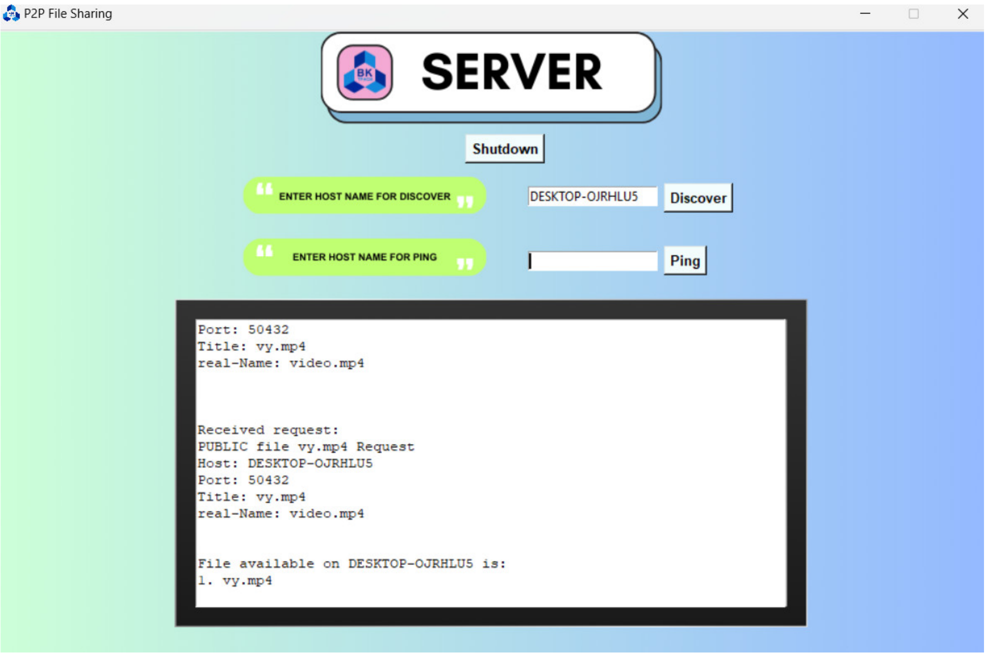


```
Received request:
PUBLIC file vy.mp4 Request
Host: DESKTOP-OJRHLU5
Port: 50432
Title: vy.mp4
real-Name: video.mp4


File available on DESKTOP-OJRHLU5 is:
1. vy.mp4
Khoa2807 via port 50231 left the server
DESKTOP-OJRHLU5 via port 50431 left the server
DESKTOP-OJRHLU5 is off now
```

Figure 6: The system will also notify immediately when any client cancels the connection

4.1.2 "*discover* hostname" command

The server interface has the function of checking files that a certain client has published. This feature is integrated in a box to enter the client name and a "Discover" button to use

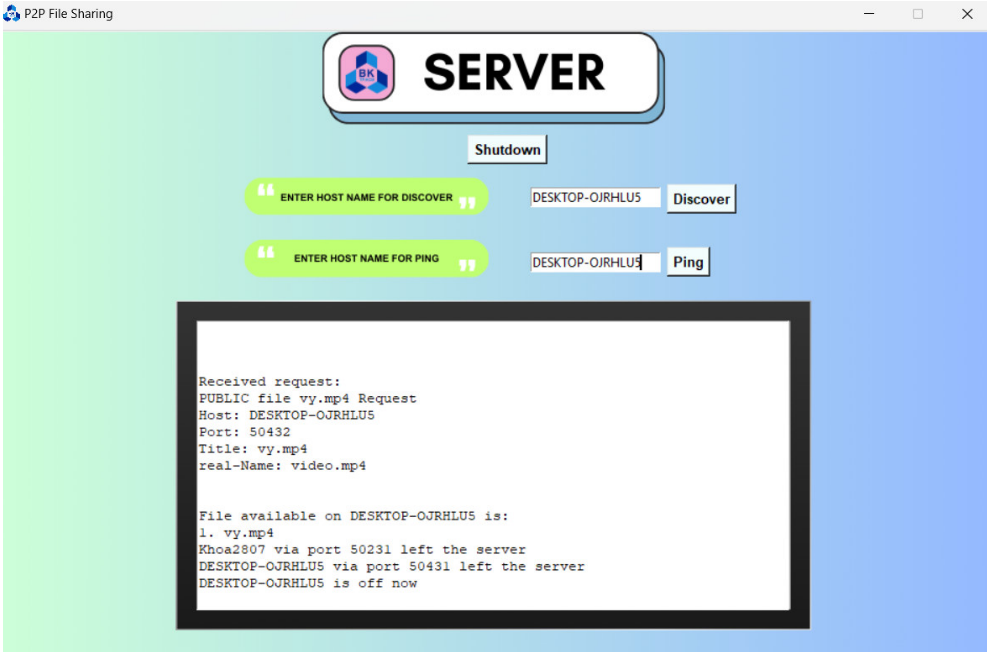


```
Port: 50432
Title: vy.mp4
real-Name: video.mp4


Received request:
PUBLIC file vy.mp4 Request
Host: DESKTOP-OJRHLU5
Port: 50432
Title: vy.mp4
real-Name: video.mp4


File available on DESKTOP-OJRHLU5 is:
1. vy.mp4
```

Figure 7: Notification when performing the discover feature

4.1.3 "*ping* hostname" command

Besides, the server interface also has a "Ping" feature to check if that client is still connected or not. This function is similar to "Discover" which is also performed through an input box and a "Ping" button to perform.



```
Received request:
PUBLIC file vy.mp4 Request
Host: DESKTOP-OJRHLU5
Port: 50432
Title: vy.mp4
real-Name: video.mp4


File available on DESKTOP-OJRHLU5 is:
1. vy.mp4
Khoa2807 via port 50231 left the server
DESKTOP-OJRHLU5 via port 50431 left the server
DESKTOP-OJRHLU5 is off now
```

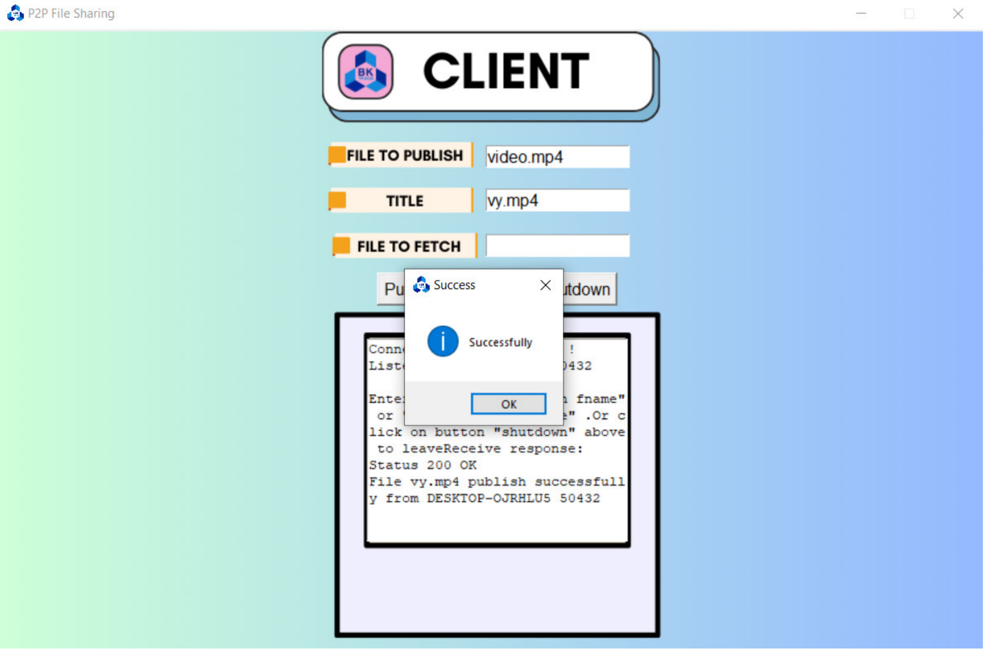
Figure 8: Notification when performing the ping feature

4.2 Client functions

The client interface has two main functions: publish files and fetch files

4.2.1 "*publish* lname fname" command

The publish function will help clients publish a certain file so that other clients can download it. To use this function, users enter the name of the file they want to publish in their file system, then enter the name they want to give that file in the "Title" input box and click the "Publish" button to proceed. After publishing successfully, the system will display a notification window.



FILE TO PUBLISH: video.mp4

TITLE: vy.mp4

FILE TO FETCH:

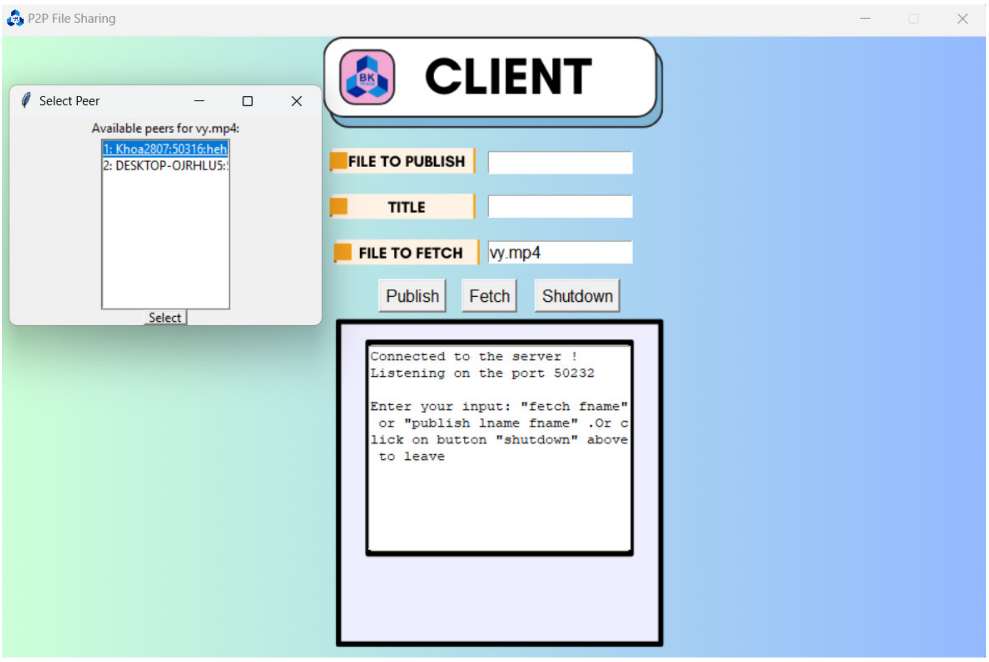
Successfully

```
Conn                         !
List                         0432

Ente                         fname"
 or                          e" .Or c
lick on button "shutdown" above
 to leaveReceive response:
Status 200 OK
File vy.mp4 publish successfull
y from DESKTOP-OJRHLU5 50432
```

Figure 9: Interface when using the publish function

4.2.2 "*fetch* fname" command

With the fetch function, users only need to enter the name of the file to fetch (corresponding to the title in the publish function) then click the fetch button. If there are multiple files with the same name published from different clients, the system will display a window allowing the user to choose which client to fetch the file from.



Available peers for vy.mp4:

1: Khoa2807:50316:heh
2: DESKTOP-OJRHLU5:

FILE TO FETCH: vy.mp4

```
Connected to the server !
Listening on the port 50232

Enter your input: "fetch fname"
 or "publish lname fname" .Or c
lick on button "shutdown" above
 to leave
```

Figure 10: Interface when using the fetch function

5 Summary

In this application, we used P2P file sharing architecture, which allows computers to download files and make them available to sharing files with other users on same the network. The implementation using Python language and "Tkinter" library for GUI (Graphic User Interface). In the nearby future, we see availability for improvement, such as making a website for eazier using purpose for user, make account for each user, refine the UI (User Interface) for better looking and so on. The identified potential enhancements exemplify the inherent adaptability and expansive growth potential of the platform, showcasing its ability to evolve and meet changing demands in the ever-dynamic technological landscape.

6 References

1. Carmen Carmack, *Peer-to-peer File Sharing*<https://computer.howstuffworks.com/bittorrent1.htm

2. *What is Class Diagram?*<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

3. *tkinter — Python interface to Tcl/Tk*<https://docs.python.org/3/library/tkinter.html