

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)

Bài tập lớn - HK241

Network Application P2P File Sharing

GVHD: Lê Bảo Khánh

Sinh viên thực hiện: Trần Thanh Phúc 2212656

Phạm Minh Khoa 2211643

Lê Võ Đăng Khoa 2211606

Trần Thế Nhân 2212383

TP.HỒ CHÍ MINH, THÁNG 11 NĂM 2024

Mục lục

1	Mở đầu	5
1.1	Đặt vấn đề	5
1.2	Mục tiêu	5
1.3	Công nghệ sử dụng	6
2	Nội dung chính	7
2.1	Cơ sở lý thuyết	7
2.1.1	Khái niệm về giao thức	7
2.1.2	Giao thức TCP/IP	7
2.1.3	Giao thức socket	8
2.1.4	Giao thức HTTP	8
2.2	Yêu cầu chức năng và phi chức năng	9
2.2.1	Yêu cầu chức năng	9
2.2.2	Yêu cầu phi chức năng	11
3	Mô hình hệ thống	13
3.1	Usecase diagram	13
3.1.1	Publish File	13
3.1.2	Request File List	14
3.1.3	Request File	14
3.1.4	Fetch File	15
3.1.5	Merge File Pieces	16
3.1.6	Start Server	16
3.1.7	Register File	17
3.1.8	Discovering File	18
3.1.9	Ping Host	18
3.1.10	Provide File List	19
3.2	Sequence diagram	20
3.2.1	Quy trình kết nối và giao tiếp giữa Client và Tracker	20
3.2.2	Quy trình tải file giữa các peer	21
4	Các chức năng của ứng dụng	22
4.1	Phía Tracker	22
4.1.1	Chức năng quản lý	22
4.1.2	Khởi tạo máy chủ	22

4.1.3	Trả lời yêu cầu của các client	22
4.1.4	Lưu trữ danh sách các client đang kết nối với server và danh sách file của các client đây	23
4.1.5	Trình thông dịch dòng lệnh: 'discover', 'ping'	23
4.1.6	Chuyển đổi	23
4.2	Phía Peer (Node)	24
4.2.1	Kết nối tới máy chủ	24
4.2.2	Thông báo cho server biết những file đang có	24
4.2.3	Gửi thông tin để truy hồi file hoặc thông báo file hiện có	24
4.2.4	Trình thông dịch dòng lệnh: 'publish', 'fetch'	24
4.2.5	Xử lý khi truyền tải tệp	24
5	Các giao thức cho ứng dụng	25
5.1	Giữa các tính năng khi giao tiếp giữa client và server	25
5.1.1	Ping	25
5.1.2	Discover	25
5.1.3	Publish	25
5.1.4	Fetch	25
5.2	Giữa các tính năng khi giao tiếp giữa client và client	26
5.2.1	Khởi tạo và lắng nghe kết nối	26
5.2.2	Kiểm tra trạng thái file tại peer	26
5.2.3	Tải xuống các mảnh file từ peer	26
5.3	Cơ chế Phân Chia Chunk Dựa Trên Điểm Số	26
6	Hiện thực hệ thống	28
6.1	Tracker server	28
6.1.1	Định nghĩa	28
6.1.2	Kiến trúc	28
6.1.3	Cấu trúc	29
6.1.4	Hiện thực	32
6.2	Client side	35
6.2.1	Download	35
6.2.2	Lấy thông tin về peers từ server	36
6.2.3	Kiểm tra Trạng thái File từ Các Peers	37
6.2.4	Quản lý Trạng thái Tải Xuống	38
6.2.5	Lựa chọn Peer Tốt Nhất	39

6.2.6	Tải Xuống File	40
6.2.7	Upload_full_file	42
6.2.8	Handle_client	43
7	Hướng dẫn sử dụng	47
7.1	Khởi động server tracker	47
7.2	Khởi động các client	47
7.3	Tải file lần đầu cho một peer để đăng kí thông tin file cho tracker	48
7.4	Download file	48
7.5	Tải file từ nhiều đồng thời từ nhiều peer	49



1 Mở đầu

1.1 Đặt vấn đề

Trong thời đại phát triển mạnh mẽ của công nghệ thông tin hiện nay, việc trao đổi dữ liệu giữa các máy tính đã trở nên vô cùng cần thiết. Có nhiều phương pháp đã được nghiên cứu và áp dụng để đáp ứng nhu cầu này, trong đó, nổi bật là phương pháp truyền tệp ngang hàng (peer-to-peer). Phương pháp này cho phép dữ liệu được truyền trực tiếp giữa hai máy tính mà không cần phải qua máy chủ trung gian.

Truyền tệp ngang hàng (P2P - Peer-to-Peer) là một mô hình mạng máy tính, trong đó các thiết bị tham gia hoạt động đồng thời như cả máy khách lẫn máy chủ, cho phép chia sẻ dữ liệu trực tiếp giữa các thiết bị mà không cần đến một máy chủ trung tâm. Khái niệm P2P xuất hiện vào cuối những năm 1990 và nhanh chóng trở nên phổ biến nhờ vào các ứng dụng chia sẻ tệp tin như Napster, Kazaa, và sau này là BitTorrent.

Với P2P, dữ liệu được chia nhỏ thành các phần và phân phối giữa các thiết bị trong mạng. Điều này giúp tăng tốc độ truyền tải và giảm tải cho một máy chủ duy nhất, đồng thời tăng khả năng chịu lỗi của mạng vì dữ liệu có thể được tải xuống từ nhiều nguồn. P2P không chỉ được ứng dụng trong chia sẻ tệp tin mà còn được sử dụng trong các lĩnh vực như blockchain, truyền phát video, và cả các hệ thống truyền thông phân tán.

Tuy nhiên, vì tính chất phân tán và không có trung gian, P2P cũng gặp một số thách thức về bảo mật và quản lý nội dung. Mặc dù vậy, với sự phát triển liên tục, mô hình P2P đã và đang khẳng định vai trò quan trọng trong việc truyền tải dữ liệu và xây dựng các hệ thống phân tán hiện đại.

1.2 Mục tiêu

Ứng dụng cần đáp ứng các yêu cầu sau:

- Xây dựng một ứng dụng truyền tệp ngang hàng đơn giản, dễ sử dụng.
- Hỗ trợ truyền tệp giữa các máy tính một cách hiệu quả.
- Cho phép tải xuống đồng thời nhiều tệp torrent từ nhiều node khác nhau.
- Cho phép tải lên đồng thời nhiều tệp torrent đến nhiều node.



- Cung cấp số liệu thống kê về lượt tải xuống/tải lên, hỗ trợ kiểm tra thông tin chi tiết về các node và tệp torrent đang hoạt động.

Để đạt được các mục tiêu trên, nhóm đã thực hiện các công việc sau:

1. Về phía client

- (a) Hiện thực đăng kí, đăng nhập và đăng xuất cho phép lưu thông tin sử dụng của người dùng.
- (b) Cho phép người dùng download file từ nhiều peer khác nhau và publish thông tin file từ máy local lên server.
- (c) Xây dựng giao diện dòng lệnh (CLI) giúp dễ dàng theo dõi hoạt động giữa các client và hỗ trợ truyền tải tệp.

2. Về phía tracker

- (a) Phát triển hệ thống cơ sở dữ liệu để lưu trữ thông tin về các client, peer và danh sách tệp của từng client.
- (b) Có khả năng xử lý và phản hồi các yêu cầu fetch, publish của người dùng.

3. Đối với yêu cầu chung của hệ thống

- (a) Hiện thực và kiểm thử các chức năng truyền tải dữ liệu giữa các client nhằm đảm bảo độ ổn định và hiệu quả của ứng dụng.
- (b) Hiện thực thống có chức năng tương tự giao thức Torrent.

1.3 Công nghệ sử dụng

- Python: Client, Client_CLI, Tracker_CLI.
- Typescript ứng dụng NestJS: Tracker_server.
- MySQL cho database và Sequelize cho công cụ ORM.

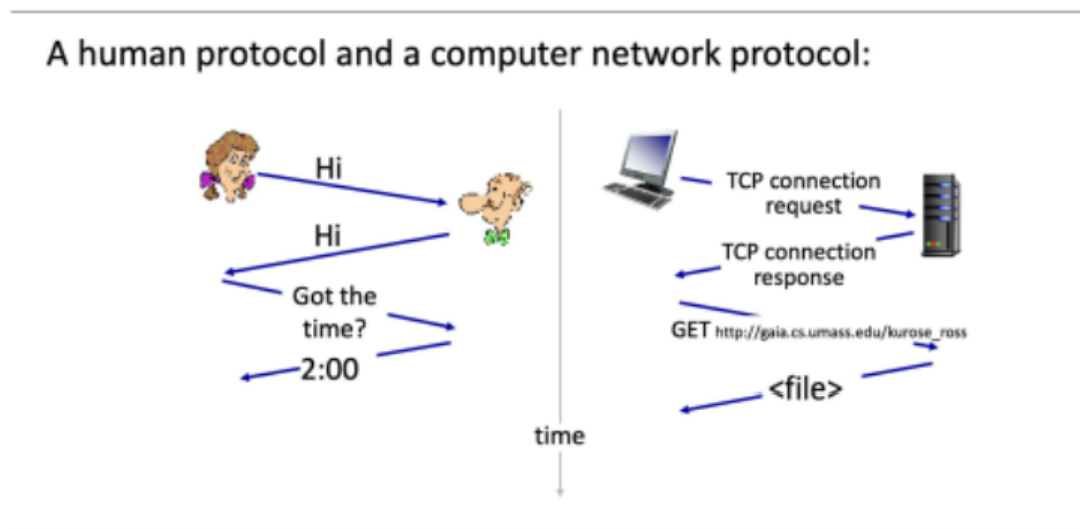


2 Nội dung chính

2.1 Cơ sở lý thuyết

2.1.1 Khái niệm về giao thức

Giao thức được định nghĩa là định dạng, thứ tự của các thông điệp được gửi và nhận giữa các thực thể mạng, cũng như các hành động khi thực hiện truyền và nhận thông điệp.



2.1.2 Giao thức TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) là một bộ giao thức mạng thiết yếu, đóng vai trò là nền tảng chính cho việc kết nối và truyền tải dữ liệu trong môi trường Internet. Ra đời vào những năm 1970, TCP/IP đã trở thành tiêu chuẩn toàn cầu, cho phép các thiết bị và hệ thống khác nhau giao tiếp với nhau bất kể nền tảng hoặc kiến trúc của chúng.

Bộ giao thức này bao gồm nhiều thành phần, nhưng hai phần quan trọng nhất là TCP (Transmission Control Protocol) và IP (Internet Protocol). IP chịu trách nhiệm về việc định địa chỉ và định tuyến các gói dữ liệu, đảm bảo rằng chúng được gửi đến đúng đích trong mạng. Mỗi thiết bị trên mạng được xác định bằng một địa chỉ IP duy nhất, giúp cho việc quản lý và tổ chức các kết nối trở nên hiệu quả.

Trong khi đó, TCP đảm bảo tính toàn vẹn và chính xác của dữ liệu trong quá trình truyền tải. Nó thực hiện điều này thông qua việc chia nhỏ dữ liệu thành các gói tin, đánh



số và đảm bảo rằng chúng được gửi và nhận theo đúng thứ tự. TCP còn có khả năng kiểm tra lỗi và yêu cầu truyền lại các gói tin bị mất, giúp đảm bảo rằng người nhận nhận được dữ liệu chính xác và đầy đủ.

Sự kết hợp giữa TCP và IP tạo ra một hệ thống mạng mạnh mẽ và linh hoạt, cho phép việc truyền tải thông tin diễn ra một cách hiệu quả và an toàn. Bộ giao thức này không chỉ hỗ trợ cho việc giao tiếp giữa các máy tính mà còn làm nền tảng cho nhiều ứng dụng và dịch vụ trực tuyến hiện nay, từ việc duyệt web đến truyền tải video, email và nhiều hình thức giao tiếp khác. TCP/IP đã và đang tiếp tục định hình cách mà chúng ta kết nối và chia sẻ thông tin trên toàn cầu.

2.1.3 Giao thức socket

Giao thức Socket là một công nghệ quan trọng trong mạng máy tính, cho phép giao tiếp giữa hai thiết bị qua mạng. Socket hoạt động như một điểm cuối trong kết nối mạng, thông qua đó các thiết bị có thể gửi và nhận dữ liệu. Về cơ bản, một socket là sự kết hợp của địa chỉ IP và cổng (port), giúp xác định duy nhất một ứng dụng hoặc dịch vụ cụ thể trên một thiết bị.

Socket thường dựa trên các giao thức như TCP hoặc UDP. Với TCP, socket đảm bảo kết nối ổn định, tuần tự và đáng tin cậy, phù hợp với các ứng dụng yêu cầu tính chính xác cao như truyền file hay email. Ngược lại, với UDP, socket hỗ trợ kết nối không đồng bộ, tốc độ nhanh, nhưng có thể bị mất dữ liệu, phù hợp cho các ứng dụng thời gian thực như truyền phát video hoặc trò chơi trực tuyến. Giao thức Socket đóng vai trò là cầu nối linh hoạt giữa các ứng dụng, cung cấp cơ sở cho việc xây dựng các ứng dụng mạng đa dạng và hiệu quả trong môi trường phân tán.

Trong khuôn khổ dự án, nhóm sử dụng phương thức socket TCP với hằng số SOCK_STREAM trong thư viện socket của Python để thiết lập kết nối tuần tự và đáng tin cậy giữ các peer với nhau và với tracker.

2.1.4 Giao thức HTTP

HTTP (HyperText Transfer Protocol) là giao thức tiêu chuẩn cho việc trao đổi thông tin trên World Wide Web, cho phép các máy tính kết nối với nhau và truyền tải dữ liệu trong môi trường Internet. Đây là giao thức nền tảng giúp trình duyệt web có thể tải và hiển thị trang web từ các server web, và cũng là giao thức chính được sử dụng trong các



ứng dụng web.

HTTP hoạt động theo mô hình client-server, trong đó **client** (thường là trình duyệt hoặc ứng dụng) gửi một yêu cầu (request) đến **server** và server sẽ phản hồi (response) với dữ liệu.

Một yêu cầu HTTP bao gồm **phương thức HTTP** (như GET, POST), **đường dẫn URL** đến tài nguyên, và **các header** chứa thông tin về yêu cầu. HTTP là giao thức không trạng thái, có nghĩa là mỗi yêu cầu được gửi đến server đều độc lập và không chứa thông tin về các yêu cầu trước đó.

Các phương thức HTTP phổ biến:

1. **GET**: Truy xuất dữ liệu từ server (dùng để lấy trang web hoặc tài nguyên khác).
2. **POST**: Gửi dữ liệu lên server (ví dụ, gửi dữ liệu từ một form đăng ký).
3. **PUT**: Cập nhật tài nguyên hiện có trên server.
4. **DELETE**: Xóa tài nguyên trên server.

Các phương thức này cho phép HTTP thực hiện nhiều loại tác vụ trên dữ liệu, giúp tổ chức các hoạt động web linh hoạt và dễ dàng quản lý.

Như vậy, trong dự án, để client thực hiện 1 yêu cầu truy vấn thông tin, phía client sẽ liên lạc với server tracker thông qua giao thức HTTP để yêu cầu phía server và nhận hồi đáp từ server thông qua giao thức trên.

2.2 Yêu cầu chức năng và phi chức năng

2.2.1 Yêu cầu chức năng

1. Đăng ký và đăng nhập
 - Đăng ký tài khoản: Người dùng có thể tạo tài khoản với tên đăng nhập và mật khẩu.
 - Đăng nhập: Người dùng đăng nhập để truy cập các tính năng của hệ thống.
 - Xác thực người dùng.
2. Quản lý tệp



- Chia sẻ tệp: Người dùng có thể chia sẻ tệp mới lên mạng P2P. Tệp sẽ được băm (hash) để tạo ra một mã định danh duy nhất.
- Tìm kiếm tệp: Người dùng có thể tìm kiếm các tệp đã được chia sẻ bởi những người dùng khác.
- Danh sách tệp chia sẻ: Hiện thị danh sách các tệp mà người dùng đã chia sẻ hoặc đang chia sẻ.

3. Tạo và Quản lý Tệp Torrent

- Tạo tệp torrent: Cho phép người dùng tạo tệp torrent từ các tệp chia sẻ. Tệp torrent sẽ chứa thông tin về tệp, kích thước, và danh sách các phần chia nhỏ của tệp.
- Quản lý các tệp torrent: Người dùng có thể xem và quản lý các tệp torrent đã tạo, bao gồm việc xóa, chỉnh sửa hoặc dừng chia sẻ.

4. Truyền Tệp (Download và Upload)

- Download tệp từ nhiều nguồn (Multi-source Download): Cho phép người dùng tải xuống tệp từ nhiều người dùng khác nhau cùng lúc, nhằm tăng tốc độ tải xuống.
- Upload tệp cho nhiều người dùng: Cho phép người dùng tải lên tệp cho nhiều peer cùng lúc, tối ưu tốc độ chia sẻ.
- Chia nhỏ tệp (Chunking): Mỗi tệp sẽ được chia nhỏ thành các phần, và người dùng sẽ tải hoặc tải lên từng phần riêng biệt.
- Tái cấu trúc tệp (Reassembly): Hệ thống sẽ tái cấu trúc các phần tải xuống thành tệp hoàn chỉnh trên máy người dùng.

5. Theo dõi Kết nối và Quản lý Peer

- Danh sách các peer: Hệ thống sẽ duy trì danh sách các peer đang có sẵn từng phần hoặc toàn bộ tệp.
- Tự động kết nối lại: Nếu kết nối tới một peer bị gián đoạn, hệ thống sẽ tự động thử lại và tìm các nguồn khác cho phần tệp còn thiếu.
- Theo dõi tiến độ tải xuống/tải lên: Người dùng có thể xem tiến độ của các phiên tải xuống/tải lên, bao gồm phần trăm hoàn thành và tốc độ truyền tải.

6. Tracker

- Tracker: Một server trung tâm sẽ đóng vai trò Tracker, theo dõi danh sách các client và các tệp họ đang chia sẻ, giúp các client kết nối nhanh chóng với nhau.

7. Hệ thống Kiểm tra và Sửa lỗi

- Kiểm tra tính toàn vẹn của tệp (Integrity Check): Khi tải xuống xong mỗi phần của tệp, hệ thống sẽ kiểm tra mã băm của phần đó để đảm bảo không có lỗi.
- Khôi phục khi lỗi: Nếu phát hiện lỗi trong một phần tải xuống, hệ thống sẽ tự động tải lại phần đó từ một peer khác.

8. Cơ chế Ưu tiên

- Ưu tiên tải lên/tải xuống: Người dùng có thể ưu tiên tải xuống hoặc tải lên cho một số tệp cụ thể.

9. Giao diện Người dùng (UI/UX)

- Giao diện quản lý phiên tải xuống/tải lên: Người dùng có thể dễ dàng quản lý, tạm dừng, tiếp tục hoặc hủy các phiên.
- Thông báo trạng thái: Hệ thống hiển thị các thông báo về trạng thái tải xuống/tải lên, tình trạng kết nối với các peer, và thông tin phiên hiện tại.

10. Thống kê và Báo cáo

- Thống kê chi tiết: Hệ thống cung cấp các thông tin về số lần tải xuống/tải lên, tốc độ trung bình, dung lượng dữ liệu đã chia sẻ.
- Lịch sử chia sẻ: Người dùng có thể xem lại lịch sử các tệp đã tải lên hoặc tải xuống, bao gồm các peer đã kết nối và các phiên đã hoàn thành.

2.2.2 Yêu cầu phi chức năng

1. Hiệu suất (Performance)

- Độ trễ thấp: Hệ thống phải đáp ứng yêu cầu tải xuống và tải lên với độ trễ tối thiểu để cung cấp trải nghiệm người dùng tốt.
- Tốc độ truyền tải ổn định: Đảm bảo tốc độ truyền tải giữa các peer duy trì ổn định ngay cả khi có nhiều người dùng truy cập đồng thời.

2. Khả năng mở rộng (Scalability)



- Hỗ trợ nhiều kết nối đồng thời: Hệ thống phải có khả năng xử lý hàng ngàn kết nối từ nhiều peer mà không ảnh hưởng đến hiệu suất.
- Khả năng mở rộng linh hoạt: Khi lượng người dùng hoặc số lượng tệp tăng lên, hệ thống phải dễ dàng mở rộng mà không làm suy giảm hiệu năng.

3. Tính sẵn sàng và Độ tin cậy (Availability and Reliability)

- Duy trì kết nối: Trong trường hợp một peer mất kết nối, hệ thống phải có khả năng tìm kiếm peer thay thế để tiếp tục phiên tải xuống/tải lên mà không bị gián đoạn.

4. Khả năng sử dụng (Usability)

- Giao diện thân thiện: Hệ thống phải cung cấp giao diện đơn giản, dễ hiểu để người dùng dễ dàng thao tác, quản lý các phiên tải xuống/tải lên.
- Thông báo trạng thái và cảnh báo: Hiển thị rõ ràng các thông báo về trạng thái của phiên tải xuống/tải lên, và cảnh báo người dùng khi có lỗi xảy ra.

5. Tối ưu hóa tài nguyên (Resource Optimization)

- Tối ưu dung lượng lưu trữ: Các tệp và dữ liệu cần được lưu trữ một cách hiệu quả để không lãng phí không gian lưu trữ.

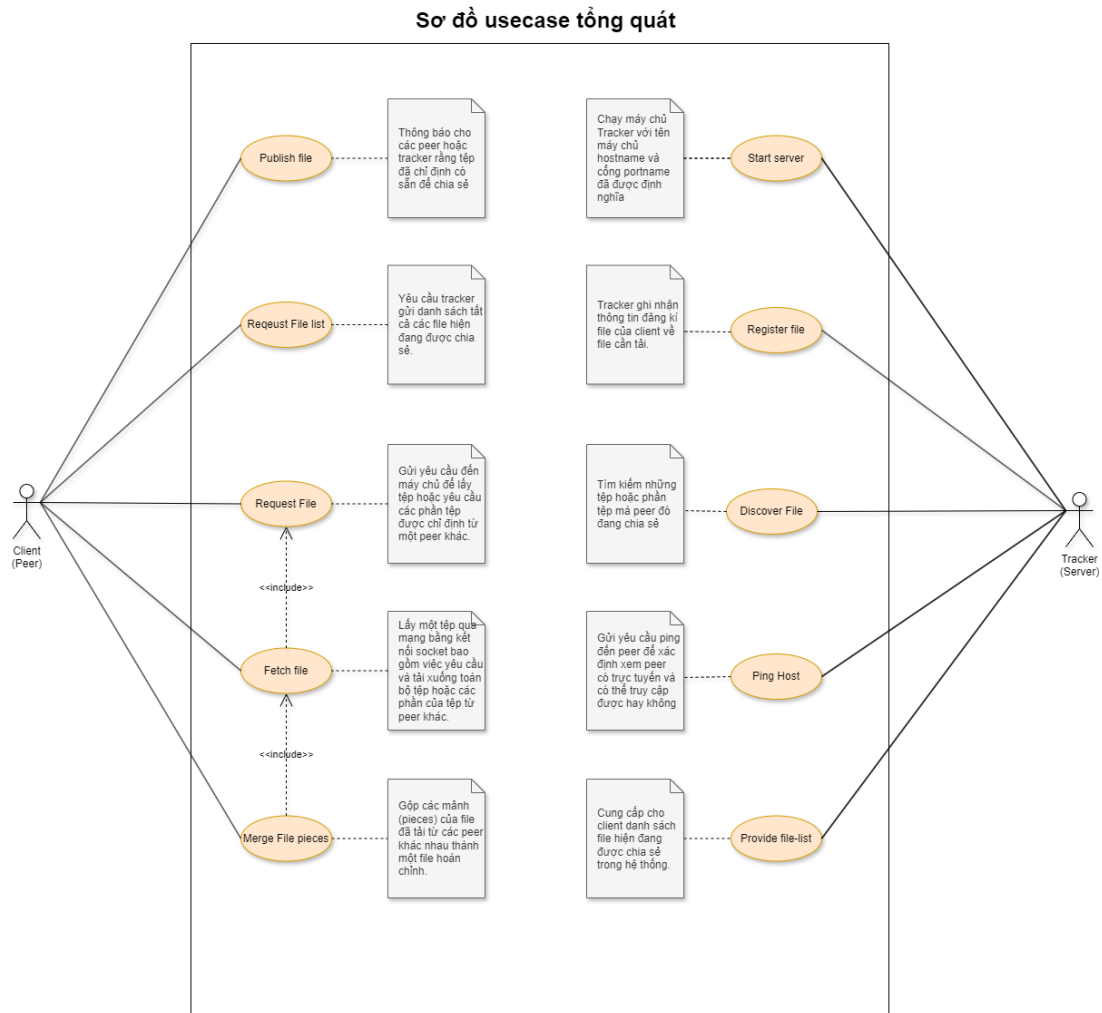
6. Khả năng mở rộng tính năng (Extensibility)

- Hỗ trợ tính năng mới: Hệ thống nên được thiết kế mở để có thể dễ dàng bổ sung tính năng mới mà không gây ảnh hưởng đến các chức năng hiện có.
- Hỗ trợ tích hợp API: Hệ thống nên có khả năng tích hợp với các API khác để mở rộng chức năng và giao tiếp với các ứng dụng hoặc dịch vụ khác.



3 Mô hình hệ thống

3.1 Usecase diagram



3.1.1 Publish File

Use Case Name	Publish File
Overview	Thông báo cho tracker hoặc các peer khác về tệp đã chỉ định có sẵn để chia sẻ.
Actors	Client (Peer)
Precondition	Client đã kết nối với hệ thống và có file chia sẻ.
Trigger	Peer muốn chia sẻ file.



Steps	<ol style="list-style-type: none">1. Peer thông báo cho tracker về file để chia sẻ.2. Tracker ghi nhận thông tin về file.
Post Condition	File của peer được ghi nhận trên tracker.
Exception Flow	Nếu tracker không sẵn sàng, peer đợi một khoảng thời gian và thử lại.

3.1.2 Request File List

Use Case Name	Request File List
Overview	Yêu cầu tracker gửi danh sách tất cả các file hiện đang được chia sẻ.
Actors	Client (Peer)
Precondition	Tracker đang hoạt động và có các file được chia sẻ.
Trigger	Peer muốn biết danh sách file hiện đang được chia sẻ.
Steps	<ol style="list-style-type: none">1. Peer gửi yêu cầu danh sách file đến tracker.2. Tracker trả về danh sách file hiện có.
Post Condition	Peer nhận được danh sách các file đang chia sẻ.
Exception Flow	Nếu không có file nào được chia sẻ, tracker trả về danh sách trống.

3.1.3 Request File

Use Case Name	Request File
Overview	Gửi yêu cầu đến máy chủ để lấy tệp hoặc yêu cầu các phần tệp được chỉ định từ một peer khác.



Actors	Client (Peer)
Precondition	Peer đã biết thông tin về file cần tải.
Trigger	Peer muốn tải một file cụ thể.
Steps	<ol style="list-style-type: none"> 1. Peer gửi yêu cầu tải file hoặc phần file. 2. Tracker hoặc peer phản hồi và bắt đầu gửi dữ liệu của file hoặc phần file.
Post Condition	Peer bắt đầu tải file hoặc phần file từ các nguồn có sẵn.
Exception Flow	Nếu phần file không có sẵn, tracker tìm kiếm các peer khác có phần file đó hoặc thông báo lỗi cho peer.

3.1.4 Fetch File

Use Case Name	Fetch File
Overview	Lấy một phần của file từ các peer khác để hoàn thành quá trình tải xuống.
Actors	Client (Peer)
Precondition	Peer đã gửi yêu cầu và biết các phần file có sẵn.
Trigger	Peer muốn tải phần file cụ thể.
Steps	<ol style="list-style-type: none"> 1. Peer yêu cầu tải một phần của file từ peer khác. 2. Peer khác gửi dữ liệu của phần file yêu cầu đến peer.
Post Condition	Peer tải xuống một phần của file từ peer khác thành công.
Exception Flow	Nếu peer không trực tuyến, tracker tìm các peer khác có phần file đó hoặc thông báo lỗi cho peer.



3.1.5 Merge File Pieces

Use Case Name	Merge File Pieces
Overview	Gộp các mảnh (pieces) của file đã tải từ các peer khác nhau thành một file hoàn chỉnh.
Actors	Client (Peer)
Precondition	Tất cả các mảnh của file cần tải đã được tải về từ các peer khác nhau và sẵn có trên hệ thống của client.
Trigger	Client hoàn tất việc tải về tất cả các mảnh của file từ các peer khác và cần gộp chúng lại thành một file hoàn chỉnh.
Steps	<ol style="list-style-type: none">1. Client kiểm tra danh sách các mảnh của file đã tải để đảm bảo rằng tất cả mảnh cần thiết đều có mặt.2. Client sắp xếp các mảnh theo thứ tự thích hợp.3. Client gộp các mảnh lại với nhau theo thứ tự để tạo thành file hoàn chỉnh.4. File hoàn chỉnh được lưu trữ trên hệ thống của client.
Post Condition	File hoàn chỉnh được tạo ra và sẵn sàng để sử dụng trên hệ thống của client.
Exception Flow	Nếu thiếu bất kỳ mảnh nào trong quá trình gộp, client sẽ gửi lại yêu cầu tải lại mảnh còn thiếu.

3.1.6 Start Server

Use Case Name	Start Server
Overview	Khởi động máy chủ tracker với các thông tin cấu hình.



Actors	Tracker (Server)
Precondition	Máy chủ đã được cài đặt và cấu hình.
Trigger	Người dùng hoặc hệ thống yêu cầu khởi động máy chủ tracker.
Steps	<ol style="list-style-type: none">1. Tracker được khởi động với các thông tin cấu hình cần thiết.2. Tracker bắt đầu lắng nghe các kết nối từ các peer.
Post Condition	Tracker hoạt động và sẵn sàng nhận kết nối.
Exception Flow	Nếu cấu hình sai, tracker dừng và báo lỗi cấu hình.

3.1.7 Register File

Use Case Name	Register File
Overview	Tracker nhận thông tin đăng ký file từ các peer và lưu trữ trong hệ thống.
Actors	Tracker (Server)
Precondition	Peer đã kết nối với tracker và gửi yêu cầu đăng ký file.
Trigger	Peer muốn chia sẻ file mới.
Steps	<ol style="list-style-type: none">1. Tracker nhận thông tin file từ peer.2. Tracker lưu trữ thông tin file trong cơ sở dữ liệu.
Post Condition	File được đăng ký thành công và có thể được yêu cầu bởi các peer khác.
Exception Flow	Nếu thông tin file bị lỗi, tracker yêu cầu peer gửi lại hoặc từ chối yêu cầu.



3.1.8 Discovering File

Use Case Name	Discovering File
Overview	Yêu cầu tracker tìm kiếm thông tin về các file, hoặc những phần tệp hiện có.
Actors	Client (Peer)
Precondition	Tracker có thông tin về các file hoặc những phần tệp được chia sẻ trong hệ thống.
Trigger	Peer muốn tìm kiếm các file hoặc mảnh file có sẵn để tải về.
Steps	<ol style="list-style-type: none">1. Peer gửi yêu cầu tìm kiếm file đến tracker.2. Tracker tìm kiếm và gửi danh sách các file hoặc phần file phù hợp với yêu cầu của peer.
Post Condition	Peer nhận được thông tin các file hoặc phần file có thể tải.
Exception Flow	Nếu không có file hoặc phần file nào phù hợp, tracker trả về thông báo không tìm thấy.

3.1.9 Ping Host

Use Case Name	Ping Host
Overview	Kiểm tra kết nối với một peer khác để đảm bảo rằng peer đó vẫn trực tuyến và sẵn sàng để chia sẻ file.
Actors	Client (Peer)
Precondition	Peer có thông tin địa chỉ của peer khác để ping.
Trigger	Peer muốn kiểm tra xem một peer khác có đang trực tuyến không.



Steps	<ol style="list-style-type: none">1. Peer gửi gói tin ping đến địa chỉ của peer khác.2. Peer khác trả lời bằng gói tin phản hồi.
Post Condition	Peer xác nhận trạng thái hoạt động của peer khác.
Exception Flow	Nếu không có phản hồi, peer xem như peer khác đã ngoại tuyến.

3.1.10 Provide File List

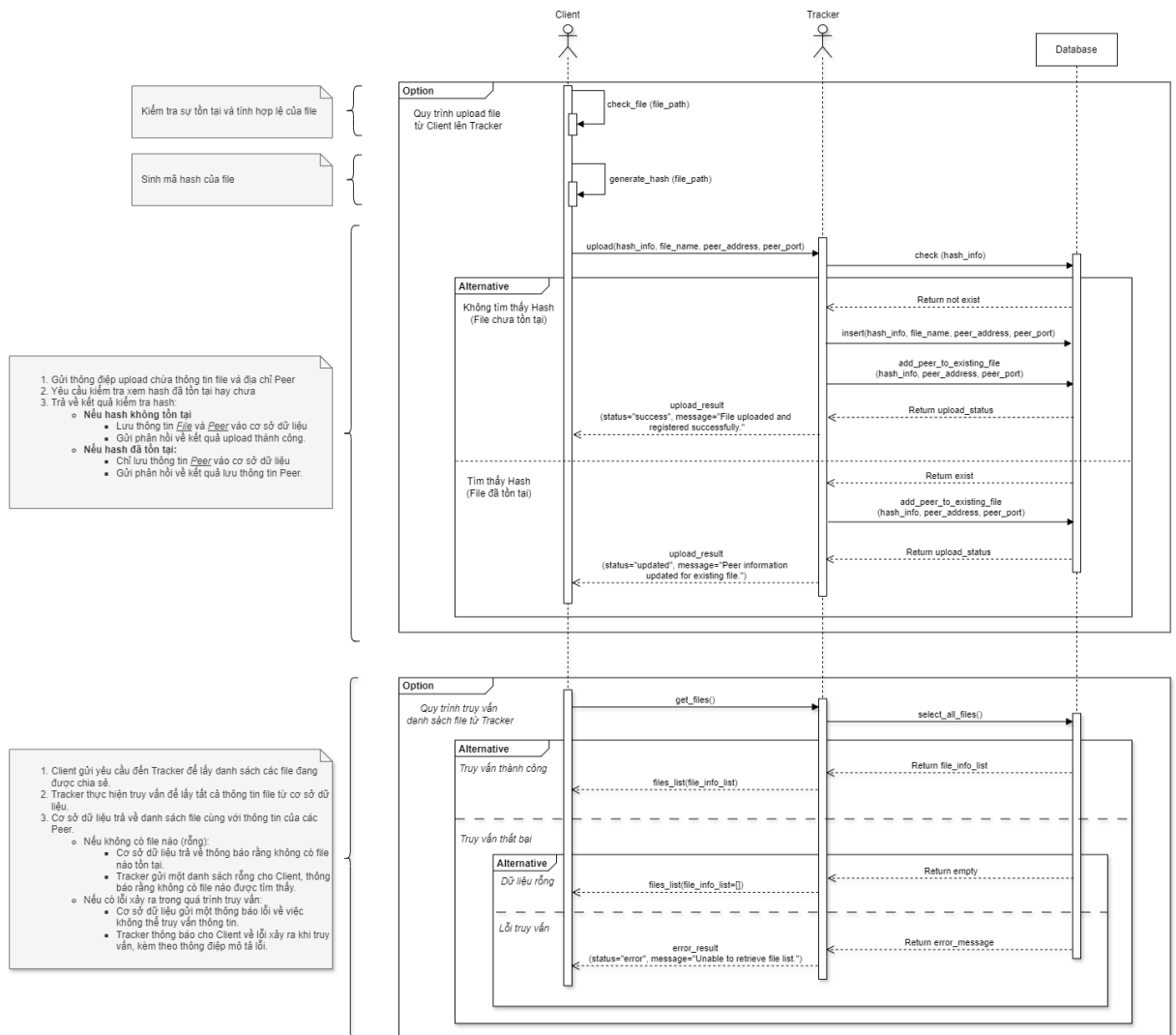
Use Case Name	Provide File List
Overview	Tracker cung cấp danh sách các file hiện đang có sẵn trong hệ thống.
Actors	Tracker (Server)
Precondition	Tracker đang hoạt động và có dữ liệu về các file đang được chia sẻ.
Trigger	Peer gửi yêu cầu danh sách file đến tracker.
Steps	<ol style="list-style-type: none">1. Tracker nhận yêu cầu từ peer về danh sách file.2. Tracker trả về danh sách các file hiện có.
Post Condition	Peer nhận được danh sách các file có thể tải về.
Exception Flow	Nếu không có file nào được chia sẻ, tracker trả về danh sách trống.



3.2 Sequence diagram

3.2.1 Quy trình kết nối và giao tiếp giữa Client và Tracker

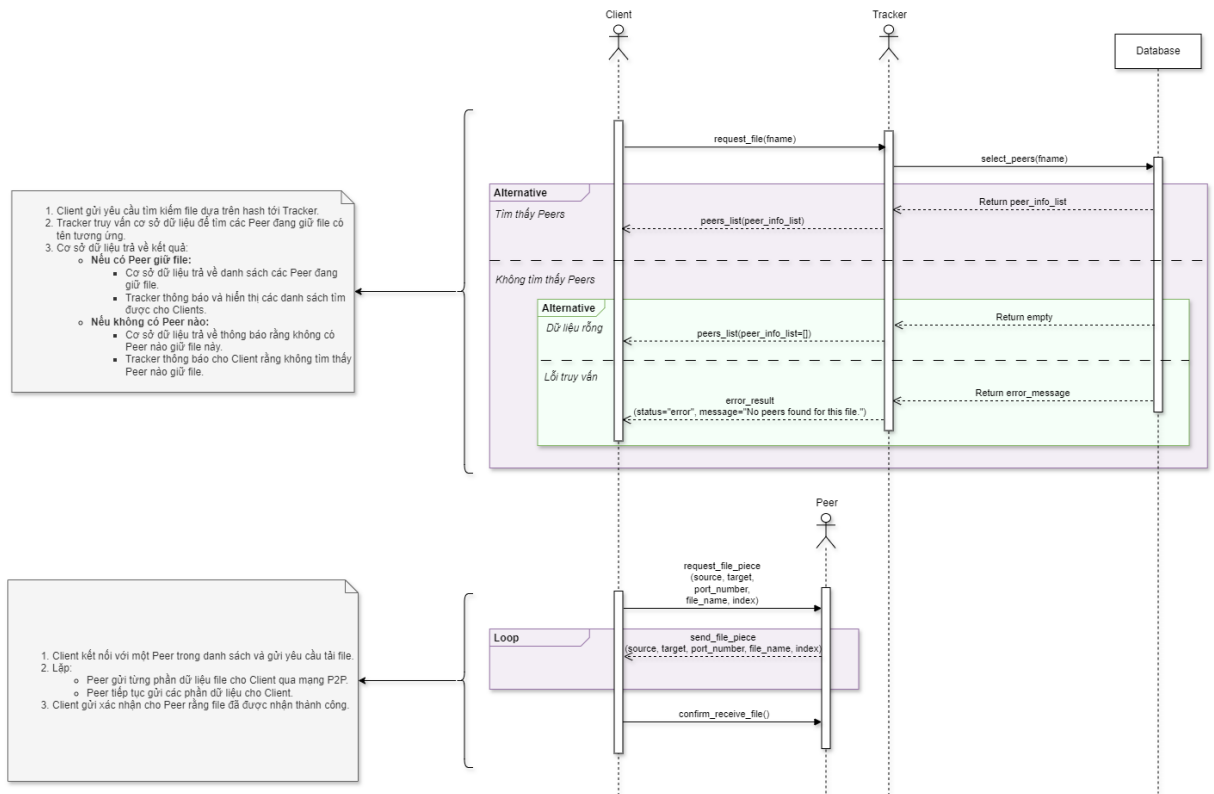
Module1: Kết nối và giao tiếp giữa Client và Tracker





3.2.2 Quy trình tải file giữa các peer

Module2: Quy trình chia sẻ và tải file giữa các Clients (Peers)





4 Các chức năng của ứng dụng

4.1 Phía Tracker

4.1.1 Chức năng quản lý

1. Các thao tác quản lý user: đăng ký, đăng nhập, tìm kiếm và cập nhật.
2. Các thao tác quản lý file:
 - Tìm kiếm: dựa trên tên file được hash, trả về tất cả các node đang giữ tên file đó.
 - Tạo mới: kiểm tra người dùng đang giữ file có tồn tại trong hệ thống hay không, nếu có thì tìm hoặc tạo **file** mới bao gồm các thông tin (infoHash, name, size, userId) cùng với **peer** mới bao gồm (IPAddress, port). 2 kiểu dữ liệu này cùng tạo ra 1 thực thể mới **peerHoldFile** với 2 thuộc tính là fileId và peerId mục đích để xác định peer với có id là "peerId" đang giữ file có id là "fileId".
3. Các thao tác quản lý peer (node): tìm kiếm và tạo mới với thông tin về IPAddress và port của peer. Đồng thời tạo ra các liên kết giữa peer với file (peerHoldFile) thông qua peerId và fileId.

4.1.2 Khởi tạo máy chủ

Chạy máy chủ Tracker với tên máy chủ hostname và cổng portname đã được định nghĩa sẵn.

4.1.3 Trả lời yêu cầu của các client

Phía client sẽ gửi lệnh bằng CLI để yêu cầu bắt đầu, dừng hoặc thông báo quá trình tải xuống. Hàm này sẽ xử lý yêu cầu này dựa trên:

- Phân tích lệnh được gửi đến. (GET, POST, PUT, DELETE)
- Thực hiện hành động phù hợp dựa trên lệnh (ví dụ: yêu cầu tracker tìm file).
- Phản hồi bằng tin nhắn hoặc hành động phù hợp.



4.1.4 Lưu trữ danh sách các client đang kết nối với server và danh sách file của các client đầy

Chức năng này sẽ lưu trữ hoặc cập nhật các thông tin quan trọng về một peer (client) trong danh sách. Mỗi peer đều đóng góp vào quá trình chia sẻ tệp, do đó, việc giữ thông tin chính xác là rất quan trọng để giao tiếp và truyền tệp hiệu quả.

4.1.5 Trình thông dịch dòng lệnh: 'discover', 'ping'

- `ping_host(hostname)`: Chức năng này sẽ gửi yêu cầu **PING** đến peer được xác định bằng hostname (thường là địa chỉ IP hoặc tên miền của peer) để xác định xem peer có trực tuyến và có thể truy cập được hay không.

Khi peer bắt được **PING** của server thì nó gửi ngược lại thông tin là **PONG**. Server nhận được **PONG** thì xác nhận peer còn sống.

Trong một nhóm torrent, điều cần thiết là phải biết peer nào đang tích cực sẵn sàng để giao tiếp và chia sẻ tệp. Nếu đối tác không phản hồi lệnh ping, client có thể quyết định dừng yêu cầu các phần tệp từ peer đó.

- `discover_file(filename)`: server yêu cầu client đưa thông tin về piece status của file có tên "filename" cho server cùng với (ip, port) của peer.

4.1.6 Chuyển đổi

- `torrent_to_magnet_link(torrent_file)`: chức năng này sẽ chuyển đổi tệp torrent thành các liên kết magnet link, đây là cách chia sẻ thông tin torrent nhỏ gọn và thân thiện hơn với người dùng.
- `magnet_link_to_torrent(magnet_link)`: chức năng này sẽ được sử dụng để chuyển đổi các liên kết magnet link trở lại thành một tệp torrent hoặc trích xuất thông tin có liên quan cần thiết để tạo thành một tệp.



4.2 Phía Peer (Node)

4.2.1 Kết nối tới máy chủ

Thiết lập kết nối đến máy chủ (có thể là tracker hoặc máy chủ trung tâm) tại hostname và port được chỉ định. Kết nối này cho phép client giao tiếp với máy chủ để thực hiện nhiều hoạt động khác nhau, chẳng hạn như yêu cầu tệp hoặc cập nhật trạng thái.

4.2.2 Thông báo cho server biết những file đang có

Hàm này lấy danh sách các tệp có sẵn để chia sẻ trên hệ thống cục bộ. Nó thường quét một thư mục được chỉ định để tìm các tệp mà client sẵn sàng chia sẻ với các peer khác trong mạng torrent.

4.2.3 Gửi thông tin để truy hồi file hoặc thông báo file hiện có

- `request_file_from_server(fname)`: Gửi yêu cầu đến máy chủ để lấy tệp được chỉ định bởi `fname`. Hàm này thường lấy siêu dữ liệu về tệp (metadata) và thông tin về các peer có sẵn chia sẻ tệp đó
- `request_file_pieces_from_peer(self_ip_address, ip_address, fname, pieces)`: nếu một peer ngắt kết nối trong quá trình truyền tải, chức năng này sẽ yêu cầu các phần tệp được chỉ định từ một peer khác. Nó xác định nguồn (địa chỉ IP của chính nó) và địa chỉ IP của peer đích mà nó tìm kiếm để lấy các phần của `fname`

4.2.4 Trình thông dịch dòng lệnh: 'publish', 'fetch'

- `publish_file(sock, fname)`: gửi thông báo hoặc cập nhật qua kết nối socket đã thiết lập (`sock`) để thông báo cho các peer hoặc tracker rằng tệp đã chỉ định (`fname`) có sẵn để chia sẻ
- `fetch_file(sock, fname)`: hàm này lấy một tệp qua mạng bằng kết nối socket (`sock`). Nó có thể bao gồm việc yêu cầu và tải xuống toàn bộ tệp hoặc các phần của tệp từ peer khác.

4.2.5 Xử lý khi truyền tải tệp

- `merge_file(fname, total_piece)`: sau khi lấy tất cả các phần của một tệp, hàm này kết hợp chúng thành một tệp hoàn chỉnh duy nhất. Nó sử dụng tên tệp (`fname`) và tổng số phần để tái tạo tệp gốc.

- `resume_send(ip_address, piece_idx)`: Nếu quá trình truyền tệp bị gián đoạn, chức năng này sẽ tiếp tục gửi một phần cụ thể của tệp tới đối tác tại `ip_address`, được xác định bởi `piece_idx`.

5 Các giao thức cho ứng dụng

5.1 Giữa các tính năng khi giao tiếp giữa client và server

Các tính năng có sự giao tiếp giữa client và server sẽ sử dụng chung giao thức kết nối là TCP vì các thao tác giữa client và server có thể chấp nhận độ trễ nhất định tuy nhiên yêu cầu ở server như là một trung gian nhằm quản lý cơ sở dữ liệu và kết nối giữa các client với nhau nên cần đảm bảo thông tin chính xác và tránh bị mất gói.

5.1.1 Ping

Khi Server sử dụng lệnh ping, Server sẽ gửi một yêu cầu đến Client với hostname tương ứng để kiểm tra tình trạng đang alive hay không. Nếu quá thời gian không có phản hồi, Server sẽ thông báo lại rằng Client đó đã offline và ngược lại.

5.1.2 Discover

Khi Server sử dụng lệnh discover, Server sẽ gửi một yêu cầu đến Client với hostname tương ứng để xin thông tin về các file đang có. Client sau khi nhận yêu cầu từ Server sẽ tiến hành truy vấn tên các file hiện có trong thư mục của mình sau đó gửi về lại cho Server

5.1.3 Publish

Khi một Client sử dụng lệnh publish, Client sẽ gửi một yêu cầu tới Server muốn lưu trữ một file trên máy có tên `cname` và đặt tên file này trên repository của mình là `fname`. Sau đó từ Server sẽ tiến hành cập nhật cơ sở dữ liệu và gửi một xác nhận về lại cho Client.

5.1.4 Fetch

Khi một Client sử dụng lệnh publish, Client sẽ gửi một yêu cầu tới Server muốn truy xuất một file có tên `fname` về máy mình. Server sẽ tiến hành truy vấn cơ sở dữ liệu để tìm ra các Client khác hiện đã publish file `fname` lên repository của mình và tiến hành



gửi lại cho Client đã gửi yêu cầu để Client đó tự tạo kết nối với Client đang giữ file để tiến hành chia sẻ file.

5.2 Giữa các tính năng khi giao tiếp giữa client và client

Các tính năng có sự giao tiếp giữa client và client sẽ sử dụng chung giao thức kết nối là TCP vì các thao tác giữa clients trong các chức năng này có thể chấp nhận độ trễ nhất định tuy nhiên yêu cầu việc dữ liệu khi nhận về cần đầy đủ, không bị mất gói.

5.2.1 Khởi tạo và lắng nghe kết nối

- `start_peer_server`: Mỗi client trong hệ thống sẽ có một server socket riêng để lắng nghe các kết nối từ các peer khác. Khi server socket này được khởi tạo, nó bắt đầu lắng nghe trên một địa chỉ IP và cổng đã chỉ định. Mỗi khi có một client kết nối đến, server sẽ chấp nhận kết nối đó và thực hiện xử lý yêu cầu từ client.

5.2.2 Kiểm tra trạng thái file tại peer

- `get_file_status_in_peer`: cho phép client gửi yêu cầu tới một peer khác và lấy về danh sách các mảnh file mà peer đó sở hữu. Yêu cầu này sẽ bao gồm thông tin nhận diện file (dưới dạng `info_hash`), giúp peer nhận dạng file được yêu cầu.

Peer nhận được yêu cầu sẽ trả về thông tin về trạng thái từng mảnh file (mảnh nào đã có, mảnh nào chưa), cùng với điểm ưu tiên (`point`) của peer. Thông tin này sẽ giúp client quyết định xem có nên tải dữ liệu từ peer này hay không.

5.2.3 Tải xuống các mảnh file từ peer

- `download_file_chunk_from_peer`: cho phép client gửi yêu cầu tải xuống các mảnh file cụ thể từ peer đã được xác định trước đó. Peer đáp ứng yêu cầu sẽ gửi lại dữ liệu của các mảnh file được yêu cầu. Client nhận được dữ liệu và ghi vào file đích tại một vị trí xác định, đồng thời cập nhật trạng thái các mảnh file đã tải xuống.

5.3 Cơ chế Phân Chia Chunk Dựa Trên Điểm Số

Việc xác định tải các chunk từ peer nào dựa trên điểm số của mỗi peer để tối ưu hóa việc chia sẻ tài nguyên và khuyến khích người dùng tích cực tham gia vào mạng. Điểm số phản ánh mức độ tải dữ liệu của từng peer và được sử dụng để xác định thứ tự ưu tiên khi phân chia chunk.



Hệ thống sử dụng priority queue (hàng đợi ưu tiên) để sắp xếp các peer dựa trên điểm số của họ. Điểm số này tăng lên khi peer tải xuống dữ liệu và giảm khi họ cung cấp dữ liệu cho các peer khác, nghĩa là những peer có nhu cầu tải file lớn hơn sẽ có điểm cao hơn và do đó sẽ có mức độ ưu tiên cao hơn trong việc phân chia chunk.



6 Hiện thực hệ thống

6.1 Tracker server

6.1.1 Định nghĩa

Trong hệ thống chia sẻ tệp ngang hàng (P2P) như torrent, tracker đóng vai trò quan trọng trong việc kết nối các peer (các thiết bị hoặc máy tính tham gia vào quá trình chia sẻ tệp) với nhau. Mặc dù bản thân tracker không chứa nội dung tệp, nó giúp các peer tìm thấy nhau và thiết lập các kết nối cần thiết để chia sẻ tệp một cách hiệu quả.

Khi một peer mới tham gia và yêu cầu tải xuống một tệp, tracker sẽ cung cấp cho nó danh sách các peer khác đang sở hữu các phần của tệp đó để nó có thể bắt đầu tải xuống. Đối với các mạng P2P mới, tracker giúp các peer mới tham gia mạng tìm thấy các peer khác, từ đó bắt đầu quá trình tải xuống. Một khi các kết nối giữa các peer được thiết lập, các peer có thể truyền dữ liệu trực tiếp với nhau mà không cần sự tham gia của tracker nữa.

Dự án lần này, nhóm sử dụng 1 tracker trung tâm đóng vai trò kết nối các peer thành viên, lưu trữ và truy vấn các thông tin cần thiết trong quá trình tải file.

6.1.2 Kiến trúc

Server tracker sử dụng trong dự án được xây dựng dựa trên mô hình MVC theo kiến trúc Monolithic hay MVC Monolithic.

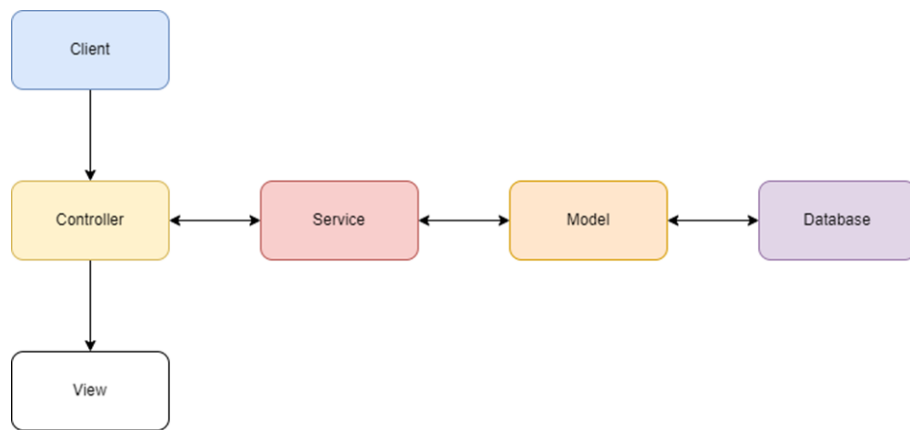
Khác với kiến trúc Microservice, kiến trúc MVC (Model-View-Controller) Monolithic là một kiểu thiết kế phần mềm được tổ chức theo mô hình MVC, trong đó toàn bộ ứng dụng được xây dựng và triển khai như một khối đơn lẻ (monolith), thay vì phân tán thành các dịch vụ độc lập như trong kiến trúc microservices. Kiến trúc MVC monolithic thường được sử dụng trong các ứng dụng web truyền thống và có cấu trúc rõ ràng, phân chia thành ba thành phần chính:

- **Model:** Quản lý dữ liệu, logic nghiệp vụ và các quy tắc của ứng dụng. Đây là phần kết nối trực tiếp với cơ sở dữ liệu, thực hiện các thao tác như lưu trữ, truy xuất và xử lý dữ liệu.
- **View:** Đóng vai trò là giao diện người dùng (UI) của ứng dụng, hiển thị dữ liệu từ



Model và nhận dữ liệu đầu vào từ người dùng. View không chứa logic nghiệp vụ mà chỉ tập trung vào việc hiển thị dữ liệu.

- **Controller:** Là trung gian giữa Model và View, tiếp nhận các yêu cầu từ người dùng qua View, xử lý và cập nhật dữ liệu bằng cách tương tác với Model, sau đó trả về kết quả cho View để hiển thị cho người dùng. Đối với hệ quản trị cơ sở dữ liệu, nhóm sử dụng MySQL là một DBMS hỗ trợ truy vấn có cấu trúc SQL để lưu trữ dữ liệu.



6.1.3 Cấu trúc

Để đáp ứng các chức năng cần thiết cho quá trình tải xuống, tracker cần lưu thông của User (người dùng), File và Peer.



Thuộc tính của User (định nghĩa trong user.entity.ts):

```
@Table
export class User extends Model<User> {
  @PrimaryKey
  @Default(UUIDV4)
  @Column(DataType.UUID)
  id: UUID;

  @AllowNull(false)
  @Column(DataType.STRING)
  username: string;

  @AllowNull(false)
  @Column(DataType.STRING)
  password: string;

  @AllowNull(false)
  @Column(DataType.STRING)
  fullName: string;

  @Default(0)
  @Column(DataType.INTEGER)
  point: number;
}
```

Thuộc tính của Peer (định nghĩa trong peer.entity.ts):

```
@Table
export class Peer extends Model<Peer> {
  @PrimaryKey
  @Default(UUIDV4)
  @AllowNull(false)
  @Column(DataType.UUID)
  id: UUID;

  @Column(DataType.STRING)
  IPaddress: string;

  @Column(DataType.INTEGER)
  port: number;

  @BelongsToMany(() => File, () => PeerHoldFile)
  files: File[];
}
```



Thuộc tính File (định nghĩa trong file.entity.ts):

```
@Table
export class File extends Model<File> {
  @PrimaryKey
  @Default(UUIDV4)
  @Column(DataType.UUID)
  id: UUID;

  @Column(DataType.STRING)
  name: string;

  @Column(DataType.INTEGER)
  size: number;

  @Unique
  @Column(DataType.STRING(255)) // or another suitable length for your data
  infoHash: string;

  @ForeignKey(() => User)
  @Column(DataType.UUID)
  userId: UUID;

  @BelongsTo(() => User)
  user: User;

  @BelongsToMany(() => Peer, () => PeerHoldFile)
  peers: Peer[];
}
```

Đối với quan hệ giữa Peer và File là quan hệ nhiều-nhiều (1 peer giữ nhiều file và 1 file có thể thuộc về nhiều peer chiếm giữ) ta định nghĩa bảng trung gian PeerHoldFile (peerHoldFile.entity.ts) để mapping 2 đối tượng

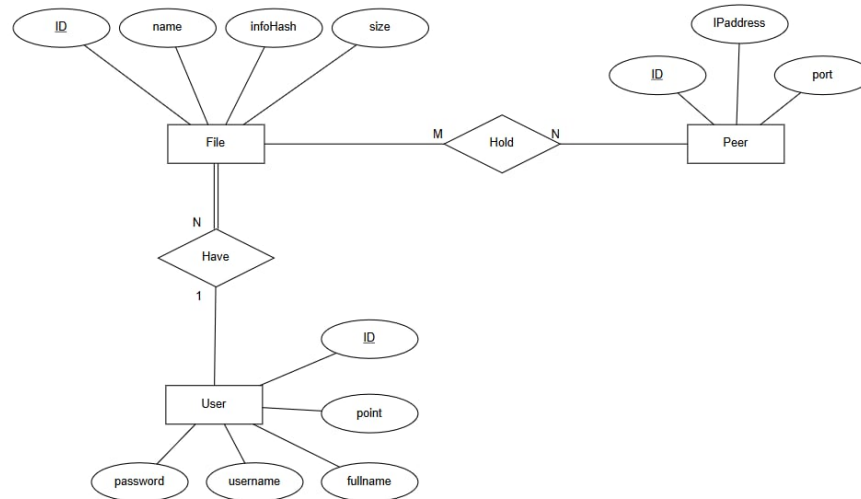
```
@Table
export class PeerHoldFile extends Model<PeerHoldFile> {
  @PrimaryKey
  @Default(UUIDV4)
  @AllowNull(false)
  @Column(DataType.UUID)
  id: UUID;

  @ForeignKey(() => File)
  @Column(DataType.UUID)
  fileId: UUID;

  @ForeignKey(() => Peer)
  @Column(DataType.UUID)
  peerId: UUID;
}
```



Mô hình ERD



Mô tả

Hệ thống cần lưu trữ các thông tin về file, user và peer trong mạng lưới. Đối với file, cần lưu các thông tin sau: mã file (danh định), tên file, kích thước và chuỗi hash.

Đối với **peer**, cần lưu: mã peer (danh định), IPaddress và port.

Đối với **user**, cần lưu: mã user (danh định), tài khoản, mật khẩu, tên đầy đủ và số điểm của user.

Mỗi peer có thể sở hữu nhiều file và mỗi file có nằm ở nhiều peer khác nhau nên quan hệ nắm giữ (Hold) là nhiều-nhiều.

Mỗi user có thể chứa nhiều file ở local nhưng 1 file chỉ thuộc về 1 user nên quan hệ có (Has) là một-nhiều.

6.1.4 Hiện thực

1. Fetch thông tin file bằng infoHash

• Lớp Controller

```
1 @Get('fetch')
2 async fetch(@Res() res, @Query('infoHash') infoHash: string) {
3     try {
4         const file = await this.fileService.search(infoHash);
5         this.response.initResponse(true, "Find information
6         successfully", file);
7         return res.status(HttpStatus.CREATED).json(this.
8         response);
```




```
7     } catch (error) {
8         this.logger.error(error.message, error.stack);
9         if (error instanceof InternalServerErrorException) {
10             this.response.initResponse(false, error.message,
11 null);
12             return res.status(HttpStatus.INTERNAL_SERVER_ERROR)
13 .json(this.response);
14         }
15
16         if (error instanceof BadRequestException) {
17             this.response.initResponse(false, error.message,
18 null);
19             return res.status(HttpStatus.BAD_REQUEST).json(this
20 .response);
21         }
22
23         this.response.initResponse(false, "An error occurred.
24 Please try again.", null);
25         return res.status(HttpStatus.INTERNAL_SERVER_ERROR).
26 json(this.response);
27     }
28 }
```

• Lớp Service

```
1 async search(infoHash: string) {
2     let findOption: FindOptions = {
3         include: [
4             {
5                 model: Peer,
6                 through: { attributes: [] },
7                 attributes: ['IPAddress', 'port']
8             },
9             {
10                 model: User,
11                 attributes: ['id', 'fullname', 'point']
12             }
13         ]
14     };
15
16     if (infoHash) {
17         findOption.where = { infoHash };
18     }
19 }
```



```
20     return await this.fileRepository.findAll(findOption);  
21 }
```

Phương thức fetch trong lớp Controller là một API endpoint xử lý yêu cầu HTTP GET tới đường dẫn /fetch với tham số truy vấn infoHash. Nó có nhiệm vụ:

- **Bước 1:** Lấy tham số infoHash từ truy vấn của yêu cầu.
- **Bước 2:** Sử dụng phương thức search của fileService để tìm kiếm thông tin tệp dựa trên infoHash đã được cung cấp.
- **Bước 3:** Sau khi nhận được kết quả từ fileService, phản hồi lại cho client với trạng thái thành công (CREATED) và dữ liệu của tệp được tìm thấy.

Nếu có lỗi xảy ra, phương thức sẽ bắt lỗi và trả về các phản hồi phù hợp:

- Internal Server Error: Nếu lỗi là InternalServerErrorException, trả về phản hồi với trạng thái 500 INTERNAL_SERVER_ERROR.
- Bad Request: Nếu lỗi là BadRequestException, trả về phản hồi với trạng thái 400 BAD_REQUEST.
- Các lỗi khác: Trả về thông báo lỗi chung và trạng thái 500 INTERNAL_SERVER_ERROR.

Phương thức fetch sử dụng **@Res()** để truy cập đối tượng phản hồi và **@Query('infoHash')** để lấy giá trị infoHash từ tham số truy vấn.

Phương thức search trong lớp Service thực hiện việc truy vấn thông tin từ cơ sở dữ liệu, dựa trên mã định danh infoHash:

- **Thiết lập tùy chọn truy vấn:** Sử dụng đối tượng FindOptions để chỉ định các bảng liên quan và các trường sẽ lấy từ mỗi bảng.
 - **Peer:** Bao gồm bảng Peer liên kết với File thông qua through, chỉ lấy các trường IPAddress và port.
 - **User:** Bao gồm bảng User với các trường id, fullname, và point.
- **Áp dụng điều kiện tìm kiếm:** Nếu infoHash có giá trị, thiết lập điều kiện tìm kiếm để tìm các tệp có infoHash tương ứng.
- **Truy vấn cơ sở dữ liệu:** Gọi this.fileRepository.findAll(findOption) để tìm tất cả các tệp phù hợp với điều kiện, sau đó trả về kết quả cho Controller.



6.2 Client side

6.2.1 Download

- Cách 1: Giải mã Magnet Link

Đầu tiên, hàm bắt đầu bằng cách giải mã liên kết magnet để lấy các thông tin như `info_hash`, tên file và kích thước file.

```
1 def decode_magnet_link(magnet_link):
2     # Parse the magnet link
3     parsed = urllib.parse.urlparse(magnet_link)
4     params = urllib.parse.parse_qs(parsed.query)
5
6     # Extract components
7     info_hash = params.get("xt", [None])[0]
8     if info_hash and info_hash.startswith("urn:btih:"):
9         info_hash = info_hash[9:] # Remove 'urn:btih:' prefix
10
11     filename = params.get("dn", [None])[0]
12     trackers = params.get("tr", [])
13     web_seeds = params.get("ws", [])
14     file_size = params.get("xl", [None])[0]
15     if file_size:
16         file_size = int(file_size)
17
18     # Return extracted info as a dictionary
19     return {
20         "info_hash": info_hash,
21         "filename": filename,
22         "trackers": trackers,
23         "web_seeds": web_seeds,
24         "file_size": file_size
25     }
```

Hàm `decode_magnet_link` được thiết kế để giải mã một liên kết magnet nhằm trích xuất các thông tin quan trọng cần thiết cho quá trình tải xuống file trong hệ thống peer-to-peer. Cụ thể, hàm này lấy mã băm của file (`info_hash`), tên file (`filename`), danh sách các tracker (`trackers`) để tìm kiếm peer, danh sách các web seeds (`web_seeds`) nếu có, và kích thước file (`file_size`).

- Cách 1: Giải mã file torrent

Từ file torrent ta lấy thông tin `infoHash`, tên file và kích thước file.



```
1 def extract_torrent_info(torrent_file):
2     # Check if torrent file exists
3     if not os.path.isfile(torrent_file):
4         raise FileNotFoundError(f"File '{torrent_file}' not exist."
5     )
6
7     # Create torrent object from file
8     torrent = Torrent.from_file(torrent_file)
9
10    # Extract necessary information
11    info_hash = torrent.info_hash # Get info hash
12    filename = torrent.name       # Get file name
13    file_size = sum(file.length for file in torrent.files) # Total
14    size of all files
15    trackers = torrent.announce_urls # Get list of trackers
16    web_seeds = torrent.webseeds     # Get list of web seeds
17
18    # Return extracted info as a dictionary
19    return {
20        "info_hash": info_hash,
21        "filename": filename,
22        "trackers": trackers,
23        "web_seeds": web_seeds,
24        "file_size": file_size
25    }
```

6.2.2 Lấy thông tin về peers từ server

Sử dụng `get_file_info_from_server(info_hash)` để nhận danh sách các peers có khả năng chia sẻ file dựa trên `info_hash`.

```
1 def get_file_info_from_server(info_hash):
2     response = requests.get(f'http://localhost:3000/v1/file/fetch?
3     hash_info={info_hash}')
4     try:
5         # Try to parse the response as JSON regardless of the content
6         type header
7         json_response = response.json()
8
9         # Extract values with defaults in case fields are missing
10        success = json_response.get('success', False)
11        data = json_response.get('data', {})
```



```
10     message = json_response.get('message', "No message provided")
11
12     return {'success': success, 'data': data, 'message': message}
13
14 except ValueError:
15     # If JSON decoding fails, handle it as a non-JSON response
16     print("Received non-JSON response:", response.text)
17     return {'success': False, 'data': {}, 'message': "Non-JSON
response"}
```

Hàm `get_file_info_from_server` gửi một yêu cầu GET đến server với tham số `infoHash` để lấy thông tin về file. Hàm phân tích phản hồi từ server dưới dạng JSON.

Nếu thành công, nó sẽ trích xuất các trường `success`, `data`, và `message` từ phản hồi JSON. Các trường này có giá trị mặc định nếu không có trong phản hồi.

Nếu phản hồi không thể phân tích thành JSON (ví dụ: phản hồi không phải JSON), hàm sẽ trả về một thông báo lỗi và dữ liệu trống.

Hàm này cung cấp thông tin cần thiết về các peers, giúp hàm `download` xác định các nguồn có sẵn để tải file.

6.2.3 Kiểm tra Trạng thái File từ Các Peers

Hàm `get_file_status_in_peer(ips[i], ports[i], info_hash)` được sử dụng để kiểm tra trạng thái file trên mỗi peer. Hàm này kết nối tới peer thông qua socket, gửi yêu cầu về trạng thái file, và nhận phản hồi về trạng thái các phần file mà peer đang sở hữu.

```
1 def get_file_status_in_peer(peer_ip, peer_port, info_hash):
2     try:
3         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
4
5             s.connect((peer_ip, peer_port))
6             print(f"Connected to {peer_ip}:{peer_port}")
7
8             request = {
9                 'type': 'GET_FILE_STATUS',
10                 'info_hash': info_hash
11             }
12
13             s.sendall(json.dumps(request).encode('utf-8'))
```



```
14
15     response_data = s.recv(4096)
16     response = json.loads(response_data.decode('utf-8'))
17     # print(f"and get pieces: {response['pieces_status']}")
18     if response['type'] == 'RETURN_FILE_STATUS' and response['
info_hash'] == info_hash:
19         pieces_status = response['pieces_status']
20         point = response['point']
21         return peer_ip, peer_port, pieces_status, point
22     else:
23         return None, None, None, 0
24 except (socket.error, ConnectionRefusedError, TimeoutError) as e:
25     print(f"Connection error: {e}")
26     return None, None, None, 0
```

6.2.4 Quản lý Trạng thái Tải Xuống

Hàm kiểm tra xem file đã được tải xuống hay chưa và tạo thư mục cùng file JSON để lưu trữ trạng thái của các phần file nếu chúng chưa tồn tại. Trạng thái hiện tại của các phần file được lưu trữ trong file status.json.

```
1 points = {}
2
3 for i in range(len(ips)):
4     points[(ips[i], ports[i])] = point[i]
5
6 peers = [(ips[i], ports[i]) for i in range(len(ips))]
7
8 selected_chunks = {peer: [] for peer in peers}
9
10 num_chunks = len(pieces[0])
11
12 currentStatus = []
13 if not os.path.exists(f'files/{info_hash}'):
14     os.makedirs(f'files/{info_hash}')
15
16 if not os.path.exists(f'files/{info_hash}/status.json'):
17     with open(f'files/{info_hash}/status.json', 'w') as f:
18         json.dump({"fileName": fileName, "piece_status": [
19             0 for _ in range(fileSize // PIECE_SIZE)
20         ]}, f)
21 if not os.path.exists(f'storage/{fileName}')
```



```
22     with open(f'storage/{fileName}', 'w') as f:
23         pass
24
25 with open(f'files/{info_hash}/status.json', 'r') as f:
26     data = json.load(f)
27
28 if data != None:
29     currentStatus = data["piece_status"]
```

6.2.5 Lựa chọn Peer Tốt Nhất

Trong quá trình tải xuống, hàm xác định peer tốt nhất cho từng phần file dựa trên trạng thái hiện tại và điểm số (priority) của từng peer. Mỗi lần người dùng tải xuống một hoặc nhiều chunk, điểm số của họ sẽ được cộng thêm tương ứng với số lượng chunk đã tải xuống. Ngược lại, khi một peer chia sẻ chunk cho người khác, điểm số của họ sẽ bị trừ đi bấy nhiêu.

```
1 for chunk_index in range(num_chunks):
2     if data != None and currentStatus[chunk_index] == 1:
3         continue
4     best_peer = None
5     best_priority = -1000
6
7     for peer_index, peer_chunks in enumerate(pieces):
8         if peer_chunks[chunk_index] == 1:
9             peer = peers[peer_index]
10            priority = points[peer]
11
12            if priority > best_priority:
13                best_priority = priority
14                best_peer = peer
15
16 if best_peer is not None:
17     selected_chunks[best_peer].append(chunk_index)
```

Hệ thống sử dụng một mảng để theo dõi trạng thái của các phần file (piece status) và điểm số của từng người dùng. Dựa trên các điểm số này, một hàng đợi ưu tiên (priority queue) sẽ được sử dụng để sắp xếp mức độ ưu tiên tải cho các peer. Điều này có nghĩa là những peer có điểm số cao hơn sẽ được ưu tiên tải xuống nhiều hơn, từ đó tạo ra mảng chunk_list cho từng peer, giúp tối ưu hóa quá trình tải xuống và khuyến khích người dùng chia sẻ file hiệu quả hơn.



6.2.6 Tải Xuống File

Sử dụng multithreading, hàm khởi động nhiều luồng để tải xuống các phần file từ các peers đã được chọn. Mỗi luồng thực hiện chức năng `download_file_chunk_from_peer`. Chức năng này thực hiện việc tải xuống một hoặc nhiều phần của file từ một peer trong mạng peer-to-peer.

Đầu tiên, hàm tạo một kết nối socket tới địa chỉ IP và cổng của peer, sau đó gửi yêu cầu dưới dạng JSON để lấy các phần file cần thiết. Khi nhận được phản hồi từ peer, hàm kiểm tra xem phản hồi có hợp lệ hay không; nếu đúng, nó trích xuất dữ liệu của các phần và ghi chúng vào vị trí chính xác trong file trên đĩa, đồng thời cập nhật trạng thái của từng phần đã tải xuống thành công.

Cuối cùng, hàm ghi lại trạng thái mới vào tệp `status.json`, giúp theo dõi quá trình tải xuống. Nếu phản hồi không hợp lệ, hàm sẽ in ra thông báo cảnh báo. Hàm này giúp tối ưu hóa quá trình tải xuống bằng cách phân chia file thành các phần nhỏ và tải từ nhiều peer khác nhau.

```
1 threads = []
2 for (ip, port), chunks in selected_chunks.items():
3     peer_thread = threading.Thread(target=download_file_chunk_from_peer,
4     args=(ip, port, info_hash, chunks, f"storage/{fileName}"), daemon=
5     True)
6     peer_thread.start()
7     threads.append(peer_thread)
8
9 for thread in threads:
10     thread.join()
```

```
1 def download_file_chunk_from_peer(peer_ip, peer_port, info_hash,
2 chunk_list, file_path):
3     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
4         s.connect((peer_ip, peer_port))
5         # print(f"Connected to {peer_ip}:{peer_port}")
6
7         request = {
8             'type': 'GET_FILE_CHUNK',
9             'info_hash': info_hash,
10            'chunk_list': chunk_list
11        }
```




```
12     s.sendall(json.dumps(request).encode('utf-8'))
13
14     response_data = s.recv(4096)
15     response = json.loads(response_data.decode('utf-8'))
16     # print(f"and get chunks: {response['chunk_data']}")
17     if response['type'] == 'RETURN_FILE_CHUNK' and response['info_hash']
    == info_hash:
18         chunk_data = response['chunk_data']
19
20         with file_lock:
21             with open(f"files/{info_hash}/status.json", "r") as file:
22                 data = json.load(file)
23                 currentStatus = data["piece_status"]
24                 fileName = data["fileName"]
25
26             with open(file_path, "r+b") as f:
27                 for i, chunk in enumerate(chunk_data):
28                     f.seek(chunk_list[i] * PIECE_SIZE)
29                     f.write(chunk.encode('latin1'))
30                     currentStatus[chunk_list[i]] = 1
31
32             print(currentStatus)
33
34             with open(f"files/{info_hash}/status.json", "w") as file:
35                 json.dump(
36                     {
37                         "fileName": fileName,
38                         "piece_status": currentStatus
39                     },
40                     file
41                 )
42     else:
43         print("Has been received invalid response from peer")
```

Cập nhật Trạng thái Người Dùng: Sau khi quá trình tải xuống hoàn tất, hàm đọc trạng thái từ file status.json, tính toán điểm số người dùng mới và cập nhật thông tin người dùng trong hệ thống.

```
1 dataInJsonFile = {}
2 with open(f"files/{info_hash}/status.json", "r") as file:
3     data = json.load(file)
4     dataInJsonFile = data
5     if all(status == 0 for status in data["piece_status"]):
```



```
6         return
```

6.2.7 Upload_full_file

Hàm này được thiết kế để tải lên một file hoàn chỉnh từ thư mục lưu trữ vào hệ thống quản lý torrent.

```
1 def upload_full_file(filename):
2     filesize = os.path.getsize(f"storage/{filename}")
3     infohash = helper.generate_hash_info(f"storage/{filename}")
4
5     os.makedirs(f"files/{infohash}", exist_ok=True)
6     with open(f'files/{infohash}/status.json', 'w') as file:
7         json.dump(
8             {
9                 "fileName": filename,
10                "piece_status": [1 for _ in range(filesize // PIECE_SIZE
11                ) ]
12            },
13            file
14        )
15
16     f = open("userId.txt", "r")
17     userId = f.read()
18     helper.upload_file(infohash, filename, filesize, self_ip_address,
19     self_port, userId)
20     helper.file_to_torrent(f"storage/{filename}", "http://localhost:3000/
21     v1", filesize)
22
23     return helper.generate_magnet_link(infohash, filename, filesize, "
24     http://localhost:3000/v1", None)
```

Quá trình thực hiện hàm diễn ra theo các bước sau:

- **Xác định kích thước file:** Hàm bắt đầu bằng việc lấy kích thước của file thông qua `os.path.getsize`, với đường dẫn là `storage/filename`. Điều này giúp xác định dung lượng của file cần tải lên.
- **Tạo infohash:** Tiếp theo, hàm gọi `generate_hash_info` để tạo ra một (infohash) cho file, giúp xác định duy nhất file trong hệ thống



- **Tạo thư mục cho file:** tạo ra một thư mục mới tương ứng với infohash trong thư mục files. Nếu thư mục đã tồn tại, lệnh này sẽ không gây lỗi.
- **Lưu trữ trạng thái file:** Trong thư mục vừa tạo, hàm mở hoặc tạo file status.json và lưu trữ tên file cùng với trạng thái các phần (pieces) của file. Trạng thái được khởi tạo với giá trị 1 cho mỗi phần, chỉ ra rằng tất cả các phần đã được tải lên.
- **Đọc userId:** Hàm mở file userId.txt để đọc userId, xác định người dùng hiện tại đang thực hiện upload.
- **Gửi file lên server:** Tiếp theo, hàm gọi upload_file(infohash, filename, filesize, self_ip_address, self_port, userId để gửi yêu cầu tải lên một file đến server thông qua HTTP POST. Trong quá trình thực hiện, hàm gửi yêu cầu đến địa chỉ file/upload, kèm theo thông tin như infoHash, tên file, địa chỉ IP và cổng của peer, kích thước file và ID người dùng.
Sau khi gửi yêu cầu, hàm cố gắng phân tích phản hồi từ server dưới dạng JSON để xác định xem yêu cầu có thành công hay không, đồng thời trích xuất dữ liệu và thông điệp từ phản hồi. Nếu phản hồi không phải là JSON, hàm sẽ trả về thông tin về lỗi. Kết quả cuối cùng là một từ điển chứa thông tin về trạng thái tải lên, giúp đảm bảo rằng quá trình tải lên diễn ra suôn sẻ và thông tin phản hồi được xử lý hiệu quả.
- **Chuyển đổi file thành định dạng torrent:** Sau khi upload thành công, hàm gọi file_to_torrent để chuyển đổi file thành định dạng torrent và gửi nó đến địa chỉ URL chỉ định.
- **Tạo liên kết magnet:** Cuối cùng, hàm trả về một liên kết magnet thông qua generate_magnet_link. Liên kết này cho phép người dùng chia sẻ file trong hệ thống torrent.

6.2.8 Handle_client

Hàm handle_client được thiết kế để xử lý các yêu cầu từ client trong một hệ thống chia sẻ file. Khi một client kết nối, hàm bắt đầu bằng cách nhận dữ liệu từ socket của client và giải mã nó từ định dạng JSON. Tùy thuộc vào loại yêu cầu (type), hàm thực hiện các thao tác khác nhau.

Hàm bắt đầu bằng cách nhận dữ liệu từ socket của client thông qua client_socket.recv(1024), sau đó giải mã dữ liệu thành định dạng JSON bằng json.loads(data).



Xử lý yêu cầu GET_FILE_STATUS

```
1 if request['type'] == 'GET_FILE_STATUS':
2     info_hash = request['info_hash']
3
4     response = {
5         'type': 'RETURN_FILE_STATUS',
6         'info_hash': info_hash,
7         'fName': None,
8         'pieces_status': [],
9         'point': 0
10    }
11
12    print(info_hash)
13
14    with open(f'files/{info_hash}/status.json', 'r') as f:
15        data = json.load(f)
16
17    if not data:
18        client_socket.sendall(json.dumps(response).encode('utf-8'))
19        return
20
21    file_name = get_filename_in_folder(f"files/{info_hash}")
22
23
24    f = open("userId.txt", "r")
25    userId = f.read()
26
27    point = helper.search_by_id(userId)["data"]["point"]
28
29    response = {
30        'type': 'RETURN_FILE_STATUS',
31        'info_hash': info_hash,
32        'fName': file_name,
33        'pieces_status': data['piece_status'],
34        'point': point
35    }
36
37    client_socket.sendall(json.dumps(response).encode('utf-8'))
```

Khi nhận được yêu cầu GET_FILE_STATUS, hàm sẽ lấy giá trị info_hash từ yêu cầu. Một phản hồi mặc định được khởi tạo, chứa thông tin về file, trạng thái các phần (pieces),



và điểm số (point).

Hàm mở file status.json nằm trong thư mục tương ứng với info_hash để đọc trạng thái của file. Nếu không tìm thấy file, hàm gửi phản hồi mặc định về cho client.

Nếu file tồn tại, hàm sẽ gọi hàm get_filename_in_folder để lấy tên file và đọc userId từ file userId.txt.

Điểm số của người dùng được truy xuất bằng cách gọi search_by_id(userId), và một phản hồi chi tiết chứa tên file, trạng thái các phần, và điểm số được gửi về cho client.

Xử lý yêu cầu GET_FILE_STATUS

Khi yêu cầu là GET_FILE_CHUNK, hàm sẽ lấy info_hash và danh sách các chunk từ request.

```
1 elif request['type'] == 'GET_FILE_CHUNK':
2     info_hash = request['info_hash']
3     chunk_list = request['chunk_list']
4     chunk_data = []
5
6     response = {
7         'type': 'RETURN_FILE_CHUNK',
8         'info_hash': info_hash,
9         'chunk_data': []
10    }
11
12    with open(f'files/{info_hash}/status.json', 'r') as f:
13        data = json.load(f)
14
15    if not data:
16        client_socket.sendall(json.dumps(response).encode('utf-8'))
17        return
18
19    file_name = get_filename_in_folder(f"files/{info_hash}")
20
21    try:
22        with open(f"storage/{file_name}", "rb") as f:
23            for chunk_index in chunk_list:
24                f.seek(chunk_index * PIECE_SIZE)
25                data = f.read(PIECE_SIZE)
26                chunk_data.append(data.decode('latin1'))
27
28        f = open("userId.txt", "r")
```



```
29         userId = f.read()
30         userPoint = helper.search_by_id(userId)["data"]["point"]
31         newPoint = userPoint - (len(chunk_list))
32         helper.update_user(newPoint, userId)
33     except FileNotFoundError:
34         print(f"File {file_name} does not exist.")
35         client_socket.sendall(json.dumps(response).encode('utf-8'))
36         return
37
38     response['chunk_data'] = chunk_data
39
40     client_socket.sendall(json.dumps(response).encode('utf-8'))
```

Một phản hồi mặc định chứa thông tin về chunk được khởi tạo.

File status.json được mở để đọc trạng thái của file. Nếu không tìm thấy file, phản hồi mặc định sẽ được gửi về cho client.

Hàm sẽ lấy tên file bằng cách gọi get_filename_in_folder và mở file trong thư mục lưu trữ để đọc dữ liệu cho các chunk đã yêu cầu.

Dữ liệu từ các chunk được lưu trữ trong một danh sách và điểm số của người dùng được cập nhật bằng cách trừ đi số lượng chunk đã tải xuống.

Cuối cùng, hàm gửi dữ liệu chunk đã đọc trở lại client.

Xử lý Yêu cầu PING

Nếu yêu cầu từ client là PING, hàm sẽ gửi lại phản hồi PONG để xác nhận kết nối và sự hiện diện của server.

```
1 elif request['type'] == 'PING':
2     response = {
3         'type': 'PONG'
4     }
5     client_socket.sendall(json.dumps(response).encode('utf-8'))
```



7 Hướng dẫn sử dụng

7.1 Khởi động server tracker

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE SEARCH ERROR
lekhoa@MacBook-Pro-cua-Le tracker_server % npm run start:dev
Debugger listening on ws://127.0.0.1:50788/24e6d3d8-35d9-40fc-b176-02a0ce896f50
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.

> media@0.0.1 start:dev
> nest start --watch

Debugger listening on ws://127.0.0.1:50791/975a4e79-0762-4b02-aaea-02b1fa10f92a
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```

Khi terminal hiển thị như hình bên dưới, server đã sẵn sàng phục vụ. Nếu server hiện lỗi chưa có package, sử dụng lệnh “npm i” để tải các packages được định nghĩa trong package.json.

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE SEARCH ERROR
lekhoa@MacBook-Pro-cua-Le tracker_server % npm run start:dev
Debugger listening on ws://127.0.0.1:50788/24e6d3d8-35d9-40fc-b176-02a0ce896f50
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.

> media@0.0.1 start:dev
> nest start --watch

Debugger listening on ws://127.0.0.1:50791/975a4e79-0762-4b02-aaea-02b1fa10f92a
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```

7.2 Khởi động các client

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
(myenv) lekhoa@MacBook-Pro-cua-Le client4 % python3 client.py
Login!
Peer hoạt động ở 127.0.0.1:65434
>> upload music.mp3
magnet:?xt=urn:btih:ff62b5b457f7df7efde99c8e9c8a86ab0383c869&dn=music.mp3&tr=h&tr=t&tr=t&tr=p&tr=%3A&tr=/&tr=/&tr=l&tr=o&tr=c&tr=a&tr=l&tr=h&tr=o&tr=s&tr=t&tr=%3A&tr=3&tr=0&tr=0&tr=0&tr=/&tr=v&tr=1&xl=5308416
>>
```

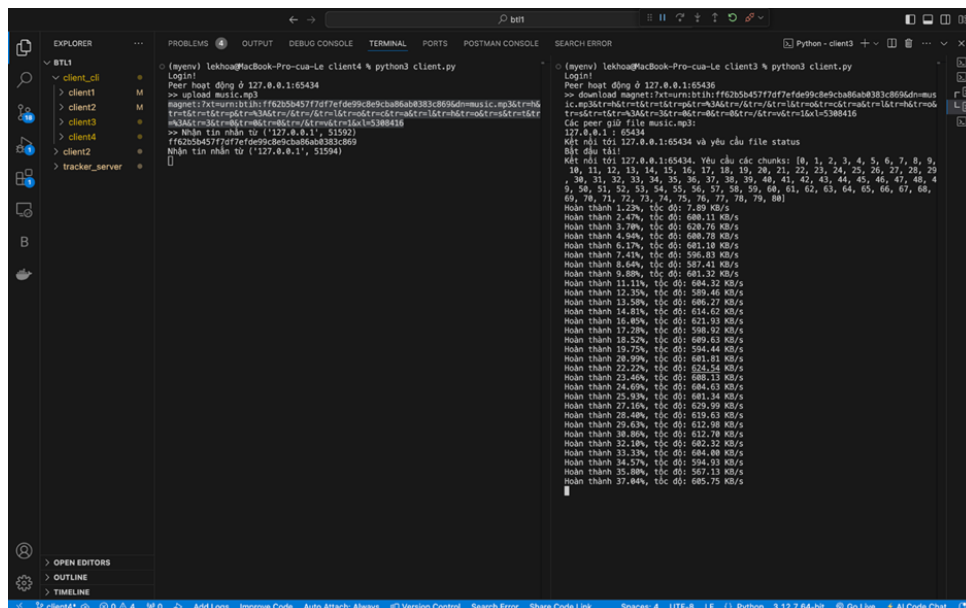


7.3 Tải file lần đầu cho một peer để đăng kí thông tin file cho tracker

```
(myenv) lekhoa@MacBook-Pro-cua-Le client4 % python3 client.py
Login!
Peer hoạt động ở 127.0.0.1:65434
>> upload music.mp3
magnet:?xt=urn:btih:ff62b5b457f7df7efde99c8e9c8a86ab0383c869&dn=music.mp3&tr=h&
tr=t&tr=t&tr=p&tr=%3A&tr=/&tr=/&tr=l&tr=o&tr=c&tr=a&tr=l&tr=h&tr=o&tr=s&tr=t&tr
=%3A&tr=3&tr=0&tr=0&tr=0&tr=/&tr=v&tr=1&xl=5308416
>> Nhận tin nhắn từ ('127.0.0.1', 51592)
ff62b5b457f7df7efde99c8e9c8a86ab0383c869
Nhận tin nhắn từ ('127.0.0.1', 51594)
[]
```

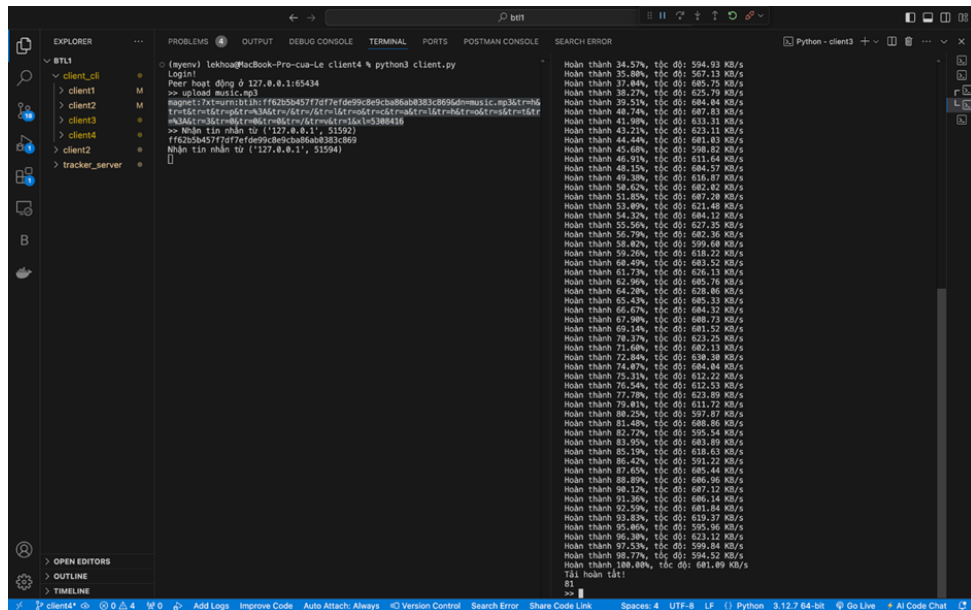
Sau khi upload, chương trình trả về một magnet link để người dùng khác có thể sử dụng để download.

7.4 Download file



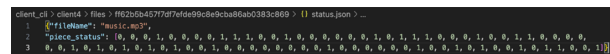
```
(myenv) lekhoa@MacBook-Pro-cua-Le client4 % python3 client.py
Login!
Peer hoạt động ở 127.0.0.1:65434
>> upload music.mp3
magnet:?xt=urn:btih:ff62b5b457f7df7efde99c8e9c8a86ab0383c869&dn=music.mp3&tr=h&
tr=t&tr=t&tr=p&tr=%3A&tr=/&tr=/&tr=l&tr=o&tr=c&tr=a&tr=l&tr=h&tr=o&tr=s&tr=t&tr
=%3A&tr=3&tr=0&tr=0&tr=0&tr=/&tr=v&tr=1&xl=5308416
>> Nhận tin nhắn từ ('127.0.0.1', 51592)
ff62b5b457f7df7efde99c8e9c8a86ab0383c869
Nhận tin nhắn từ ('127.0.0.1', 51594)
[]
```

```
(myenv) lekhoa@MacBook-Pro-cua-Le client3 % python3 client.py
Login!
Peer hoạt động ở 127.0.0.1:65436
>> download magnet:?xt=urn:btih:ff62b5b457f7df7efde99c8e9c8a86ab0383c869&dn=mu
ic.mp3&tr=h&tr=t&tr=t&tr=p&tr=%3A&tr=/&tr=/&tr=l&tr=o&tr=c&tr=a&tr=l&tr=h&tr=o&tr=s&tr=t&tr
=%3A&tr=3&tr=0&tr=0&tr=0&tr=/&tr=v&tr=1&xl=5308416
Các peer gửi file music.mp3
127.0.0.1 : 65434
Mật mã tại 127.0.0.1:65434. Yêu cầu các chunks [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 4
9, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80]
Hoàn thành 2.42%, tốc độ: 7.89 KB/s
Hoàn thành 3.78%, tốc độ: 628.76 KB/s
Hoàn thành 4.94%, tốc độ: 688.78 KB/s
Hoàn thành 6.17%, tốc độ: 681.18 KB/s
Hoàn thành 7.41%, tốc độ: 590.83 KB/s
Hoàn thành 8.64%, tốc độ: 587.41 KB/s
Hoàn thành 9.88%, tốc độ: 681.32 KB/s
Hoàn thành 11.11%, tốc độ: 684.32 KB/s
Hoàn thành 12.35%, tốc độ: 589.46 KB/s
Hoàn thành 13.59%, tốc độ: 686.27 KB/s
Hoàn thành 14.81%, tốc độ: 634.62 KB/s
Hoàn thành 16.05%, tốc độ: 621.93 KB/s
Hoàn thành 17.29%, tốc độ: 598.92 KB/s
Hoàn thành 18.52%, tốc độ: 689.63 KB/s
Hoàn thành 19.75%, tốc độ: 584.44 KB/s
Hoàn thành 20.99%, tốc độ: 681.81 KB/s
Hoàn thành 22.23%, tốc độ: 552.54 KB/s
Hoàn thành 23.46%, tốc độ: 688.13 KB/s
Hoàn thành 24.69%, tốc độ: 684.63 KB/s
Hoàn thành 25.93%, tốc độ: 681.14 KB/s
Hoàn thành 27.16%, tốc độ: 629.99 KB/s
Hoàn thành 28.40%, tốc độ: 619.63 KB/s
Hoàn thành 29.63%, tốc độ: 632.98 KB/s
Hoàn thành 30.86%, tốc độ: 632.78 KB/s
Hoàn thành 32.10%, tốc độ: 682.32 KB/s
Hoàn thành 33.33%, tốc độ: 684.88 KB/s
Hoàn thành 34.57%, tốc độ: 594.93 KB/s
Hoàn thành 35.80%, tốc độ: 567.13 KB/s
Hoàn thành 37.04%, tốc độ: 685.75 KB/s
```

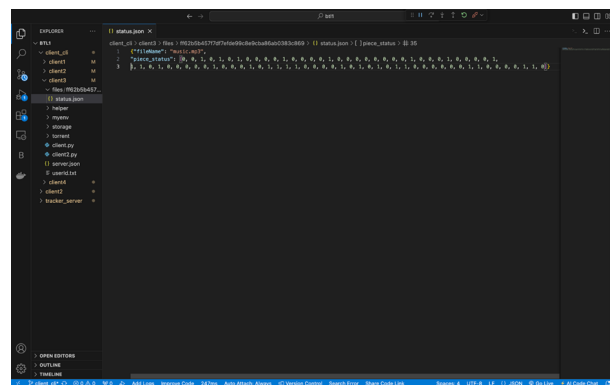



Nhập lệnh “download magnet_link” chương trình sẽ tiến hành download và hiển thị tuần tự quá trình tải xuống cho người dùng.

7.5 Tải file từ nhiều đồng thời từ nhiều peer

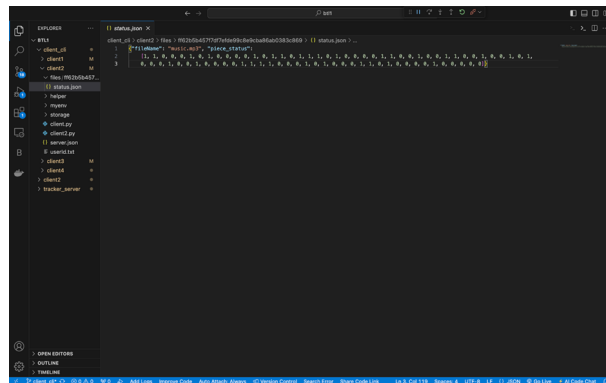


Hình 7.1: Trạng thái file piece của peer 1.

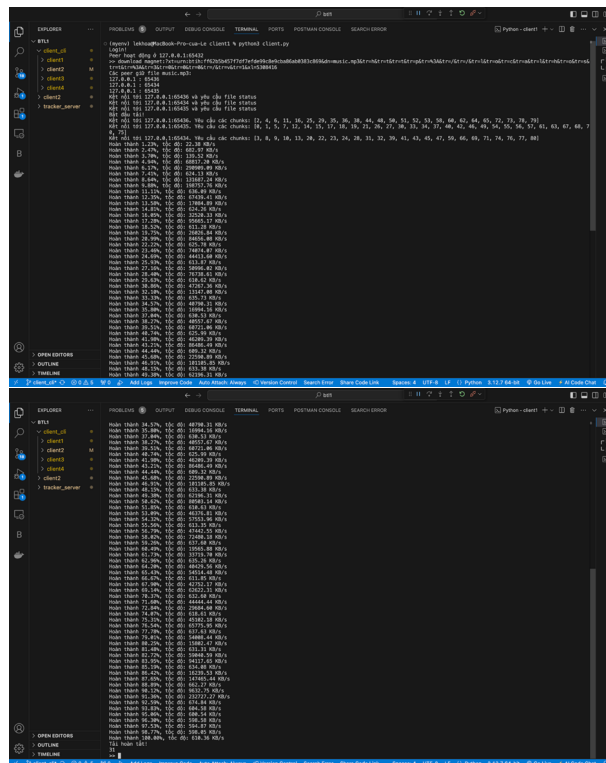


Hình 7.2: Trạng thái file piece của peer 2.

Peer 4 tiến hành download từ 3 peer trên



Hình 7.3: Trạng thái file piece của peer 3.



Người dùng sau khi download, kiểm tra thư mục storage/file_name để xem nội dung file.

Tài liệu

- [1] James Kurose, Keith Ross - Computer Networking A Top-Down Approach-Pearson (2021)
- [2] Computer Networks-Prentice Hall (2011)
- [3] P2P (Peer To Peer) File Sharing <https://www.geeksforgeeks.org/p2p-peer-to-peer-file-sharing/>
- [4] Peer-to-peer file sharing https://en.wikipedia.org/wiki/Peer-to-peer_file_sharing