ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MẠNG MÁY TÍNH

(CO3093)

ASSIGNMENT 1
Develop A Network Application

GVHD: Nguyễn Lê Duy Lai
Lớp: TN01
SV thực hiện: Nguyễn Thái Tân - 2112256
Huỳnh Lê Đăng Khoa - 2211590

THÀNH PHỐ HỒ CHÍ MINH, 11/2024

Contents

1 Cơ sở lý thuyết

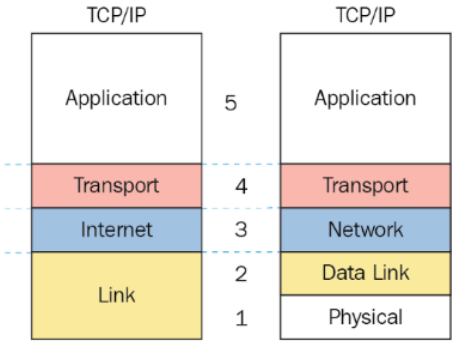
1.1 Protocol

1. Giao thức (protocol) là một bộ quy tắc và hướng dẫn để định dạng, truyền và giải mã dữ liệu giữa các thiết bị máy tính trên mạng. Giao thức xác định cách các thiết bị kết nối với nhau, cách chúng chia sẻ thông tin và cách chúng xử lí lỗi lúc truyền dữ liệu.
2. Giao thức được sử dụng trong tất cả các loại mạng, bao gồm mạng cục bộ (LAN), mạng diện rộng (WAN) và cả Internet. Chúng là nền tảng cho mọi hoạt động truyền gửi dữ liệu trên mạng, từ truy cập website đến gửi mail hay gọi điện video.
3. Một số giao thức phổ biến:
    - Giao thức vận chuyển (Transport Protocol)
        - IP (Internet Protocol)
        - TCP (Transmission Control Protocol)
        - UDP (User Datagram Protocol)
    - Giao thức ứng dụng (Application Protocol)
        - HTTP (Hypertext Transfer Protocol)
        - SMTP (Simple Mail Transfer Protocol)
        - FTP (File Transfer Protocol)

Trong Bài Tập Lớn 1 này, nhóm sẽ sử dụng mô hình TCP/IP là chủ yếu.

1.2 Mô hình TCP/IP

Mô hình TCP/IP (Transmission Control Protocol/Internet Protocol) là một bộ giao thức chuẩn được sử dụng để quản lý việc truyền thông tin trên Internet. Nó bao gồm hai giao thức chính: TCP (Transmission Control Protocol) và IP (Internet Protocol).
Một mô hình TCP/IP tiêu chuẩn bao gồm 4 lớp được chồng lên nhau, bắt đầu từ tầng thấp nhất là Tầng Datalink (Link) → Tầng mạng (Network) → Tầng vận tải (Transport) và cuối cùng là Tầng ứng dụng (Application) (Hình bên trái). Tuy nhiên, một số ý kiến lại cho rằng mô hình TCP/IP là 5 tầng, tức các tầng 4 đến 2 đều được giữ nguyên, nhưng tầng Datalink sẽ được tách riêng và là tầng nằm trên so với tầng vật lý. (Hình bên phải).

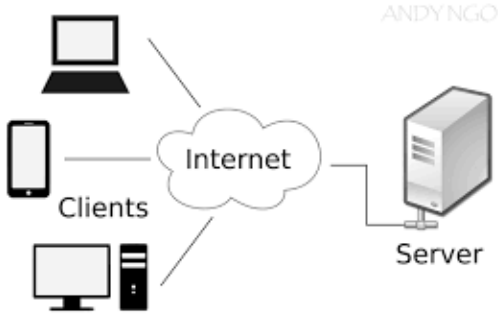


Hình 1: Mô hình TCP/IP

Trong bài tập lớn này, chúng ta cùng hiện thực lại tầng ứng dụng (Application).

1.2.1 Mô hình Client-Server

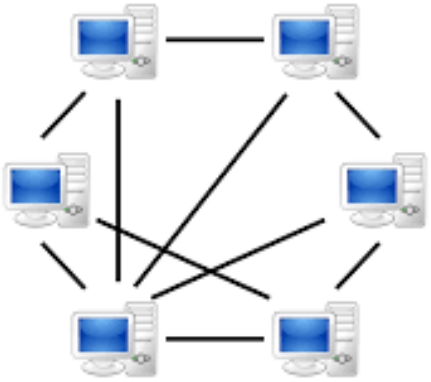
Client-Server là mô hình mạng máy tính gồm có 2 thành phần chính đó là máy khách (Client) và máy chủ (Server). Server chính là nơi giúp lưu trữ tài nguyên cũng như cài đặt các chương trình dịch vụ theo đúng như yêu cầu của Client. Ngược lại, Client bao gồm máy tính cũng như các loại thiết bị điện tử nói chung sẽ tiến hành gửi yêu cầu truy cập hay khai thác thông tin đến cho Server. Hay nói đơn giản thì Client là các thiết bị gửi yêu cầu đến Server để thực hiện một việc gì đó (Request) và Server là nơi tiếp nhận và xử lí các yêu cầu mà Client gửi đến rồi phản hồi lại với Client (Response).



Hình 2: Mô hình Client - Server

1.2.2 Mô hình Peer-To-Peer(P2P)

Peer-to-peer (P2P) là một kiến trúc mạng trong đó các thiết bị (hoặc "Peer") kết nối và trao đổi thông tin trực tiếp với nhau mà không thông qua một trung tâm điều khiển hay máy chủ trung gian như mô hình Client-Server. Trong mô hình này, các thiết bị được coi là ngang hàng với nhau và có khả năng chia sẻ tài nguyên và dữ liệu với nhau. Trong mạng P2P, mỗi Peer có khả năng đóng vai trò là cả người gửi và người nhận thông tin. Điều này cho phép các thiết bị trực tiếp kết nối với nhau để trao đổi dữ liệu, chia sẻ tài nguyên như tệp tin, ứng dụng, hoặc dịch vụ mà không cần một máy chủ trung gian quản lý.



Hình 3: Mô hình Peer to Peer

1.2.3 So sánh mô hình Client-Server và mô hình Peer-To-Peer

Điểm so sánh	Client-Server	Peer-To-Peer
Vai trò, phân quyền	Phân chia vai trò rõ ràng giữa Client và Server	Trong cùng một mạng thì tất cả các Peer đều ngang hàng
Quản trị mạng	Cần có người quản trị mạng	Không cần người quản trị mạng
Phần cứng, phần mềm cần thiết	Cần máy chủ, phần cứng và hệ điều hành	Cần khá ít phần cứng, có thể không cần hệ điều hành và máy chủ
Chi phí cài đặt	Cao	Thấp
Quản lí tài nguyên	Tài nguyên được tập trung trên máy chủ	Tài nguyên được chia sẻ trực tiếp giữa các thiết bị
Hiệu suất và mở rộng	Có thể quản lí số lượng lớn Client và tối ưu hóa hiệu suất của Server	Hiệu suất có thể giảm đi khi số lượng thiết bị tăng lên, do cần tiêu tốn thêm tài nguyên để xử lý các yêu cầu từ các peer khác.
Bảo mật	Một máy chủ trung tâm có thể tập trung vào việc triển khai các biện pháp bảo mật, giúp kiểm soát và bảo vệ tài nguyên.	Cần phải có các biện pháp bảo mật phân tán cho mỗi thiết bị, do không có một điểm trung tâm quản lý.

2 Thiết kế ứng dụng

2.1 Các hàm chức năng

Có 2 loại thông điệp mà các socket trao đổi với nhau trong ứng dụng:

- Yêu cầu(Request): là các yêu cầu từ máy người dùng gửi đến với mong muốn nhận được kết quả của yêu cầu. phía nhận có thể là máy của người dùng khác hoặc tracker.
- Phản hồi(Reponse): Là phản hồi của máy nhận với yêu cầu của máy gửi; phản hồi có thể được gửi bởi Tracker hoặc Peer khác trong mạng.

Các hàm chức năng:

- Peer:
  - **Publish:** Tiến hành gửi thông tin chi tiết và dữ liệu của tất cả file lên Tracker.
  - **Fetch file:** Fetch một bản copy của file mà người dùng yêu cầu. Peer gửi yêu cầu lên Tracker và Tracker gửi danh sách các hostname đã publish file kèm theo thông tin, Peer chọn kết nối peer to peer với một trong các hostname để fetch bản copy của file cần tải về.
  - **History file**: Peer gửi yêu cầu lên Tracker và Tracker gửi danh sách lịch sử của file.
  - **Reset file**: Peer gửi yêu cầu lên Tracker và Tracker gửi dữ liệu của file với phiên bản được yêu cầu.
  - **Block/Unblock**: Peer gửi yêu cầu lên Tracker để thông báo rằng Peer này chặn/gỡ chặn một Peer khác.
  - **Quit:** Peer ngắt kết nối với Tracker và ngừng hoạt động.
- Tracker:
  - **List:** Tracker tiến hành kiểm tra xem hiện có Peer nào đang hoạt động.
  - **Update:** Tracker tiến hành kiểm tra và cập nhật trạng thái của các file hiện có trong kho chứa của tất cả các Peer.
  - **Quit:** Tracker ngắt kết nối với tất cả Peer và ngừng hoạt động.

2.2 Các giao thức

- Peer-to-Tracker protocol:
  - **Connect:** Peer kết nối với Tracker.
    - Request: CONNECT files_information
    - Response: CONNECT_RESULT peer_address
  - **List:** Yêu cầu Tracker cung cấp danh sách các Peer đang hoạt động
    - Request: LIST
    - Response: LIST_RESULT peer_list
  - **Publish:** Publish thông tin và dữ liệu của tất cả file trên local của Peer, Tracker sẽ lưu lại các thông tin và dữ liệu trên.
    - Request: PUBLISH files_information

    - * Response: PUBLISH_RESULT status
  - **Fetch file_name:** Yêu cầu Tracker cung cấp thông tin địa chỉ của các peer có dữ liệu của file file_name.
    - * Request: FETCH file_name
    - * Response: FETCH_RESULT list_peers_has_file
  - **History file_name:** Yêu cầu Tracker cung cấp danh sách lịch sử của một file cụ thể
    - * Request: HISTORY file_name
    - * Response: HISTORY_RESULT file_history
  - **Reset file_name version:** Yêu cầu Tracker cung cấp dữ liệu của file theo phiên bản được yêu cầu
    - * Request: RESET file_name version
    - * Response: RESET_RESULT file_name, file_data_encoded
  - **Block peer_ip, peer_port:** Yêu cầu Tracker lưu trữ các peer bị chặn
    - * Request: BLOCK peer_ip, peer_port
    - * Response: BLOCK_RESULT status
  - **Unblock peer_ip, peer_port:** Yêu cầu Tracker cập nhật các peer bị chặn
    - * Request: UNBLOCK peer_ip, peer_port
    - * Response: UNBLOCK_RESULT status

- Peer-to-Peer protocol:
  - **Fetch_Peer file_name:** Peer yêu cầu fetch file từ Peer khác
    - * Request: FETCH_PEER file_name
    - * Response: file_data

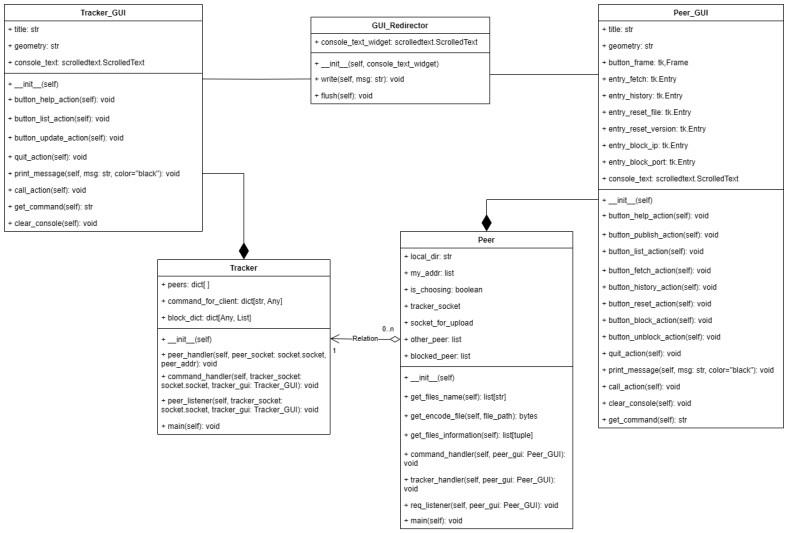
- Tracker-to-Peer protocol:
  - **Update:** Yêu cầu tất cả các Peer gửi thông tin và dữ liệu của tất cả các file trên local của họ, Tracker sẽ lưu lại các thông tin và dữ liệu trên.
    - * Request: UPDATE
    - * Response: UPDATE_RESULT files_information

- TCP/IP protocol stack:
  - **ICP:** TCP sẽ được sử dụng để cung cấp kết nối đáng tin cậy cho tầng transport của ứng dụng. Điều này là cần thiết để đảm bảo rằng các file sẽ được truyền một cách chính xác và đầy đủ.
  - **IP:** IP được sử dụng để định tuyến đường đi cho các gói dữ liệu(packet) giữa các Peer và Tracker.

3 Hiện thực ứng dụng

3.1 Class diagram

Đây là đường dẫn đến class diagram chi tiết: link



Hình 4: Class diagram của ứng dụng

3.2 Hiện thực GUI bằng Tkinter

Tkinter là thư viện GUI tiêu chuẩn cho Python. Tkinter trong Python cung cấp một cách nhanh chóng và dễ dàng để tạo các ứng dụng GUI. Tkinter cung cấp giao diện hướng đối tượng cho bộ công cụ Tk GUI.

Để sử dụng được thư viện Tkinter, ta sử dụng lệnh

```
import tkinter as tk
```

3.2.1 Class Tracker_GUI - Peer_GUI

Để hiện thực GUI, ta viết một class **Tracker_GUI** cho Tracker (với Peer, ta hoàn toàn làm tương tự với class **Peer_GUI**), kế thừa từ *tk.Tk*.

- Ở constructor, ngoài việc gọi lại constructor của *tk.Tk*, ta cũng viết thêm một số giao diện cho **Tracker_GUI**.
  - Tiêu đề: sử dụng *tk.Label*; để điều chỉnh vị trí cho tiêu đề, sử dụng *tk.Label.pack*
  - Tạo khung để chứa các nút và khung để chứa các dòng log: sử dụng *tk.Frame*; để điều chỉnh vị trí các nút, sử dụng *tk.Frame.pack*
  - Các nút: sử dụng *tk.Button*; để điều chỉnh vị trí các nút, sử dụng *tk.Button.grid*
  - Các khung nhỏ để nhập dữ liệu: sử dụng *tk.Entry*; để điều chỉnh vị trí các nút, sử dụng *tk.Entry.grid*

  - Khung log chính: sử dụng *tkinter.scrolledtext.ScrolledText*; để điều chỉnh vị trí của khung log này, sử dụng *tkinter.scrolledtext.ScrolledText.pack*

- Các hàm **button_help_action()**, **button_list_action()**, **button_update_action()**, **button_quit_action()**: xử lý các tác vụ nếu ấn vào các nút help, list, update, quit
- Các hàm hỗ trợ khác:
  - **def print_message(self, msg: str, color="black")**: in ra vào cuối khung log chuỗi msg, với màu color.
  - **def call_action(self)**: thực hiện tác vụ chính
  - **def get_command(self):** xác định tác vụ cần thực hiện tiếp theo
  - **def clear_console(self)**: làm sạch khung log

3.2.2 Class GUI_Redirector

Class này được dùng để chuyển tất cả những *stdout* sang Tracker_GUI và Peer_GUI.

- Constructor: chọn nơi để *stdout* được in ra, ở trường hợp này là khung log đã đề cập bên trên.
- **def write(self, msg)**: khi in *stdout*, sẽ thực hiện in vào phía dưới cùng của khung log.

3.3 Hiện thực các hàm chức năng

3.3.1 Tracker.py

Class Tracker có các thuộc tính sau đây:

- **peers**: một dictionary chứa các peer hiện đang kết nối đến tracker.
- **command_for_peer**: một dictionary thể hiện tracker đang gửi yêu cầu đến peer nào trong danh sách các peer, việc này dùng để xác định thông tin được gửi tiếp theo là yêu cầu của 1 peer khác hay là câu trả lời từ peer mà tracker vừa gửi yêu cầu.

Class Tracker có các hàm sau đây:

- **def __init__(self) -> None**: constructor, khởi tạo các giá trị ban đầu cho các thuộc tính vừa kể trên.
- **def peer_listener(self, tracker_socket, my_terminal)**: hàm này dùng để thiết lập kết nối ban đầu giữa các peer và tracker; bên trong hàm cũng sử dụng cơ chế thread để kết nối được nhiều peer.
- **def peer_handler(self, peer_socket, peer_addr)**: hàm này dùng để xử lý các lệnh đến từ các peer (có các lệnh fetch, list, publish, history, reset, block, unblock như đã nói ở trên).
- **def command_handler(self, tracker_socket, my_terminal)**: một hàm chung để xử lý các lệnh của chính tracker (update, quit như đã nói ở trên).
- **def main(self)**: thiết lập GUI; kết nối GUI và các giao thức mạng, có sử dụng thread để xử lý đa luồng.

3.3.2 Peer.py

Class Peer chứa các thuộc tính sau đây:

- **local_dir**: đường dẫn đến thư mục chứa các file được chia sẻ.
- **my_addr**: bao gồm địa chỉ ip và port của máy nhận, dùng để:
  - Lưu địa chỉ của máy đến tracker.
  - Tạo connection mới với ip = my_addr[0] và port = my_addr[1] + 1 để chờ kết nối từ các peer khác.
- **is_choosing**: cờ để xác định câu input tiếp theo từ terminal là số thứ tự peer được chọn để lấy file về .
- **tracker_socket**: socket kết nối giữa tracker và peer này.
- **socket_for_upload**: socket dùng để tiếp nhận kết nối từ các peer khác.
- **other_peer**: danh sách các peer khác có file yêu cầu trả về từ tracker.
- **blocked_peer**: danh sách các peer đã bị chặn

Class Peer có các hàm sau đây:

- **def __init__(self) -> None**: constructor, khởi tạo các giá trị ban đầu cho các thuộc tính vừa kể trên.
- **def get_files_name(self)**: sử dụng os.listdir() để trả về một danh sách chứa tên của các mục trong thư mục được cung cấp bởi đường dẫn.
- **def get_encode_file(self)**: sử dụng base64 để trả về dữ liệu mã hóa của một file.
- **def get_files_information(self)**: trả về một danh sách chứa thông tin của các file bao gồm tên, kích thước và dữ liệu đã được mã hóa.
- **def command_handler(self, my_terminal)**: một hàm chung để xử lý các lệnh của peer (fetch, list, publish, history, reset, block, unblock, quit như đã nói ở trên).
- **def tracker_handler(self, my_terminal)**: hàm này dùng để xử lý các lệnh (update) và kết quả (list_result, fetch_result, history_result, reset_result, block_result, unblock_result) đến từ tracker.
- **def req_listener(self, my_terminal)**: xử lý các yêu cầu đến từ các peer khác.
- **def main(self)**: thiết lập GUI; kết nối GUI và các giao thức mạng, có sử dụng thread để xử lý đa luồng.

4 Cách sử dụng ứng dụng

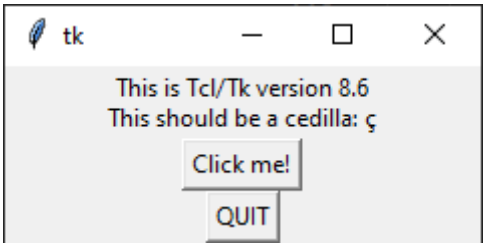
Đầu tiên, ta cài đặt thư viện Tkinter bằng cách nhập vào command line dòng lệnh

```
pip install tk
```

Để kiểm tra việc cài đặt Tkinter có thành công không, ta nhập vào command line dòng lệnh

```
python -m tkinter
```

Nếu máy tính tự động hiện ra một window như bên dưới, tức là Tkinter đã được cài đặt thành công.

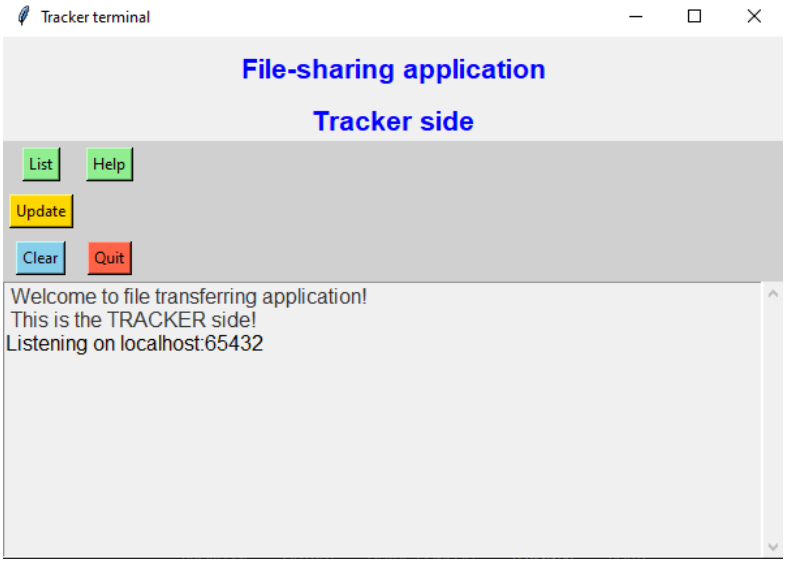


Hình 5: Kiểm tra Tkinter

Ta khởi động Tracker bằng cách chạy file *Tracker.py*. Ta nhập vào command line dòng lệnh sau:

```
python Tracker.py
```

Nếu thành công, máy tính sẽ tự động hiện ra một window cho Tracker như sau:

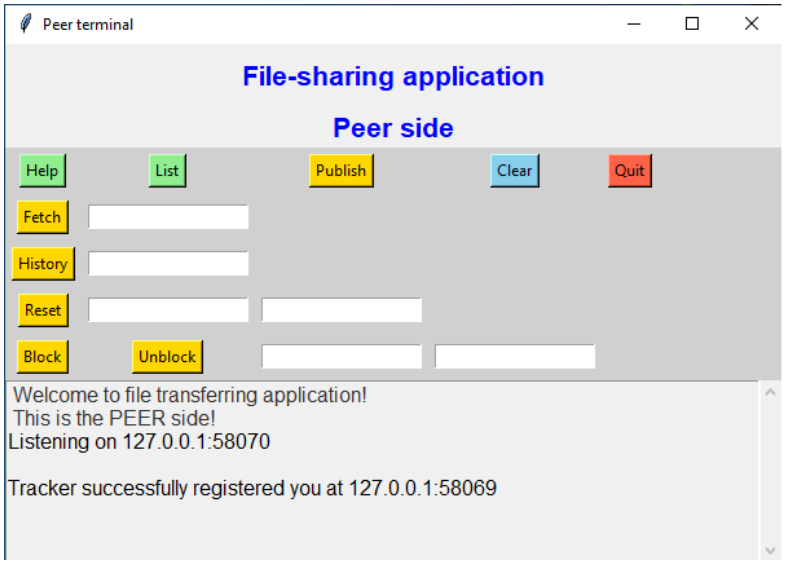


Hình 6: Giao diện của Tracker

Nhập vào command line câu lệnh sau để chạy file *Peer.py*:

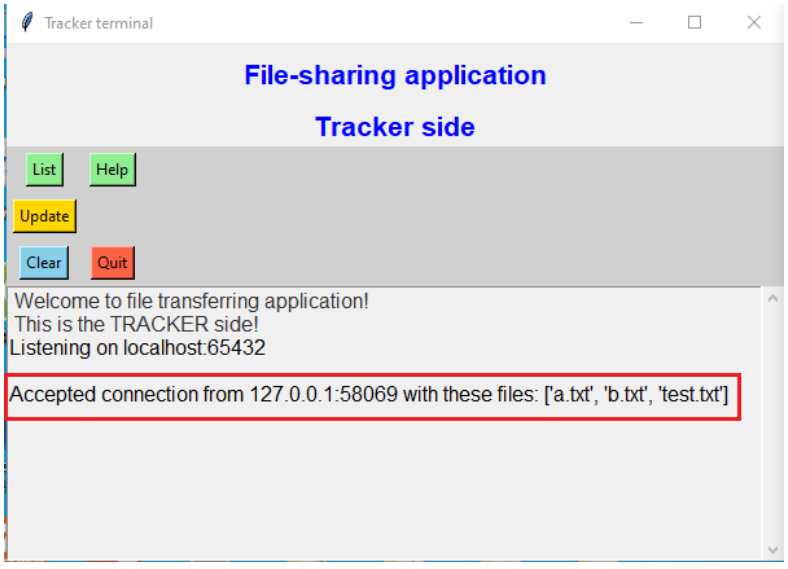
```
python Peer.py
```

Nếu thành công, máy tính sẽ tự động hiện ra một window cho Peer như sau:



Hình 7: Giao diện của Peer

Sau khi thành công mở một peer, thì tracker cũng sẽ tự động có một dòng thông báo là có một peer mới.



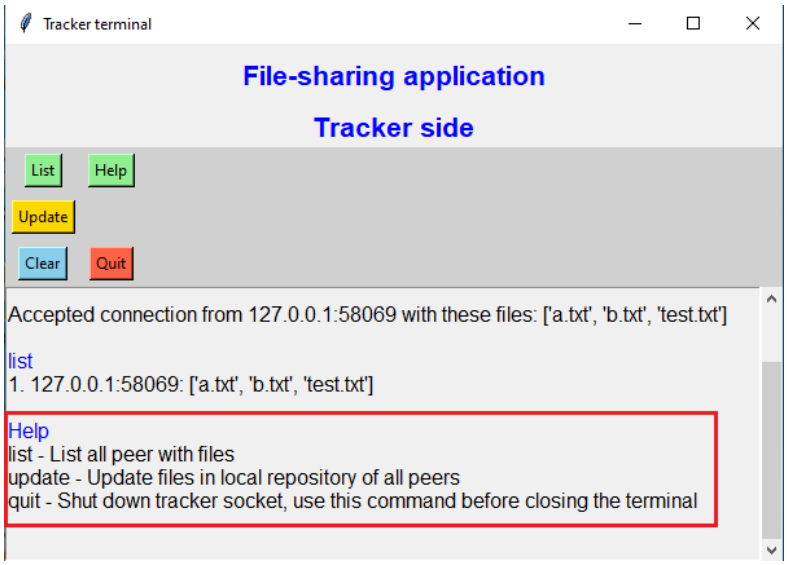
Hình 8: Thông báo một peer mới

4.1 Tracker manual

Trong tracker ta có các câu lệnh sau:

- **help**

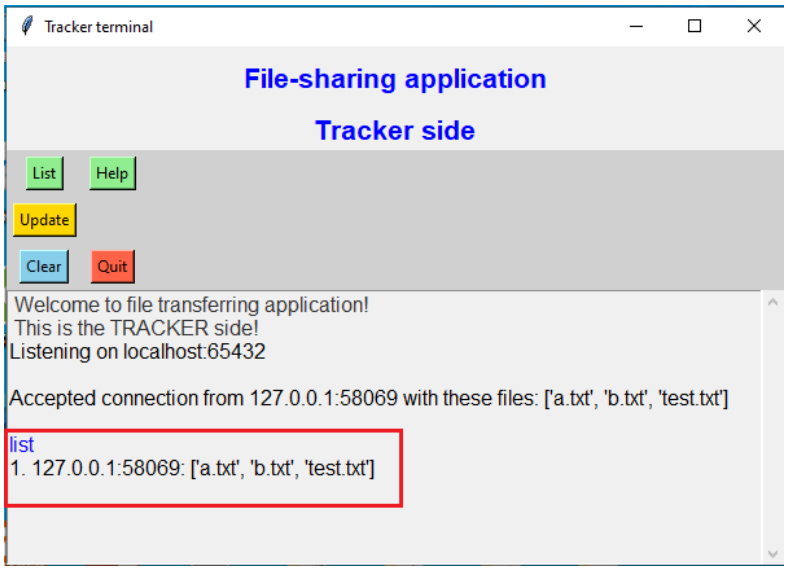
Khi thực hiện lệnh *help*, ứng dụng sẽ in ra một loạt các câu hướng dẫn sử dụng cho người dùng, như sau:



Hình 9: Tracker help

- **list**

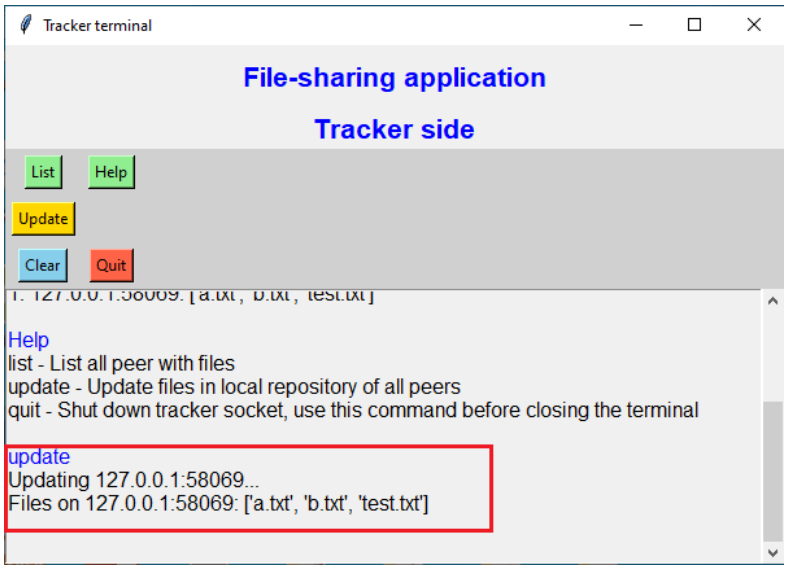
Khi thực hiện lệnh *list*, ứng dụng sẽ liệt kê ra các peer đang được đăng ký với tracker. Trong trường hợp ví dụ này, có **1** peer đăng ký với tracker.



Hình 10: Tracker list

- **update**

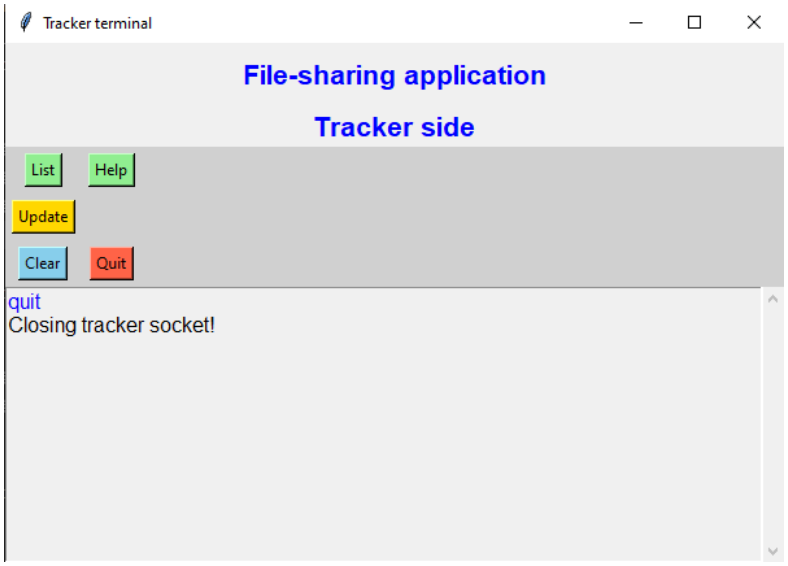
Khi thực hiện lệnh *update*, ứng dụng sẽ lần lượt cập nhật thông tin của các file có trong từng peer.



Hình 11: Tracker update

• **quit**

Khi thực hiện lệnh *quit*, ứng dụng sẽ đóng tất cả các socket liên quan đến tracker này, khi đó ta có thể đóng ứng dụng một cách an toàn.



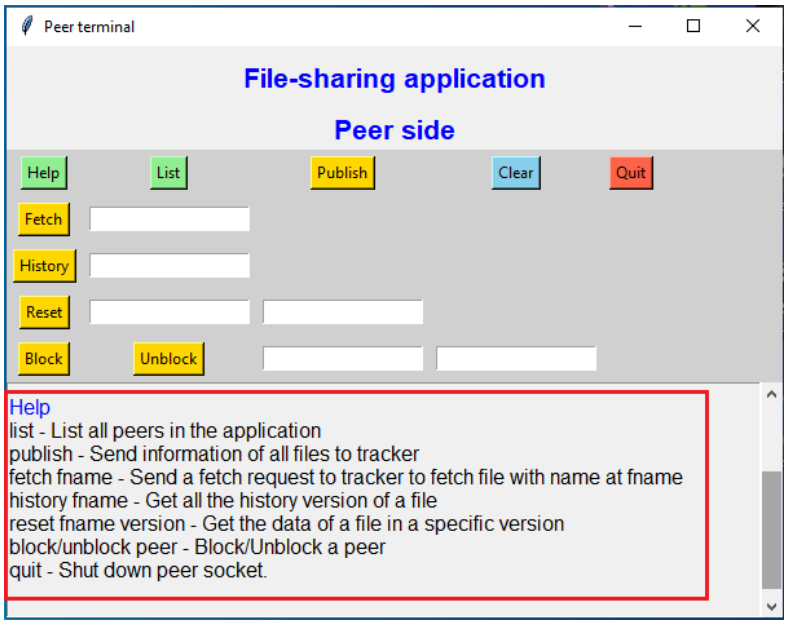
Hình 12: Tracker quit

4.2 Peer manual

Trong peer ta có các câu lệnh sau:

- **help**

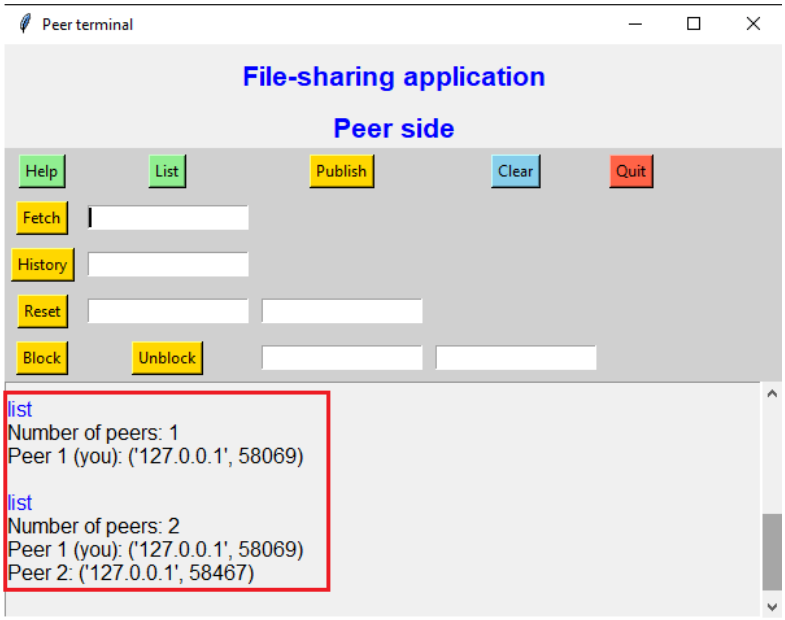
Khi thực hiện lệnh *help*, ứng dụng sẽ in ra một loạt các câu hướng dẫn sử dụng cho người dùng, như sau:



Hình 13: Peer help

- **list**

Khi thực hiện lệnh *list*, ứng dụng sẽ liệt kê ra các peer đang được đăng ký với tracker.

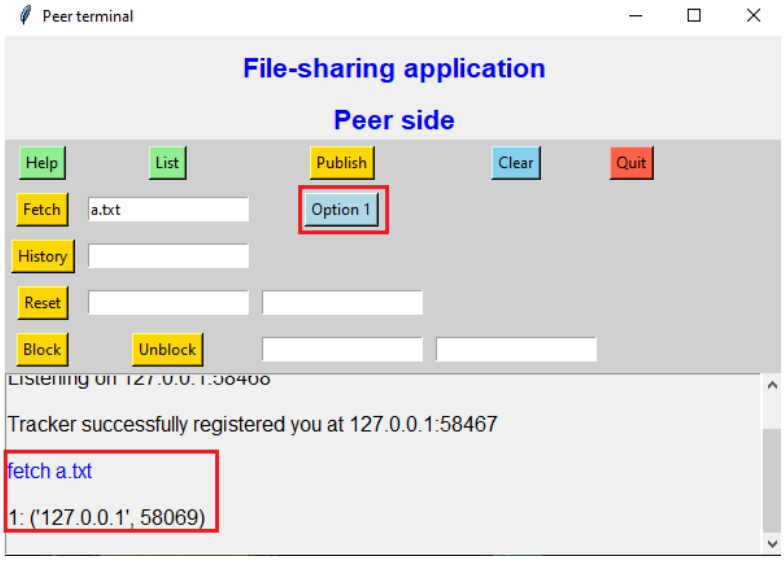


Hình 14: Peer list

- **fetch**

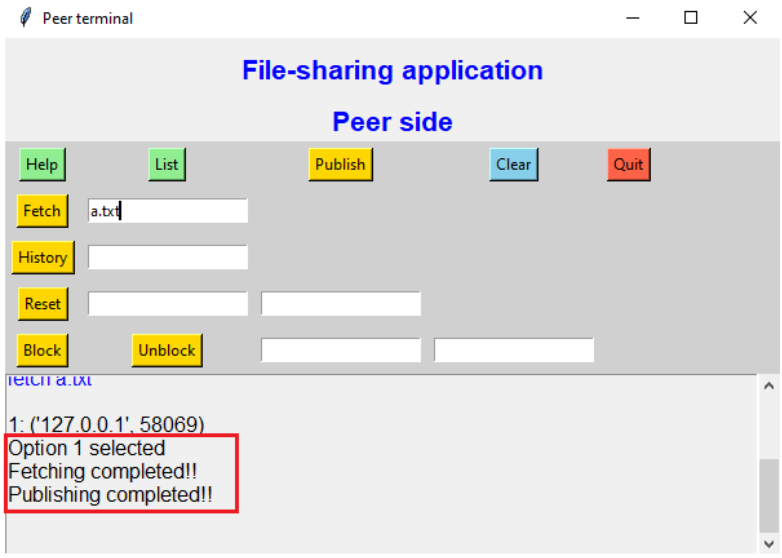
Khi thực hiện lệnh *fetch + fname*, ứng dụng sẽ tìm kiếm file có tên trùng với *fname* từ các peer khác cùng đăng ký với tracker.

Trước ta, ta cần phải có ít nhất 2 peer. Sau khi thực hiện lệnh fetch một file *a.txt*, ứng dụng sẽ trả về peer có chứa file này.



Hình 15: Peer fetch

Sau đó, ta lựa chọn peer muốn fetch file này. Trong trường hợp này là peer 1.



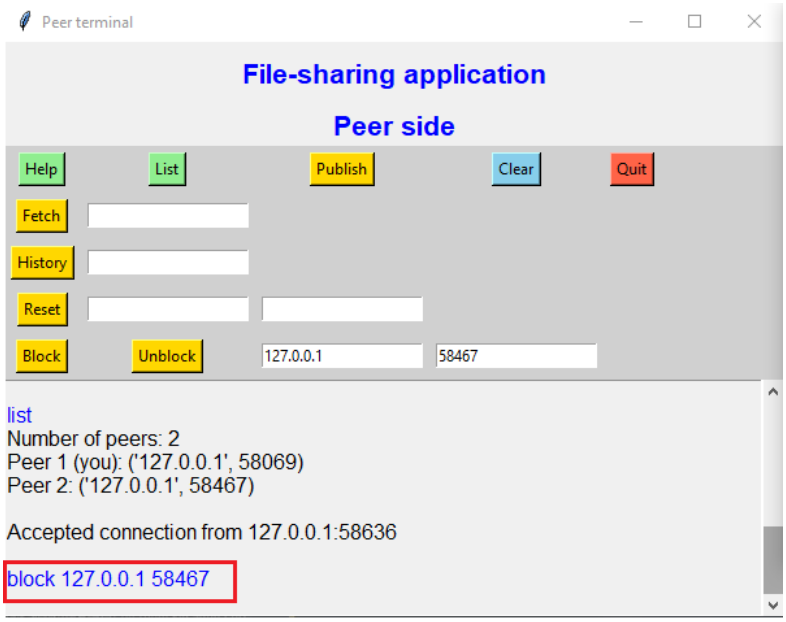
Hình 16: Peer fetch thành công

Như vậy, ta đã fetch thành công file *a.txt* cho peer này.

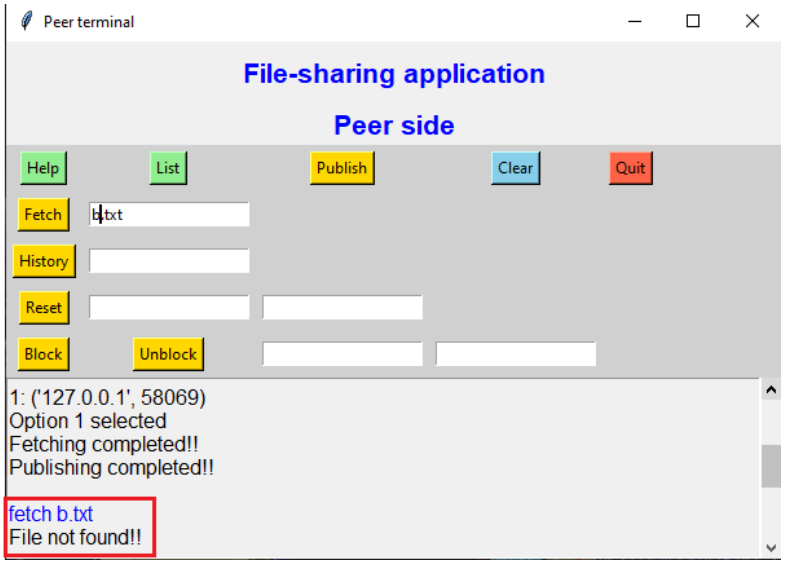
• **block và unblock**

Khi thực hiện lệnh *block + peer*, ứng dụng sẽ chặn peer.

Giả sử một peer A bị chặn bởi peer B, khi peer A thực hiện fetch file nào đó, mặc dù peer B vẫn có file này, nhưng peer A sẽ không nhận được thông tin này.

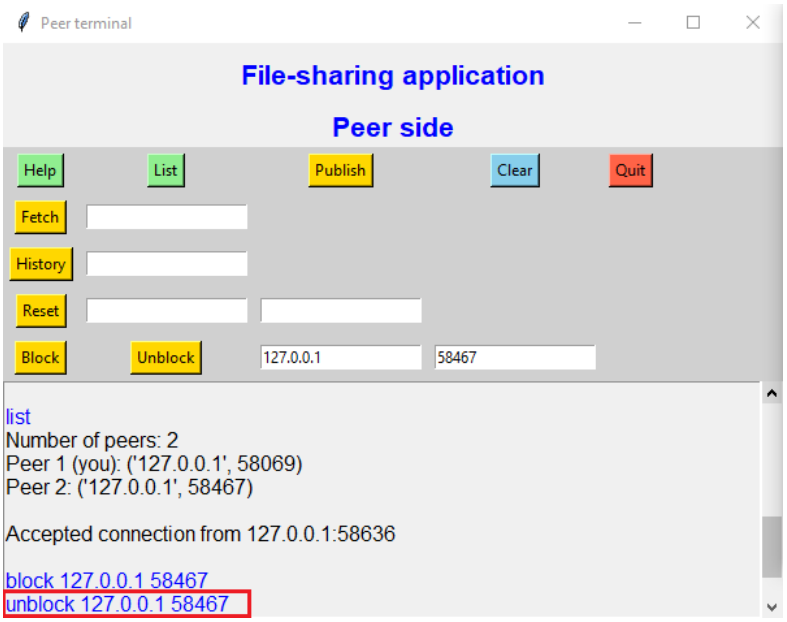


Hình 17: Peer block

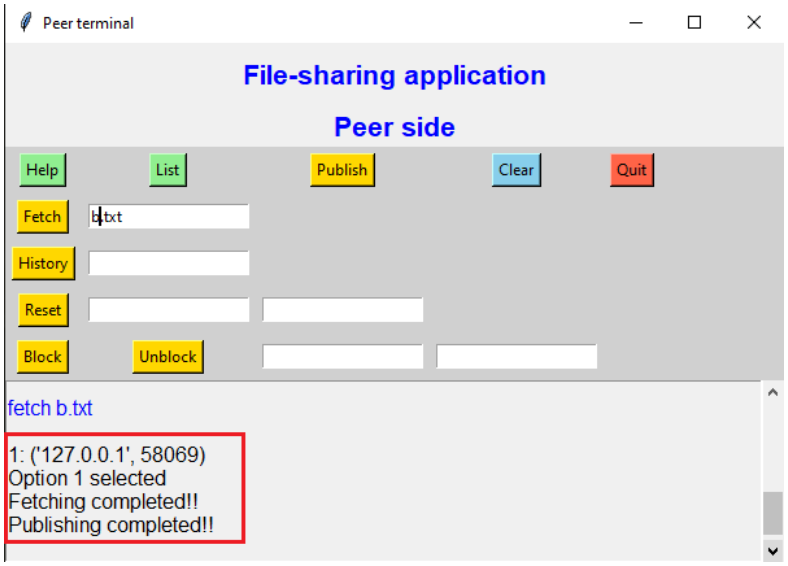


Hình 18: Peer fetch nhưng không nhận được kết quả

Khi thực hiện lệnh *unblock + peer*, ứng dụng sẽ gỡ chặn peer.



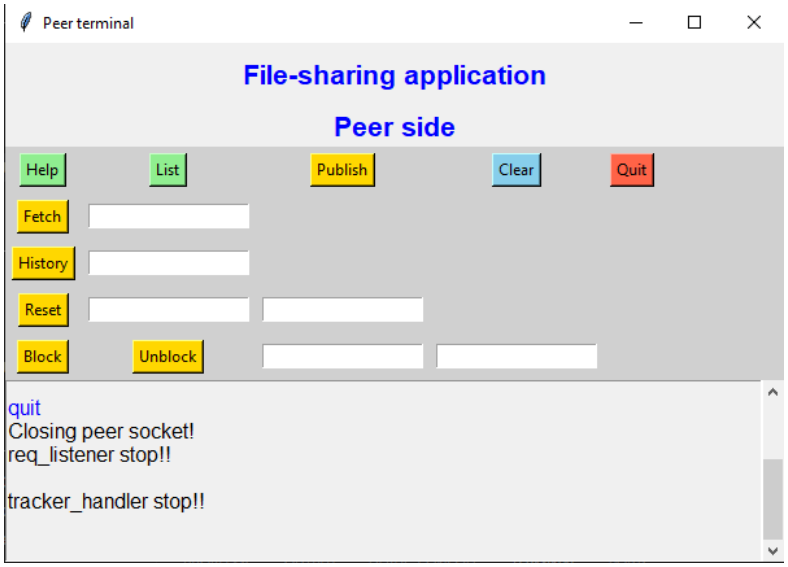
Hình 19: Peer unblock



Hình 20: Peer fetch và nhận được kết quả

- **quit**

Khi thực hiện lệnh *quit*, ứng dụng sẽ đóng tất cả các socket liên quan đến peer này, khi đó ta có thể đóng ứng dụng một cách an toàn.



Hình 21: Peer quit

Lưu ý: Với các lệnh **fetch**, **history**, **reset**, **block**, **unblock**, ứng dụng sẽ kiểm tra ở mức GUI trước khi thực hiện lệnh. Nghĩa là nếu người dùng không điền vào ô thông tin ở từng lệnh, ứng dụng sẽ yêu cầu người dùng nhập lại.

5 Kết quả

5.1 Kết quả hiện tại

Ưu điểm

- Ứng dụng hoạt động bình thường trên hệ điều hành Window 10.
- Ứng dụng hoạt động bình thường khi tracker và peer được host trên cùng một localhost.
- Ứng dụng hoạt động bình thường khi tracker và peer được host trên các máy tính riêng biệt và kết nối vào cùng một wifi.
- Việc fetch được các file chứng tỏ ứng dụng xử lý các thread bình thường.

Nhược điểm

- Giao diện GUI vẫn chưa đẹp.
- GUI vẫn còn bug: khi ấn chuột vào vùng log, ứng dụng không thể nhận dạng các lệnh chính xác.

5.2 Định hướng phát triển

- Cải thiện giao diện GUI.
- Sửa lỗi liên quan đến vùng log của GUI.

6 Phân công nhiệm vụ

Họ và tên	Nhiệm vụ	Đánh giá
Huỳnh Lê Đăng Khoa	Hiện thực GUI	100%
Nguyễn Thái Tân	Hiện thực socket	100%
Các nhiệm vụ chung	Kiểm tra chéo, viết báo cáo	100%

Bảng 1: Bảng nhiệm vụ của từng thành viên

References

[1] Jim Kurose, Keith Ross. *Computer Networking: A Top-Down Approach.* 8th Global Edition, Pearson, 2021.

[2] Python. *tkinter - Python interface to Tcl/Tk*.