HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA ĐÀO TẠO SAU ĐẠI HỌC

BÁO CÁO MÔN HỌC:
CÁC HỆ THỐNG PHÂN TÁN
HỆ THỐNG CHIA SẺ FILE P2P

Giảng viên hướng dẫn: **TS. Kim Ngọc Bách**

Học viên thực hiện:	**MHV:**
Lê Minh Giang	**B25CHHT016**
Trịnh Minh Tuấn	**B25CHHT060**
Trương Thanh Tùng	**B25CHHT065**

Hà Nội, 12/2025

Lời mở đầu

Trong bối cảnh công nghệ thông tin và truyền thông phát triển mạnh mẽ, nhu cầu trao đổi, chia sẻ và lưu trữ dữ liệu ngày càng gia tăng cả về quy mô lẫn tần suất. Các hệ thống mạng truyền thống dựa trên mô hình Client–Server tuy vẫn giữ vai trò quan trọng, song đã dần bộc lộ những hạn chế về khả năng mở rộng, chi phí vận hành và tính chịu lỗi khi số lượng người dùng tăng cao. Trước thực tiễn đó, mô hình mạng ngang hàng (Peer-to-Peer – P2P) đã ra đời như một hướng tiếp cận hiệu quả, cho phép phân tán tài nguyên và giảm sự phụ thuộc vào hạ tầng trung tâm.

Hệ thống chia sẻ tệp tin ngang hàng P2P không chỉ giúp tối ưu hóa băng thông và tận dụng tài nguyên của các nút tham gia, mà còn nâng cao tính sẵn sàng, khả năng mở rộng và độ tin cậy của toàn hệ thống. Nhiều ứng dụng thực tiễn như chia sẻ dữ liệu dung lượng lớn, phân phối nội dung, hay lưu trữ phân tán đều dựa trên nền tảng P2P và đã chứng minh hiệu quả vượt trội so với các mô hình tập trung truyền thống.

Xuất phát từ những cơ sở lý luận và nhu cầu thực tiễn nêu trên, nhóm thực hiện đề tài **"Xây dựng hệ thống chia sẻ file ngang hàng (P2P)"** với mục tiêu nghiên cứu nguyên lý hoạt động của mạng P2P, phân tích các mô hình kiến trúc tiêu biểu và triển khai một hệ thống chia sẻ tệp tin có khả năng hoạt động ổn định, hỗ trợ tải đa nguồn, đảm bảo tính toàn vẹn dữ liệu và có khả năng mở rộng trong thực tế.

Báo cáo này trình bày một cách có hệ thống các nội dung từ cơ sở lý thuyết, phân tích yêu cầu, thiết kế kiến trúc, cài đặt, thử nghiệm cho đến đánh giá kết quả và đề xuất hướng phát triển. Thông qua đề tài, nhóm mong muốn củng cố kiến thức về mạng máy tính, hệ thống phân tán và rèn luyện kỹ năng thiết kế, triển khai một ứng dụng mạng mang tính thực tiễn cao.

I. GIỚI THIỆU

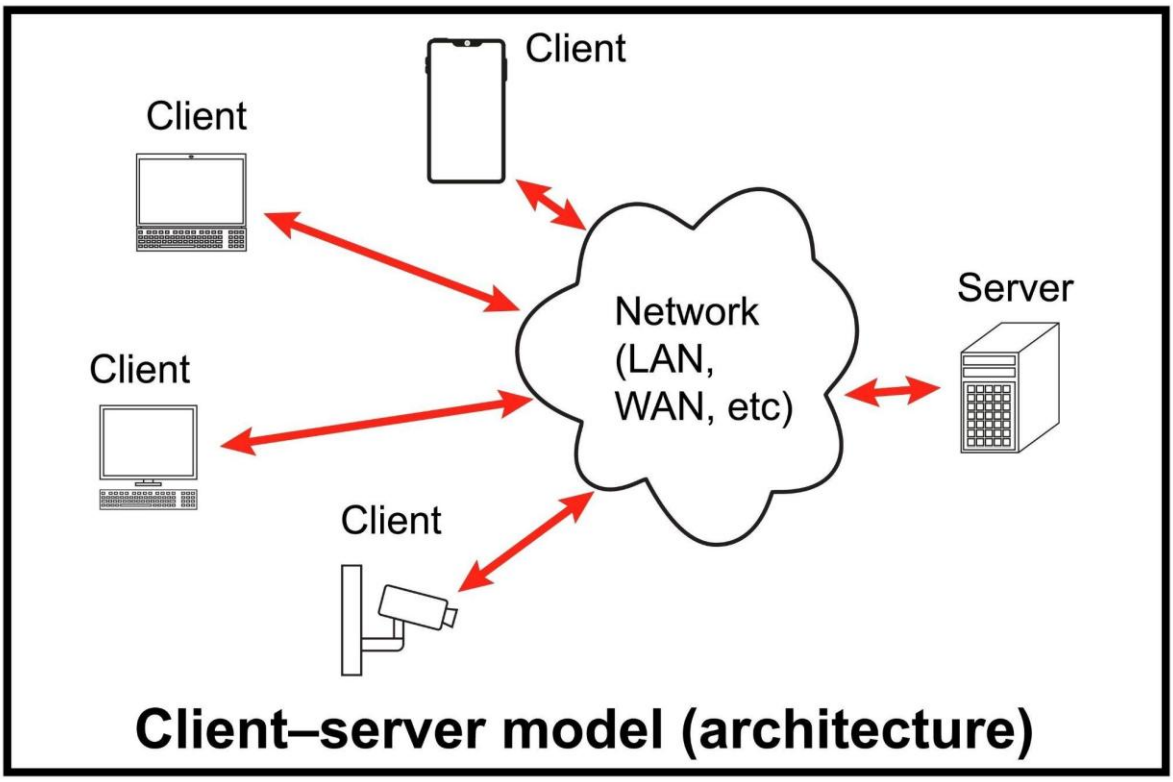
1. Lý do chọn đề tài

Trong kỷ nguyên chuyển đổi số, nhu cầu trao đổi và lưu trữ dữ liệu ngày càng tăng cao. Hiện nay, hai mô hình mạng phổ biến nhất là Client-Server (Khách-Chủ) và Peer-to-Peer (Mạng ngang hàng). Tuy nhiên, mỗi mô hình lại bộc lộ những ưu và nhược điểm riêng trong các kịch bản cụ thể:

1.1. Hạn chế của mô hình Client – Server truyền thống

Mô hình Client-Server dựa trên sự quản lý tập trung, nơi các máy khách (Client) gửi yêu cầu và máy chủ (Server) phản hồi dữ liệu. Dù phổ biến, mô hình này đối mặt với các vấn đề:

- Nếu máy chủ gặp sự cố hoặc bị tấn công (DDoS), toàn bộ hệ thống sẽ ngưng trệ
- Khi số lượng Client tăng đột biến, Server dễ rơi vào tình trạng quá tải, gây chậm trễ trong việc phản hồi.
- Việc duy trì một máy chủ trung tâm có cấu hình mạnh và băng thông lớn rất tốn kém

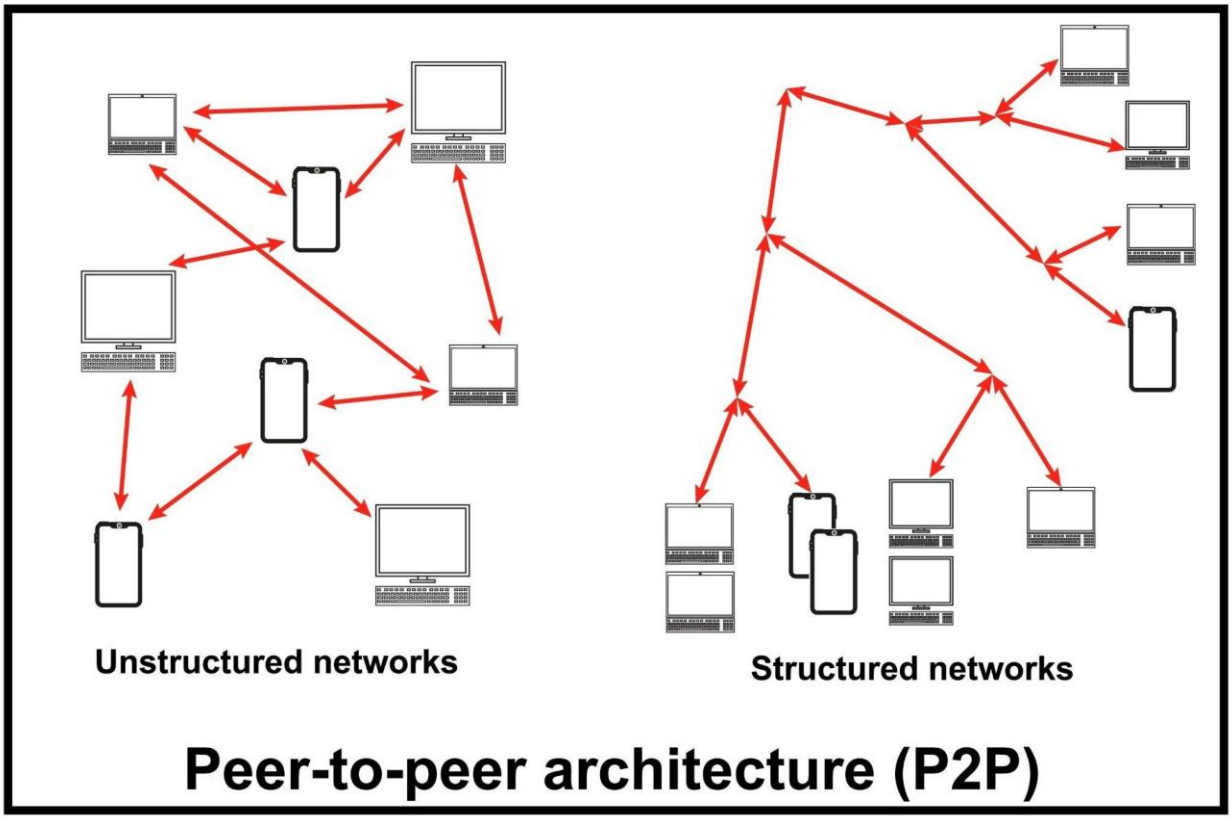


Hình 1.1. Mô hình Client – Server

1.2. Mô hình P2P

Ngược lại, mô hình P2P cho phép mỗi máy trạm tham gia vừa đóng vai trò là Client, vừa là Server. Sự khác biệt cốt lõi này mang lại:

- Dữ liệu được phân tán ở nhiều máy khác nhau, hệ thống vẫn hoạt động tốt ngay cả khi một số nút bị ngắt kết nối
- Càng nhiều người tham gia, tài nguyên (băng thông, dữ liệu) của hệ thống càng trở nên dồi dào
- Tận dụng tối đa tài nguyên phần cứng sẵn có từ phía người dùng cuối



Hình 1.2. Mô hình P2P

Xuất phát từ thực tiễn đó, việc nghiên cứu và xây dựng một Hệ thống chia sẻ file ngang hàng (P2P) là vô cùng thiết thực nhằm tối ưu hóa hiệu suất truyền tải dữ liệu và giảm bớt sự phụ thuộc vào hạ tầng trung tâm.

2. Mục tiêu đề tài

Mục tiêu chính của đồ án là nghiên cứu kiến trúc mạng phân tán và thực thi một ứng dụng chia sẻ file hiệu quả. Cụ thể bao gồm:

- Thiết lập cơ chế cho phép các máy trạm tìm kiếm và nhận diện nhau trong mạng lưới
- Thực hiện việc truyền tải dữ liệu (upload/download) trực tiếp giữa các nút mà không cần đi qua máy chủ trung gian
- Đảm bảo dữ liệu sau khi được ghép lại từ nhiều nguồn khác nhau vẫn giữ nguyên tính chính xác (sử dụng Hash hoặc Checksum)
- Cho phép tải xuống từ nhiều nguồn cùng lúc để đạt tốc độ tối đa

3. Phạm vi nghiên cứu

Báo cáo tập trung vào các nội dung sau:

- Hệ thống được thiết kế để hoạt động ổn định trong mạng cục bộ (LAN) và có khả năng mở rộng ra Internet thông qua các kỹ thuật chuyển tiếp cổng (Port Forwarding)
- Sử dụng Tracker (Tập trung hóa một phần): Một máy chủ đóng vai trò điều phối, lưu giữ danh sách các nút đang online
- Tập trung vào các định dạng file phổ biến (tài liệu, hình ảnh, video) với dung lượng từ trung bình đến lớn

II. PHÂN TÍCH YÊU CẦU

1. Yêu cầu chức năng

- **Tham gia/Rời mạng:**
  - Hệ thống cho phép các máy trạm đăng ký sự hiện diện khi gia nhập mạng lưới.
  - Duy trì danh sách các nút đang hoạt động để thiết lập kết nối ngang hàng.
- **Quản lý tệp tin cục bộ:**
  - Hệ thống có khả năng quản lý danh sách các tệp tin hiện có trong thư mục chia sẻ của người dùng
  - Thực hiện băm nội dung tệp tin để tạo định danh duy nhất cho mỗi file và sử dụng mã này để kiểm tra tính toàn vẹn của dữ liệu sau khi truyền tải
- **Tìm kiếm file (Search):**
  - Cho phép người dùng tìm kiếm tệp tin dựa trên tên hoặc mã định danh.
  - Hệ thống trả về danh sách chi tiết các máy trạm hiện đang nắm giữ tệp tin đó để người dùng lựa chọn tải về
- **Tải xuống (Download):**
  - Hỗ trợ tải file từ một hoặc nhiều Peer cùng lúc (Multi-source downloading).
  - Chia nhỏ file thành các mảnh (chunks/pieces) để tải song song.
- **Chia sẻ (Upload/Seeding):**
  - Hệ thống tự động lắng nghe và chấp nhận các yêu cầu kết nối từ Peer khác.

- Thực hiện gửi các mảnh file đã có sẵn cho các Peer đang yêu cầu trong mạng lưới.

- **Quản lý tiến trình:**
  - Hỗ trợ các chức năng Tạm dừng (Pause) và Tiếp tục (Resume) quá trình tải để người dùng linh hoạt quản lý tài nguyên mạng.

2. Yêu cầu phi chức năng

- Hệ thống phải đảm bảo tốc độ truyền tải nhanh, tối ưu hóa băng thông đường truyền giữa các máy trạm.
- Đảm bảo dữ liệu không bị lỗi hoặc sai lệch thông qua việc kiểm tra tính toàn vẹn (Data Integrity) sau khi ghép các mảnh file hoàn chỉnh.
- Kiến trúc hệ thống phải đảm bảo hoạt động ổn định và hiệu quả ngay cả khi số lượng Peer tham gia vào mạng lưới tăng lên

III. THIẾT KẾ PHẦN MỀM

1. Kiến trúc hệ thống

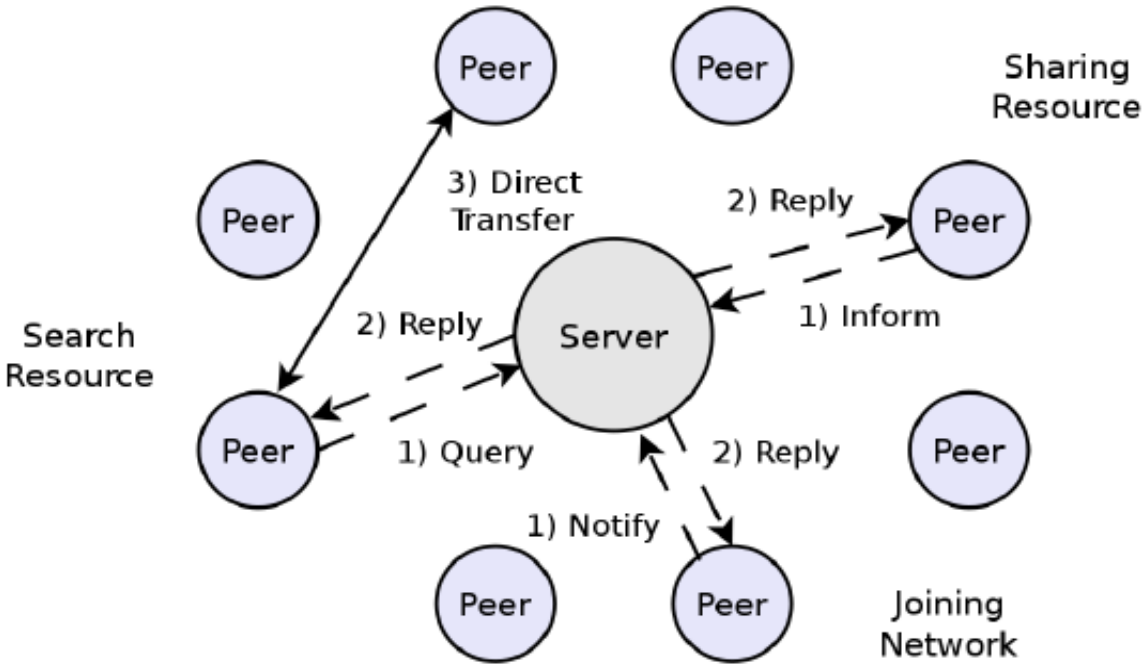
1.1. Mô hình Centralized P2P (Napster Style):

Có một Server trung tâm (Tracker) lưu trữ danh sách file và địa chỉ các Peer. Được gọi là mô hình thế hệ thứ nhất (như Napster), là sự kết hợp giữa kiến trúc Client-Server truyền thống và tính chất chia sẻ trực tiếp của P2P

Cách thức hoạt động:

Trong mô hình này, một Server trung tâm đóng vai trò là "người điều phối" hoặc "danh bạ điện thoại", trong khi việc truyền tải dữ liệu thực tế diễn ra trực tiếp giữa các máy con (Peers). Quy trình diễn ra:

- Khi một Peer tham gia vào mạng lưới, nó sẽ thông báo cho Server trung tâm địa chỉ IP của mình và danh sách các file mà nó sẵn sàng chia sẻ
- Khi Peer A muốn tìm một file (ví dụ: "song.mp3"), nó gửi yêu cầu truy vấn đến Server trung tâm
- Server kiểm tra cơ sở dữ liệu và gửi lại cho Peer A danh sách các Peer đang giữ file đó (ví dụ: Peer B, Peer C)
- Tải về (Download): Peer A thiết lập kết nối trực tiếp với Peer B để tải file. Server không tham gia vào quá trình truyền tải dữ liệu này



Hình 3.1.1. Mô hình Centralized P2P

Bảng 3.1. Mô tả các thành phần trong mô hình Centralized P2P.

Thành phần	Vai trò trong mô hình Centralized P2P
Server trung tâm	Lưu trữ chỉ mục (Index), địa chỉ IP, hỗ trợ tìm kiếm. Không lưu file.
Peer (Nút con)	Lưu trữ file thực tế và thực hiện việc gửi/nhận file.
Băng thông	Tiết kiệm cho Server vì dữ liệu lớn không đi qua nó.

1.2. Mô hình Decentralized P2P (Gnutella style)

Không có server, các Peer tự tìm kiếm nhau qua cơ chế Flooding hoặc DHT (Distributed Hash Table).

Cơ chế Flooding (Gnutella Style - Thế hệ 2)

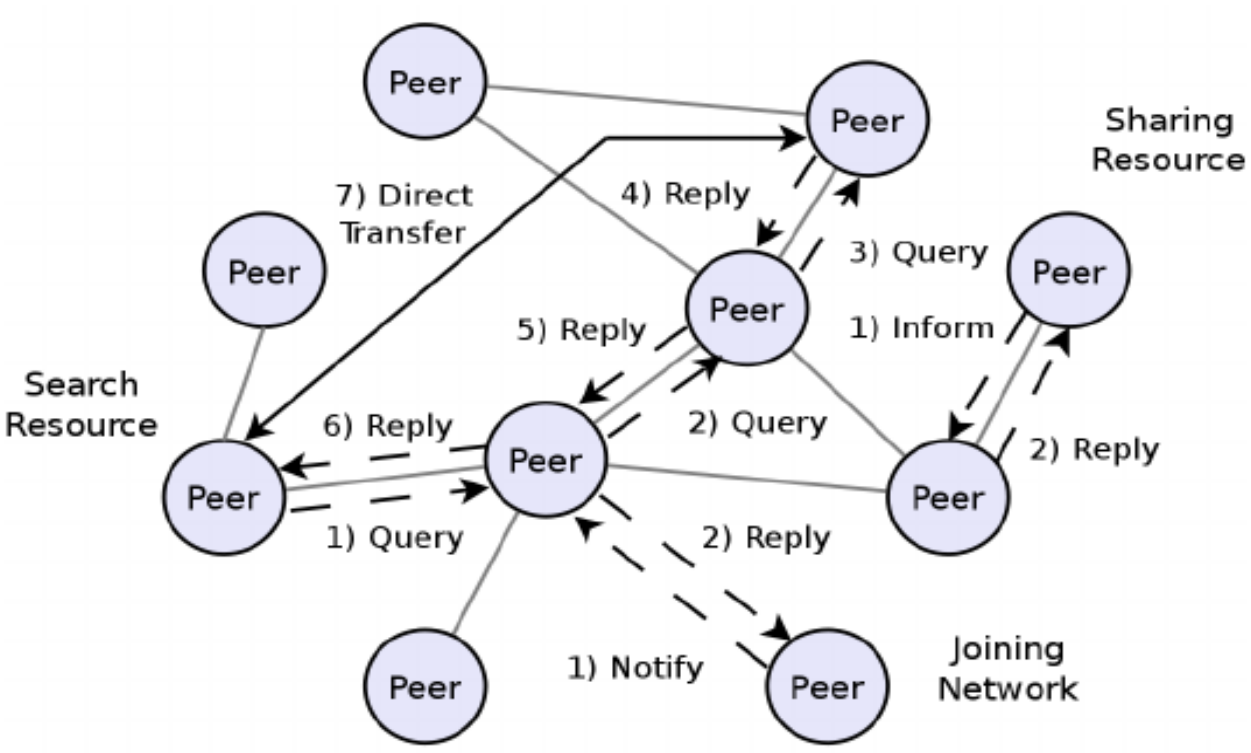
Đây là cách tiếp cận không cấu trúc (Unstructured). Khi bạn muốn tìm một file, bạn sẽ "hỏi" các láng giềng của mình, và họ lại tiếp tục hỏi láng giềng của họ.

- Cách thức hoạt động: 1. Truy vấn (Query): Peer A gửi yêu cầu tìm file đến tất cả các Peer đang kết nối trực tiếp với nó. 2. Lan tỏa (Flooding): Nếu các Peer láng giềng không có file, họ sẽ chuyển tiếp yêu cầu đó đến láng giềng của họ. Quá trình này lặp lại như một "cơn sóng" lan tỏa khắp mạng lưới. 3. TTL (Time-to-Live): Để tránh việc yêu cầu chạy vĩnh viễn, mỗi tin nhắn có một chỉ số TTL (thường là 7). Sau mỗi "bước nhảy" (hop), TTL giảm đi 1. Khi TTL = 0, tin nhắn sẽ bị hủy. 4. Phản hồi (Query Hit): Nếu một Peer có file, nó sẽ gửi phản hồi ngược lại theo đúng lộ trình mà truy vấn đã đi qua.
- Vấn đề "Bão tín hiệu" (Broadcast Storm): Nhược điểm lớn nhất của Flooding là nó tạo ra lượng lưu lượng mạng khổng lồ. Nếu có quá nhiều người cùng tìm kiếm, băng thông của toàn mạng sẽ bị nghẽn chỉ để chuyển tiếp các câu hỏi.

Cơ chế DHT (Distributed Hash Table - Thế hệ hiện đại)

Để khắc phục nhược điểm của Flooding, các hệ thống hiện đại (như BitTorrent không Tracker, IPFS) sử dụng cơ chế có cấu trúc (Structured) dựa trên bảng băm phân tán.

- **Cách thức hoạt động:**
    - Mỗi file và mỗi Peer được gán cho một ID duy nhất (thường là một dãy số 160-bit hoặc 256-bit) bằng cách sử dụng các hàm băm như SHA-1.
    - Thay vì hỏi loạn xạ, hệ thống quy định: Peer có ID "gần" với ID của file nhất sẽ chịu trách nhiệm lưu thông tin về vị trí của file đó.
    - Khi cần tìm file, bạn chỉ cần thực hiện một lộ trình có tính toán (thường chỉ mất O(log N) bước) để đến đúng Peer đang giữ thông tin, thay vì hỏi tất cả mọi người.



Hình 3.1.2. Mô hình Decentralized P2P

1.3. Mô hình Hybrid P2P (BitTorrent Style)

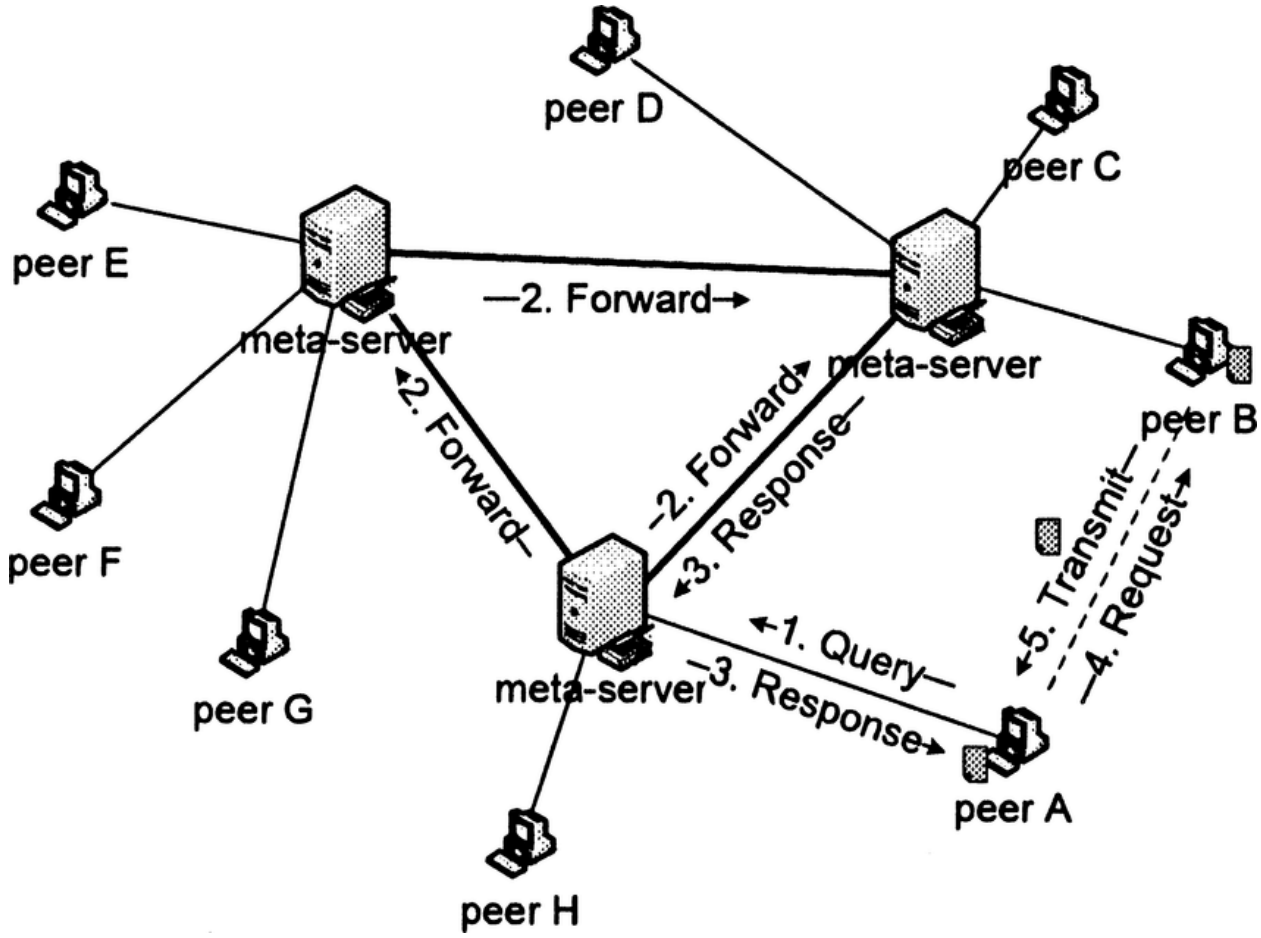
Kết hợp Tracker để điều phối nhưng dữ liệu truyền đi hoàn toàn giữa các Peer**.** Sự kết hợp thông minh giữa tính hiệu quả của Server trung tâm (để quản lý danh bạ) và sức mạnh phân tán của mạng P2P (để truyền tải dữ liệu). Trong mô hình này, **Tracker** đóng vai trò là "người môi giới", giúp các máy con tìm thấy nhau mà không trực tiếp tham gia vào việc gửi hay nhận file.

Quy trình:

- Người chia sẻ tạo ra một file nhỏ chứa thông tin mô tả về file gốc (kích thước, số lượng mảnh, mã băm SHA-1) và địa chỉ của Tracker
- Khi Peer A muốn tải file, nó mở file .torrent bằng một phần mềm (Client). Phần mềm này gửi địa chỉ IP của Peer A tới Tracker
- Tracker phản hồi cho Peer A một danh sách các địa chỉ IP của những người khác cũng đang chia sẻ hoặc tải file đó (gọi là một Swarm - Bầy đàn)
- File gốc được chia thành hàng ngàn mảnh nhỏ. Peer A sẽ kết nối đồng thời với nhiều Peer khác để tải các mảnh khác nhau

Các khái niệm cốt lõi trong BitTorrent:

- **Seeder**: Những người đã tải xong 100% file và vẫn đang tiếp tục giữ kết nối để "re" (tải lên) cho người khác
- **Leecher**: Những người đang trong quá trình tải file (họ vừa tải về từ người khác, vừa tải lên những mảnh mình đã có)
- **Choking (Nghẽn):** Một cơ chế giúp tối ưu băng thông. Peer sẽ ưu tiên gửi dữ liệu cho những ai đang gửi lại dữ liệu cho mình nhanh nhất
- **Tit-for-Tat (Có đi có lại):** Thuật toán khuyến khích người dùng chia sẻ. Nếu bạn không cho đi (upload), các Peer khác sẽ hạn chế tốc độ tải



Hình 3.1.3. Mô hình Hybrid P2P

1.4. Lựa chọn mô hình

Ta có bảng so sánh

Bảng 3.1.2: So sánh giữa các mô hình

Tiêu chí	Centralized (Napster)	Decentralized (Gnutella/DHT)	Hybrid (BitTorrent)
Tốc độ tìm kiếm	Rất nhanh (Tra cứu tại Server)	Chậm (Flooding) hoặc Trung bình (DHT)	Nhanh (Qua Tracker)
Độ tin cậy	Thấp (Dễ sập toàn hệ thống)	Rất cao (Không có điểm yếu chí tử)	Cao (Tracker có thể thay thế)
Khả năng mở rộng	Kém (Server bị quá tải)	Rất tốt (Càng nhiều người càng mạnh)	Cực tốt (Tối ưu băng thông)
Độ phức tạp cài đặt	Thấp	Rất cao (Đặc biệt là DHT)	Trung bình
Ứng dụng phổ biến	Mạng nội bộ doanh nghiệp	Blockchain, IPFS	Chia sẻ file dung lượng lớn

Trường hợp 1: Chọn mô hình Hybrid (Lựa chọn tối ưu cho đại đa số)

Nếu muốn xây dựng một hệ thống chia sẻ file công cộng hoặc cần xử lý các file dung lượng lớn (Phim, bộ cài phần mềm, ISO), Hybrid P2P (BitTorrent style) là sự lựa chọn tốt nhất vì:

- Hiệu suất vượt trội: Tận dụng tối đa băng thông của tất cả người dùng
- Chi phí thấp: Bạn chỉ cần một Server cấu hình nhẹ làm Tracker để điều phối, không cần Server lưu trữ dữ liệu hàng Petabyte

- Tính linh hoạt: Có thể nâng cấp lên cơ chế không Tracker (Magnet link) để tăng tính bền vững

Trường hợp 2: Chọn Mô hình Decentralized (Lựa chọn cho bảo mật và sự bền vững)

Nếu ưu tiên hàng đầu của bạn là không thể bị đánh sập (Censorship-resistant) hoặc xây dựng hệ thống lưu trữ phân tán lâu dài:

- DHT (Distributed Hash Table) là hướng đi đúng đắn. Nó loại bỏ hoàn toàn sự phụ thuộc vào bất kỳ máy chủ nào. Đây là nền tảng cho các công nghệ hiện đại như Web 3.0, IPFS hoặc các hệ thống lưu trữ dữ liệu trong Blockchain

Trường hợp 3: Chọn Mô hình Centralized (Lựa chọn cho mạng nội bộ)

Dù đã lỗi thời ở quy mô Internet, nhưng mô hình tập trung vẫn hữu ích trong:

- Môi trường doanh nghiệp (Intranet): Nơi bạn có toàn quyền kiểm soát Server, yêu cầu tìm kiếm file cực nhanh và dữ liệu cần được quản lý tập trung để đảm bảo bảo mật nội bộ

⇨ Kết luận: chọn mô hình Hybrid P2P (BitTorrent style) để có một giải pháp thực tế, cân bằng và mạnh mẽ nhất

2. Thiết kế các module chức năng

2.1. Giao tiếp mạng

Đây là module nền tảng, chịu trách nhiệm thiết lập "đường truyền" giữa các nút trong mạng. Chức năng chính:

- **Lắng nghe kết nối (Listening):** Khởi tạo một ServerSocket tại một cổng (Port) xác định để chờ các yêu cầu kết nối từ Peer khác
- **Khởi tạo kết nối (Connecting):** Chủ động tạo Socket để kết nối tới địa chỉ IP và Port của Peer đích
- **Quản lý luồng (Stream Management):** Quản lý InputStream và OutputStream để truyền tải dữ liệu thô (bytes)
- **Xử lý đa luồng (Multi-threading):** Mỗi kết nối tới một Peer mới sẽ được quản lý bởi một Thread riêng biệt để đảm bảo hệ thống không bị treo khi truyền file

2.1.1. Tổng quan về giao thức sử dụng – UDP

Mặc dù thông thường việc truyền file hay dùng TCP để đảm bảo tin cậy, nhưng việc sử dụng UDP (User Datagram Protocol) mang lại những đặc điểm riêng biệt cho hệ thống:

- Giảm thiểu độ trễ do không cần quá trình bắt tay (3-way handshake) phức tạp

- Hướng gói (Packet-oriented), dữ liệu được chia nhỏ thành các DatagramPacket

2.1.2. Lắng nghe kết nối (Listening)

Đây là trạng thái thụ động của một Peer, cho phép nó sẵn sàng nhận yêu cầu từ các Peer khác trong mạng**.** Hệ thống khởi tạo một DatagramSocket gắn (bind) với một số hiệu cổng (Port) cố định trên máy cục bộ**.** Quy trình xử lý:

- Liên tục chạy một vòng lặp ở chế độ nền để chờ gói tin đến
- Sử dụng phương thức receive() để chặn (block) cho đến khi có một gói tin DatagramPacket được gửi tới
- Phân tích gói tin để lấy địa chỉ IP và Port của bên gửi nhằm phục vụ việc phản hồi

2.1.3. Khởi tạo kết nối (Connecting)

Khác với TCP tạo kết nối vật lý duy nhất, "kết nối" trong UDP ở đây mang tính logic, nghĩa là thiết lập mục tiêu gửi dữ liệu

- Peer chủ động tạo một gói tin chứa dữ liệu yêu cầu (Request) hoặc dữ liệu file
- Bao gồm địa chỉ IP đích (InetAddress) và số hiệu Port của Peer nhận
- Cho phép gửi dữ liệu đến nhiều Peer khác nhau từ cùng một Socket mà không cần thiết lập lại kết nối phức tạp

2.1.4. Quản lý luồng thông tin dữ liệu (Stream/Packet Management)

Vì UDP hoạt động dựa trên các gói tin (Datagram) thay vì luồng (Stream) liên tục như TCP, chức năng này sẽ tập trung vào việc đóng gói và giải nén dữ liệu:

- **Phân mảnh dữ liệu (Fragmentation):** Nếu dữ liệu hoặc file lớn hơn kích thước tối đa của một gói tin UDP (thường là 64KB, nhưng thực tế nên để khoảng 1024 - 8192 bytes để tránh phân mảnh IP), module sẽ chia nhỏ dữ liệu thành các đoạn
- **Đệm dữ liệu (Buffering):** Sử dụng các mảng byte (byte[] buffer) để lưu trữ tạm thời dữ liệu thô trước khi chuyển đổi chúng thành các đối tượng có ý nghĩa (như tin nhắn, lệnh điều khiển hoặc khối dữ liệu file)

Xác nhận (ACK - Optional): Thiết kế các gói tin phản hồi nhỏ để xác nhận bên kia đã nhận được gói tin, nhằm khắc phục nhược điểm mất gói của UDP

2.1.5. Xử lý đa luồng (Multi-threading)

Để đảm bảo hiệu năng cao và tính sẵn sàng, Module Network áp dụng mô hình đa luồng:

- Một luồng riêng biệt chỉ làm nhiệm vụ listen và nhận gói tin. Khi nhận được dữ liệu, nó sẽ đẩy dữ liệu vào một hàng đợi xử lý hoặc chuyển giao cho luồng khác, tránh làm tắc nghẽn cổng nhận
- Khi cần gửi một file lớn, hệ thống sẽ khởi tạo một luồng gửi riêng. Điều này giúp người dùng vẫn có thể thao tác trên giao diện (UI) hoặc tiếp tục nhận tin nhắn từ Peer khác mà không bị treo hệ thống
- Sử dụng thread pool để quản lý tài nguyên, tránh việc tạo quá nhiều luồng gây quá tải CPU khi có nhiều Peer kết nối cùng lúc

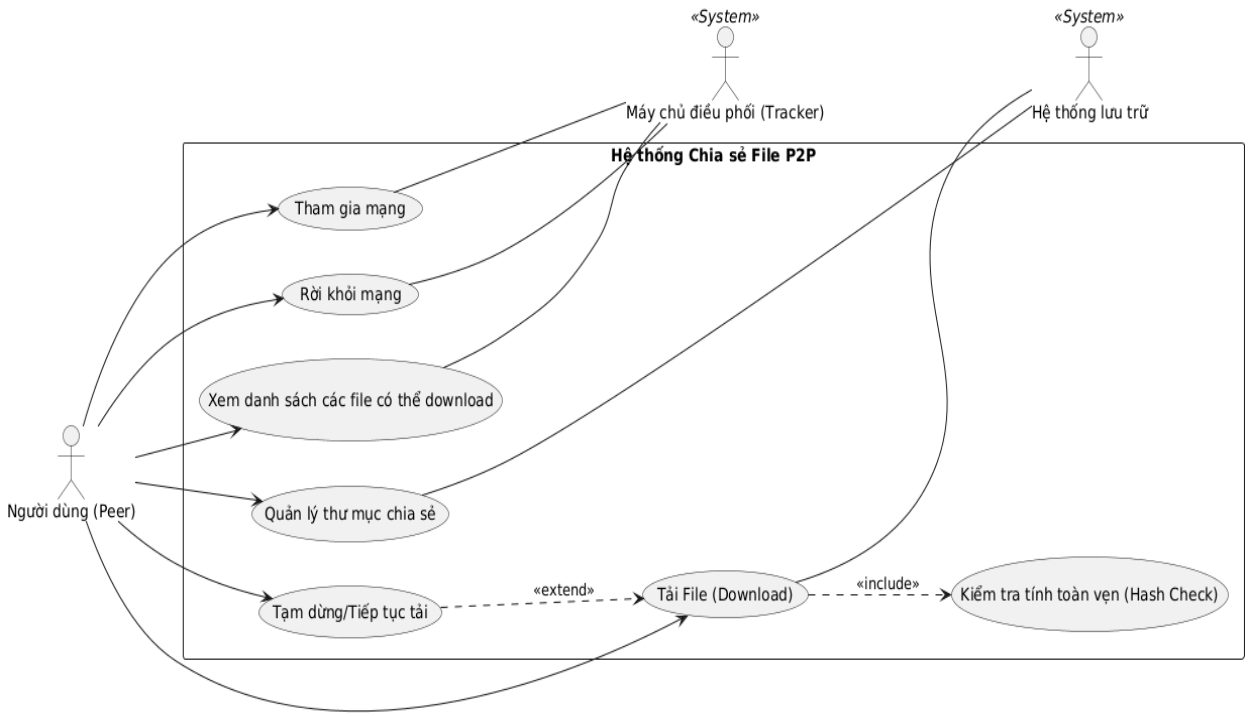
2.2. Giao tiếp giữa frontend và backend

Lựa chọn **HTTP (Hypertext Transfer Protocol)** làm giao thức giao tiếp chính giữa Frontend (FE) và Backend (BE). Đây là tiêu chuẩn cho hầu hết các ứng dụng hiện nay. Việc lựa chọn HTTP (Hypertext Transfer Protocol) làm phương thức giao tiếp chính giữa Frontend và Backend được xác định là giải pháp tối ưu, tuân theo tiêu chuẩn công nghiệp phổ biến hiện nay.

Vận hành dựa trên mô hình Client-Server, HTTP đảm bảo quy trình trao đổi dữ liệu minh bạch thông qua cơ chế Request-Response. Đặc điểm ưu việt của giao thức này nằm ở tính phi trạng thái (stateless), trong đó mỗi yêu cầu từ Frontend đều độc lập và chứa đầy đủ thông tin cần thiết, giúp Server xử lý mà không cần lưu trữ lịch sử các yêu cầu trước đó.

Chính nhờ đặc tính này, khả năng mở rộng hệ thống (Scaling) trở nên cực kỳ linh hoạt; ta có thể dễ dàng triển khai nhiều server Backend phía sau bộ cân bằng tải (Load Balancer) mà không gặp trở ngại về đồng bộ dữ liệu phiên làm việc. Kết hợp với cơ chế Caching tích hợp sẵn mạnh mẽ, HTTP khẳng định vai trò là lựa chọn an toàn và hiệu quả nhất nhờ sự đơn giản, tính tương thích cao và khả năng đáp ứng tốt sự phát triển của hệ thống

2.3. Sơ đồ Usecase hệ thống



Hình 3.2.1. Sơ đồ Usecase hệ thống

Mô tả chi tiết sơ đồ Usecase

Sơ đồ Use Case thể hiện các tương tác giữa người dùng và hệ thống, giúp xác định ranh giới và các chức năng chính của phần mềm chia sẻ file ngang hàng

2.3.1. Các tác nhân (Actors)

- **Người dùng (Peer):** Là tác nhân chính, đại diện cho một nút (node) trong mạng. Peer thực hiện các hành động như tìm kiếm, tải tệp và quản lý tài nguyên chia sẻ của chính mình
- **Máy chủ điều phối (Tracker):** Là tác nhân hệ thống đóng vai trò trung gian trong mô hình Hybrid P2P. Tracker quản lý danh sách các Peer đang hoạt động và vị trí của các tệp tin trong mạng nhưng không tham gia trực tiếp vào quá trình truyền dữ liệu
- **Hệ thống lưu trữ (Storage):** Đại diện cho hệ thống tệp tin cục bộ (Local File System) trên máy của người dùng, nơi dữ liệu được đọc ra để chia sẻ hoặc ghi vào khi tải về

Nhóm chức năng	Tên Use Case	Mô tả chi tiết
Quản lý mạng	Tham gia/Rời mạng	Peer thông báo trạng thái hoạt động cho Tracker để được cập nhật vào danh sách các nút hiện diện (Active Peers)
Tìm kiếm	Xem danh sách file	Peer gửi yêu cầu đến Tracker để lấy thông tin về các tệp tin hiện có sẵn trong mạng lưới
Quản lý dữ liệu	Quản lý thư mục chia sẻ	Người dùng chọn các tệp tin trên máy cục bộ để đăng ký chia sẻ với mạng lưới. Hệ thống sẽ lập chỉ mục (index) các tệp này
Truyền tải	Tải File (Download)	Đây là chức năng cốt lõi. Hệ thống thiết lập kết nối Socket tới các Peer khác để truyền dữ liệu theo từng mảnh (chunks)
Hỗ trợ	Tạm dừng/Tiếp tục	Cho phép người dùng kiểm soát quá trình tải về dựa trên tình trạng băng thông hoặc nhu cầu cá nhân

Đảm bảo	Kiểm tra tính toàn vẹn	Sau khi tải đủ các mảnh, hệ thống tự động tính toán mã Hash để đảm bảo file không bị lỗi hoặc bị sửa đổi so với gốc

2.3.2. Các mối quan hệ đặc biệt

Mối quan hệ <<include>> (Bao hàm):

- Use Case Tải File bao gồm Kiểm tra tính toàn vẹn. Điều này có nghĩa là quy trình kiểm tra Hash là một bước bắt buộc và tự động sau khi quá trình download hoàn tất để đảm bảo dữ liệu hợp lệ

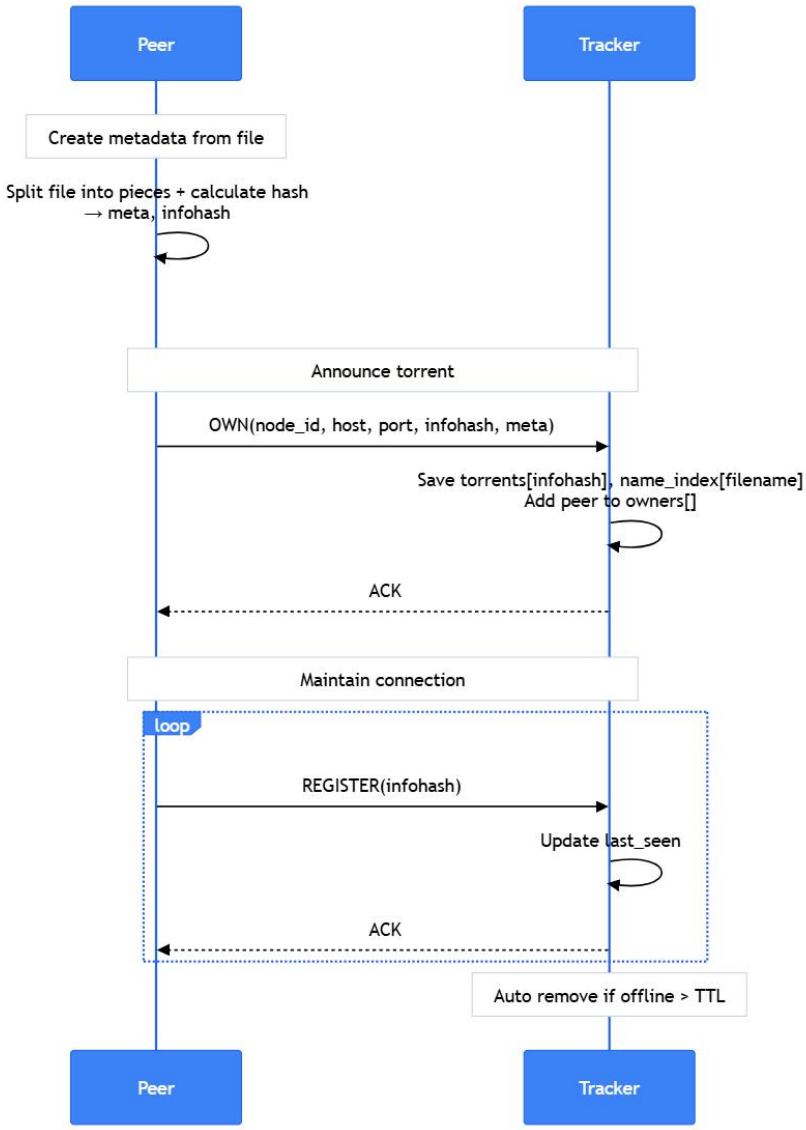
Mối quan hệ <<extend>> (Mở rộng):

- Use Case Tạm dừng/Tiếp tục mở rộng cho Tải File. Đây là chức năng tùy chọn, người dùng có thể kích hoạt hoặc không trong suốt vòng đời của một tiến trình tải file

2.4. Thiết kế luồng dữ liệu

2.4.1. Luồng đang ký và chia sẻ tệp (Upload / Seed)

Luồng này mô tả cách một node tham gia chia sẻ dữ liệu vào mạng P2P



Bước 1. Phân mảnh và xây dựng metadata của tệp

- Node đọc tệp cần chia sẻ và tiến hành phân mảnh tệp thành các đơn vị dữ liệu (piece) có kích thước cố định. Với mỗi piece, hệ thống tính toán hàm băm SHA-256 để tạo ra danh sách các giá trị băm đại diện cho toàn bộ nội dung tệp.
- Các thông tin này được đóng gói thành metadata gồm: tên tệp, kích thước tệp, kích thước piece và danh sách giá trị băm của từng piece. Metadata tiếp tục được băm để sinh ra một định danh duy nhất gọi là *infohash*.

Bước 2. Đăng ký torrent với máy chủ điều phối (Tracker)

- Node gửi thông điệp đăng ký chứa infohash, metadata và thông tin định danh node đến tracker. Tracker tiếp nhận, lưu metadata theo khóa infohash và cập nhật bảng ánh xạ từ tên tệp sang infohash. Đồng thời, node được thêm vào danh sách các node đang sở hữu torrent (owners).

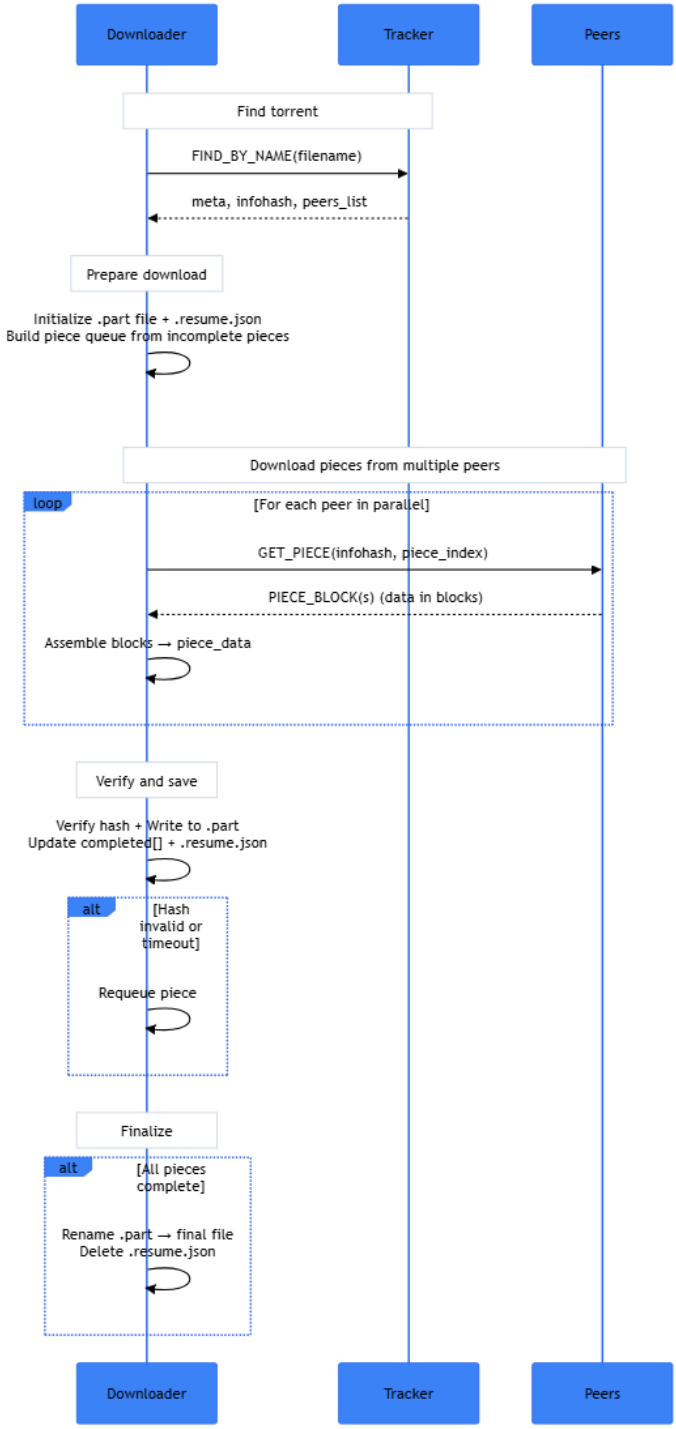
Bước 3. Duy trì trạng thái trực tuyến của node

- Sau khi đăng ký thành công, node định kỳ gửi các thông điệp duy trì trạng thái đến tracker. Tracker cập nhật dấu thời gian hoạt động cuối cùng của node nhằm đảm bảo node vẫn đang khả dụng để cung cấp dữ liệu.

Bước 4. Loại bỏ node không còn khả dụng

- Tracker định kỳ thực hiện tiến trình kiểm tra trạng thái các node. Những node không gửi thông điệp duy trì trong khoảng thời gian vượt quá TTL sẽ bị loại khỏi danh sách owners. Nếu một torrent không còn node nào giữ dữ liệu, torrent đó sẽ bị loại khỏi hệ thống.

2.4.2. Luồng Tải xuống dữ liệu phân tán (Download)



Luồng này mô tả quá trình một node tải dữ liệu từ nhiều node ngang hàng trong mạng P2P, đảm bảo tính toàn vẹn dữ liệu và hỗ trợ khôi phục khi bị gián đoạn.

Bước 1. Tra cứu torrent tại tracker

- Node tải xuống gửi yêu cầu truy vấn torrent theo tên tệp đến tracker. Tracker phản hồi thông tin metadata của torrent bao gồm định danh infohash và danh sách các node hiện đang sở hữu dữ liệu (peers_list). Thông tin này xác định các nguồn dữ liệu hợp lệ cho quá trình tải.

Bước 2. Khởi tạo phiên tải

- Node tiến hành chuẩn bị môi trường tải bằng cách tạo tệp dữ liệu tạm thời (.part) để lưu nội dung đang tải và tạo cấu trúc lưu trữ trạng thái tải (.resume.json). Trạng thái này ghi nhận các piece đã hoàn thành và chưa hoàn thành, làm cơ sở cho cơ chế tiếp tục tải.

Bước 3. Xây dựng hàng đợi các piece chưa hoàn thành

- Hệ thống tạo hàng đợi xử lý chỉ bao gồm các piece chưa được tải đầy đủ (completed[i] = 0). Điều này giúp tránh việc tải lại các dữ liệu đã có và tối ưu hóa băng thông.

Bước 4. Thiết lập các tiến trình tải song song

- Node khởi tạo nhiều tiến trình tải song song, mỗi tiến trình tương tác với một node nguồn trong danh sách peers_list. Các tiến trình này thực hiện tải dữ liệu đồng thời để tăng hiệu suất truyền dữ liệu.

Bước 5. Truyền dữ liệu theo khối

- Mỗi tiến trình gửi yêu cầu GET_PIECE kèm theo infohash và chỉ số piece đến node nguồn. Node nguồn phản hồi bằng cách truyền từng khối dữ liệu nhỏ (block) của piece tương ứng qua giao thức UDP. Node tải xuống tiếp nhận và lưu trữ tạm thời các block này trong bộ đệm.

Bước 6. Lắp ráp và kiểm tra toàn vẹn dữ liệu

- Sau khi nhận đủ các block của một piece, node tải xuống tiến hành lắp ráp các block thành piece hoàn chỉnh và kiểm tra toàn vẹn dữ liệu bằng cách so sánh giá trị băm của piece với metadata đã lưu.

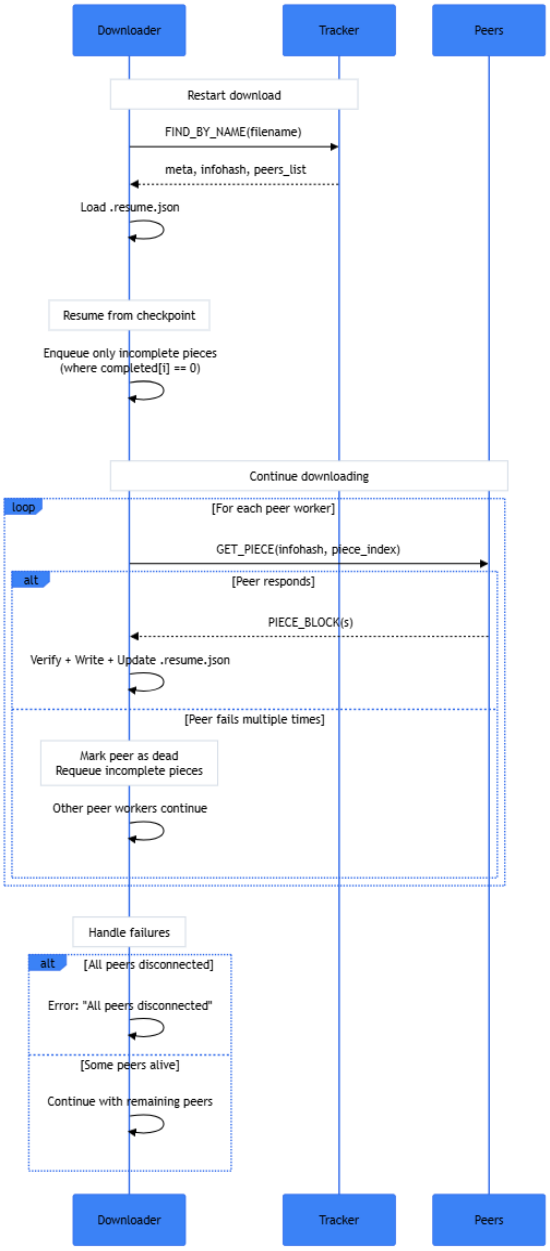
Bước 7. Ghi dữ liệu và cập nhật trạng thái tải

- Nếu piece hợp lệ, node ghi piece vào đúng vị trí trong tệp tạm (.part) và cập nhật trạng thái tải trong tệp .resume.json. Nếu piece không hợp lệ hoặc xảy ra lỗi trong quá trình truyền, piece sẽ được đưa lại vào hàng đợi để tải lại.

Bước 8. Hoàn tất tải xuống

- Khi toàn bộ các piece đã được tải thành công, hệ thống đổi tên tệp tạm thành tệp hoàn chỉnh và xóa dữ liệu resume, kết thúc phiên tải.

2.4.3. Luồng Tiếp tục tải khi gián đoạn (Resume Download)



Luồng này mô tả cơ chế cho phép một node tiếp tục quá trình tải dữ liệu đã bị gián đoạn, đảm bảo không phải tải lại các phần dữ liệu đã hoàn thành và có khả năng thích nghi với sự thay đổi trạng thái của các node nguồn trong mạng P2P.

Bước 1. Khởi động lại phiên tải và tra cứu torrent

- Khi node tải xuống khởi động lại phiên tải, hệ thống gửi yêu cầu truy vấn torrent theo tên tệp đến tracker. Tracker phản hồi với metadata của torrent, infohash và danh sách các node nguồn hiện đang khả dụng.

Bước 2. Khôi phục trạng thái tải từ tệp resume

- Node tiến hành đọc tệp resume.json để khôi phục trạng thái tải trước đó. Tệp này chứa thông tin về các piece đã được tải hoàn tất và các piece còn thiếu tại thời điểm gián đoạn.

Bước 3. Tái lập hàng đợi tải từ checkpoint

- Chỉ các piece chưa hoàn thành (completed[i] = 0) mới được đưa vào hàng đợi xử lý. Các piece đã hoàn tất được loại khỏi hàng đợi, giúp tiết kiệm băng thông và thời gian tải.

Bước 4. Tiếp tục tải song song từ nhiều node nguồn

- Node tiếp tục khởi tạo các tiến trình tải song song tương ứng với từng node nguồn trong danh sách peers_list. Mỗi tiến trình gửi yêu cầu GET_PIECE kèm theo infohash và chỉ số piece đến node nguồn.

Bước 5. Kiểm tra, ghi dữ liệu và cập nhật checkpoint

- Khi nhận được các khối dữ liệu của piece, node tiến hành lắp ráp, kiểm tra toàn vẹn dữ liệu và ghi vào tệp tạm (.part). Sau mỗi piece hoàn tất, hệ thống cập nhật trạng thái tải và ghi lại tệp resume.json nhằm tạo checkpoint mới.

Bước 6. Xử lý sự cố trong quá trình tiếp tục tải

- Nếu một node nguồn không phản hồi hoặc thất bại nhiều lần liên tiếp, node tải xuống sẽ đánh dấu node đó là không khả dụng và đưa các piece đang tải dở trở lại hàng đợi. Các tiến trình còn lại tiếp tục tải dữ liệu từ các node nguồn khác còn khả dụng.

Bước 7. Xử lý trường hợp mất toàn bộ node nguồn

- Trong trường hợp tất cả các node nguồn đều mất kết nối, hệ thống dừng phiên tải và sinh thông báo lỗi. Nếu vẫn còn ít nhất một node nguồn hoạt động, quá trình tiếp tục tải được duy trì cho đến khi hoàn tất.

Bước 8. Hoàn tất phiên resume

- Khi toàn bộ các piece còn thiếu đã được tải thành công, hệ thống hợp nhất dữ liệu thành tệp hoàn chỉnh và xóa dữ liệu resume, kết thúc hoàn toàn phiên tải.

2.5. Thuật toán

2.5.1. Cơ chế phân mảnh dữ liệu (Piece-based Data Partitioning)

Hệ thống áp dụng cơ chế phân mảnh dữ liệu theo đơn vị piece nhằm đảm bảo khả năng truyền dữ liệu phân tán, song song và có khả năng phục hồi khi xảy ra lỗi. Mỗi tệp trước khi được chia sẻ được đọc tuần tự và chia thành các khối dữ liệu có kích thước cố định là 256 KB. Việc chuẩn hóa kích thước piece cho toàn bộ hệ thống cho phép các node có thể yêu cầu và truyền các phần dữ liệu độc lập với nhau, từ đó hình thành nền tảng cho cơ chế tải đa nguồn.

Đối với mỗi piece, hệ thống tính toán một giá trị băm mật mã SHA-256 nhằm tạo ra danh sách các hàm băm đại diện cho toàn bộ nội dung tệp. Danh sách này được lưu trữ trong metadata của torrent và được sử dụng như một tham chiếu chuẩn để xác minh tính toàn vẹn dữ liệu trong suốt quá trình truyền. Trong quá trình tải, mỗi piece nhận được từ mạng sẽ được kiểm tra bằng cách so sánh giá trị băm của piece với giá trị tương ứng trong metadata. Nếu hai giá trị không trùng khớp, piece sẽ bị loại bỏ và được yêu cầu tải lại, nhờ đó đảm bảo rằng dữ liệu hoàn chỉnh cuối cùng luôn chính xác tuyệt đối so với dữ liệu gốc.

Ví dụ: Giả sử một tệp có kích thước 50 MB được đưa vào hệ thống để chia sẻ. Với kích thước piece cố định là 256 KB, tệp này sẽ được chia thành khoảng 197 piece. Mỗi piece được đánh số thứ tự từ 0 đến 196. Với mỗi piece, hệ thống tính toán một giá trị băm SHA-256, ví dụ:

- Piece 0 → a93f...
- Piece 1 → 7c4e...
- ...

- Piece 196 → bf21...

Danh sách các giá trị băm này được lưu trong metadata của torrent. Khi một node tải piece 12 từ mạng, hệ thống sẽ kiểm tra lại giá trị băm của piece đó với hash đã lưu trong metadata. Nếu trùng khớp, piece được chấp nhận và ghi vào tệp tạm. Nếu không trùng, piece sẽ bị loại bỏ và được yêu cầu tải lại. Nhờ đó, hệ thống đảm bảo rằng dữ liệu cuối cùng luôn chính xác tuyệt đối so với tệp gốc.

2.5.2. Cơ chế xử lý song song bằng đa luồng (Multi-threaded Download Workers)

Để tối ưu hóa hiệu năng tải dữ liệu, hệ thống sử dụng mô hình xử lý song song dựa trên đa luồng. Sau khi nhận được danh sách các node nguồn từ tracker, node tải xuống sẽ khởi tạo một tập các luồng tải (worker threads), trong đó mỗi luồng tương ứng với một node nguồn. Các luồng này hoạt động độc lập và đồng thời, cùng chia sẻ một hàng đợi trung tâm chứa các piece chưa hoàn thành.

Mỗi luồng sẽ lấy một piece từ hàng đợi, gửi yêu cầu đến node nguồn tương ứng và chờ nhận dữ liệu. Khi một piece được tải và xác minh thành công, nó được ghi vào tệp tạm và bị loại khỏi hàng đợi. Luồng đó tiếp tục lấy piece tiếp theo cho đến khi hàng đợi rỗng. Nhờ cơ chế phân phối động này, hệ thống có khả năng tự cân bằng tải giữa các node nguồn, tận dụng tối đa băng thông mạng và giảm đáng kể tổng thời gian tải so với phương thức tải đơn nguồn.

Ví dụ: Giả sử một node tải xuống nhận được danh sách ba node nguồn từ tracker, bao gồm Peer A, Peer B và Peer C. Hệ thống sẽ tạo ba worker thread tương ứng với ba node nguồn này.

Tại thời điểm bắt đầu tải:

- Peer A lấy piece 0 từ hàng đợi và tải từ Peer A
- Peer B lấy piece 1 từ hàng đợi và tải từ Peer B
- Peer C lấy piece 2 từ hàng đợi và tải từ Peer C

Khi Worker A hoàn tất tải piece 0, nó tiếp tục lấy piece 3 từ hàng đợi để tải tiếp. Trong khi đó, Worker B và Worker C vẫn đang tải piece 1 và 2. Nhờ đó, các piece của cùng một tệp được tải song song từ nhiều nguồn, giúp giảm đáng kể tổng thời gian tải so với việc chỉ tải từ một node nguồn duy nhất.

2.5.3. Cơ chế chia sẻ và cung cấp dữ liệu giữa các peer (Peer-to-Peer Sharing Mechanism)

Trong kiến trúc P2P, mỗi node vừa có thể đóng vai trò là node tải xuống vừa có thể trở thành node chia sẻ dữ liệu (seeder). Khi một node hoàn tất việc tải một tệp, node đó có thể đăng ký chia sẻ lại tệp với tracker, trở thành một nguồn dữ liệu mới cho các node khác trong mạng. Tracker chỉ lưu trữ metadata và danh sách các node đang sở hữu dữ liệu, trong khi việc truyền dữ liệu thực tế được thực hiện trực tiếp giữa các peer.

Khi một node tải xuống yêu cầu một piece, các node nguồn sẽ đọc dữ liệu từ tệp gốc, chia piece thành các block nhỏ hơn để phù hợp với giới hạn MTU của UDP và truyền tuần tự các block này đến node yêu cầu. Cơ chế này cho phép nhiều node nguồn cùng lúc tham gia cung cấp các piece khác nhau của cùng một tệp, hình thành mô hình phân phối dữ liệu đa nguồn. Điều này giúp tăng tốc độ tải và nâng cao độ tin cậy của hệ thống khi một hoặc nhiều node nguồn gặp sự cố.

Ví dụ: Giả sử ban đầu chỉ có Peer A sở hữu tệp demo.txt và đã đăng ký chia sẻ với tracker. Khi Peer B tham gia tải demo.txt, tracker chỉ ra rằng Peer A là nguồn duy nhất. Peer B sẽ tải các piece từ Peer A.

Sau khi Peer B tải xong và đăng ký chia sẻ lại tệp này với tracker, lúc này hệ thống có hai node nguồn là Peer A và Peer B. Khi Peer C tiếp tục tải demo.txt, tracker sẽ trả về danh sách hai node nguồn. Peer C sẽ tải song song:

- Piece 0–50 từ Peer A
- Piece 51–100 từ Peer B

Như vậy, dữ liệu của một tệp được phân phối từ nhiều node nguồn khác nhau, vừa tăng tốc độ tải vừa giảm tải cho từng node riêng lẻ. Nếu trong quá trình tải Peer A bị mất kết nối, Peer C vẫn có thể tiếp tục tải các piece còn lại từ Peer B mà không bị gián đoạn.

IV. CÀI ĐẶT VÀ THỬ NGHIỆM

1. Hướng dẫn cài đặt và triển khai

1.1. Tải mã nguồn về máy

```
git clone https://github.com/looongtom/p2p-file-sharing.git

cd p2p-file-sharing

git checkout main
```

1.2. Build và chạy bằng Docker Compose

Bạn đang dùng docker/Dockerfile cho tracker/peers, và front-end/Dockerfile cho FE. Vì compose sử dụng đường dẫn context: .. nên bạn phải chạy lệnh trong thư mục docker/.

```
cd docker

docker compose up -d --build
```

docker	-	-	-	15.49%	4 minutes ago
bt-tracker	a0dbe322d173	docker-tracker	12345:12345 (UDP)	0.21%	4 minutes ago
bt-peer1	eb8e6f0202e8	docker-peer1	20001:20001 (UDP) Show all ports (2)	7.19%	4 minutes ago
bt-peer3	1de25345dd1a	docker-peer3	20003:20003 (UDP) Show all ports (2)	0.04%	4 minutes ago
bt-peer2	5e4089470043	docker-peer2	20002:20002 (UDP) Show all ports (2)	7.25%	4 minutes ago
fe-peer1	c4338cf97c1c	docker-fe-peer1	4200:4200	0.25%	4 minutes ago
fe-peer3	ec99acdd123d	docker-fe-peer3	4202:4200	0.25%	4 minutes ago
fe-peer2	b70db9835728	docker-fe-peer2	4201:4200	0.3%	4 minutes ago

1.3. Kiểm tra trạng thái cụm Docker

```
docker compose ps

CONTAINER ID   IMAGE          COMMAND              CREATED        STATUS        PORTS     NAMES

c4338cf97c1c   docker-fe-peer1   "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes   0.0.0.0:4200->4200/tcp, [::]:4200->4200/tcp

fe-peer1

ec99acdd123d   docker-fe-peer3   "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes   0.0.0.0:4202->4200/tcp, [::]:4202->4200/tcp

fe-peer3

b70db9835728   docker-fe-peer2   "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes   0.0.0.0:4201->4200/tcp, [::]:4201->4200/tcp

fe-peer2

eb8e6f0202e8   docker-peer1     "python peer/node.py…"   4 minutes ago   Up 4 minutes   0.0.0.0:20001->20001/udp, [::]:20001->20001/udp, 0.0.0.0:5001->5000/tcp, [::]:5001->5000/tcp   bt-peer1

1de25345dd1a   docker-peer3     "python peer/node.py…"   4 minutes ago   Up 4 minutes   0.0.0.0:20003->20003/udp, [::]:20003->20003/udp, 0.0.0.0:5003->5000/tcp, [::]:5003->5000/tcp   bt-peer3

5e4089470043   docker-peer2     "python peer/node.py…"   4 minutes ago   Up 4 minutes   0.0.0.0:20002->20002/udp, [::]:20002->20002/udp, 0.0.0.0:5002->5000/tcp, [::]:5002->5000/tcp   bt-peer2
```

```
a0dbe322d173   docker-tracker   "python tracker/trac…"   4 minutes ago   Up 4 minutes   0.0.0.0:12345->12345/udp, [::]:12345->12345/udp

bt-tracker
```

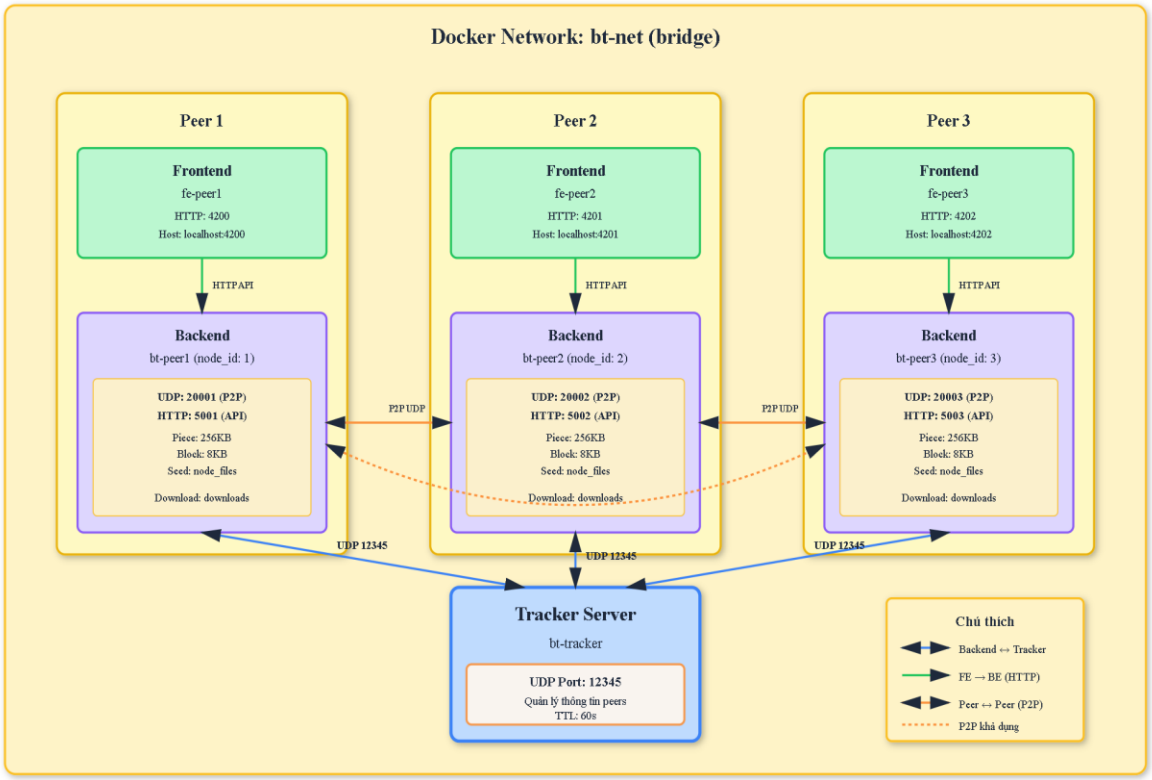
```
docker compose logs -f tracker

bt-tracker  | 2025-12-28T09:15:31 [TRACKER] listening udp 0.0.0.0:12345

bt-tracker  | 2025-12-28T09:15:38 [TRACKER] OWN ih=2727e463fc.. file=Xshell5.rar owner=peer1:20001

bt-tracker  | 2025-12-28T09:15:38 [TRACKER] OWN ih=2727e463fc.. file=Xshell5.rar owner=peer2:20002

bt-tracker  | 2025-12-28T09:16:40 [TRACKER] OWN ih=2727e463fc.. file=Xshell5.rar owner=peer3:20003
```

Dừng và xóa container/network

```
docker compose down
```

2. Mô tả kịch bản triển khai cụm Docker Compose

2.1. Thành phần service



Cụm gồm 7 container chính chạy chung network bt-net:

- **Bt-tracker (service: tracker)**
  - **Chạy:** python tracker/tracker.py
  - **UDP port:** 12345 (map ra host: 12345/udp)
  - Lưu DB tracker vào volume host: data/tracker_db
  - TTL loại peer offline: TRACKER_TTL_SEC = 60
- **Bt-peer1 / bt-peer2 / bt-peer3 (service: peer1/2/3)**

Mỗi peer BE có 2 kênh chính:

- **Kênh P2P UDP** (truyền piece/block):
  - peer1: UDP 20001
  - peer2: UDP 20002
  - peer3: UDP 20003
- **Kênh API Flask** (điều khiển/quan sát qua HTTP):
  - peer1: host 5001 -> container 5000
  - peer2: host 5002 -> container 5000
  - peer3: host 5003 -> container 5000

fe-peer1 / fe-peer2 / fe-peer3 (frontend)

- Build từ ../front-end/Dockerfile
- Port ra host:
  - fe-peer1: 4200
  - fe-peer2: 4201
  - fe-peer3: 4202
  - Mỗi FE phụ thuộc BE tương ứng (depends_on).

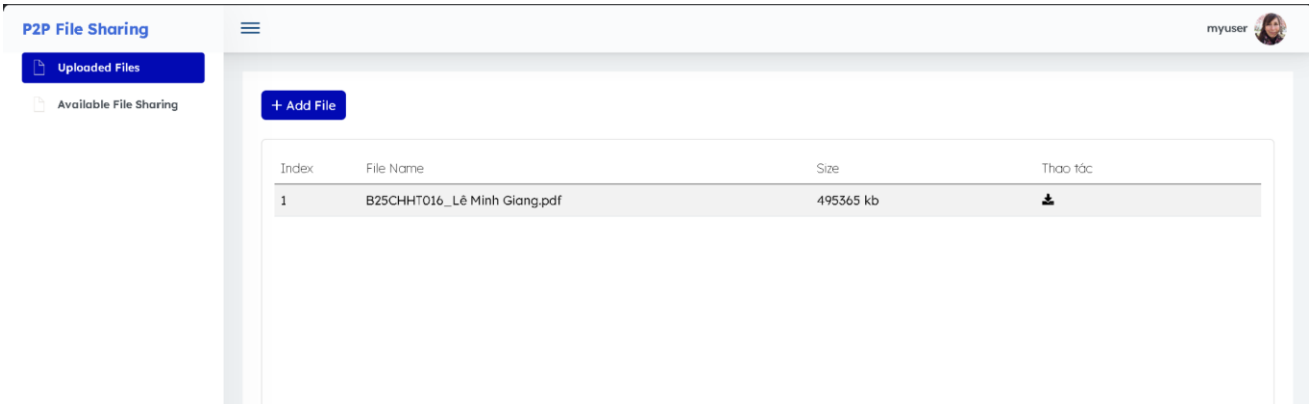
3. Kịch bản thử nghiệm và kết quả

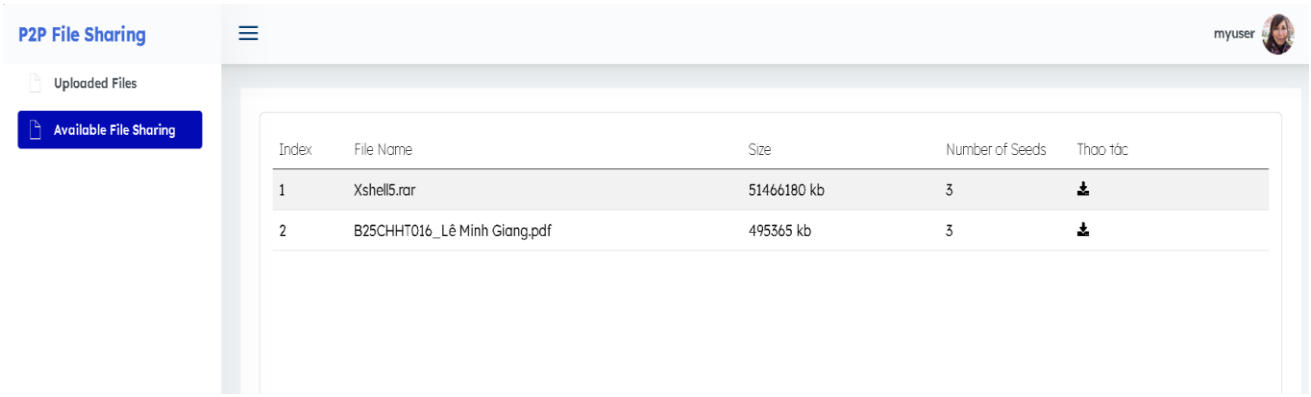
3.1. Kịch bản 1: Thử nghiệm chia sẻ tệp (upload/seeding)

Mục tiêu: Kiểm chứng peer có thể chia sẻ file và tracker ghi nhận torrent.

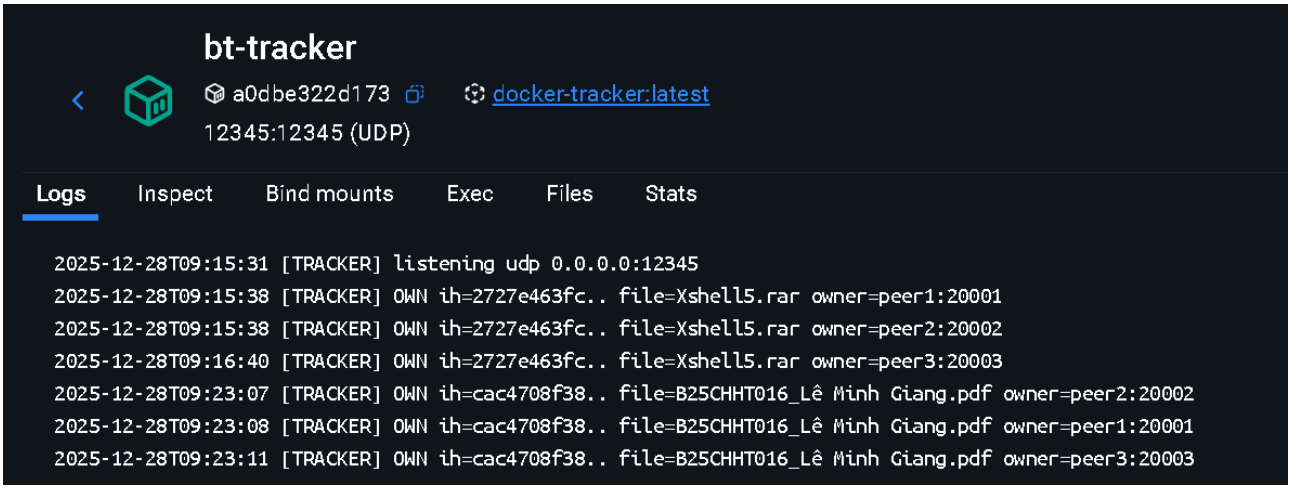
Các bước thực hiện:

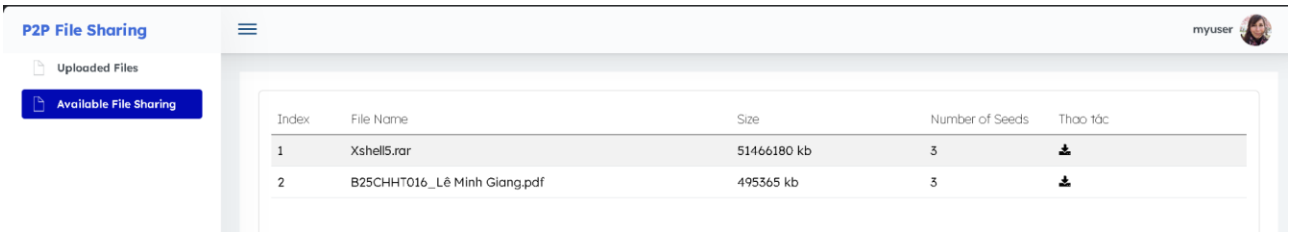
Bước 1: Peer tạo torrent và gửi thông tin file lên tracker (OWN).





Bước 2: Kiểm tra thông tin file qua log trên Tracker và thông tin các file sẵn sàng download



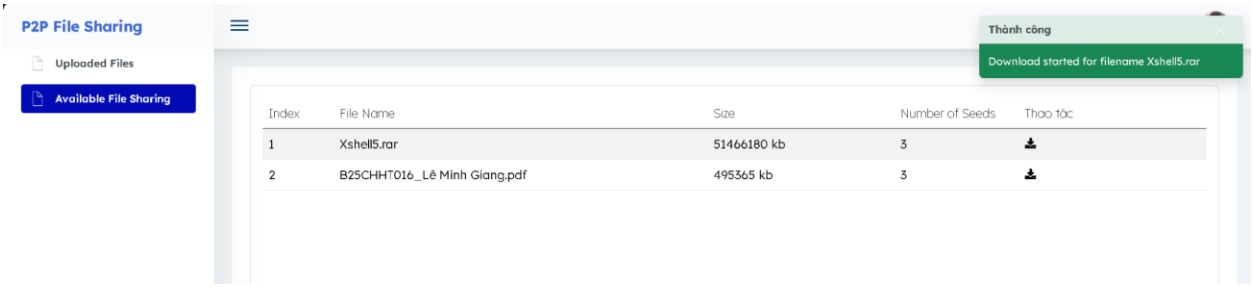


=> Kết quả: Tracker ghi nhận peer là owner của file.

3.2. Kịch bản 2: Thử nghiệm Download File

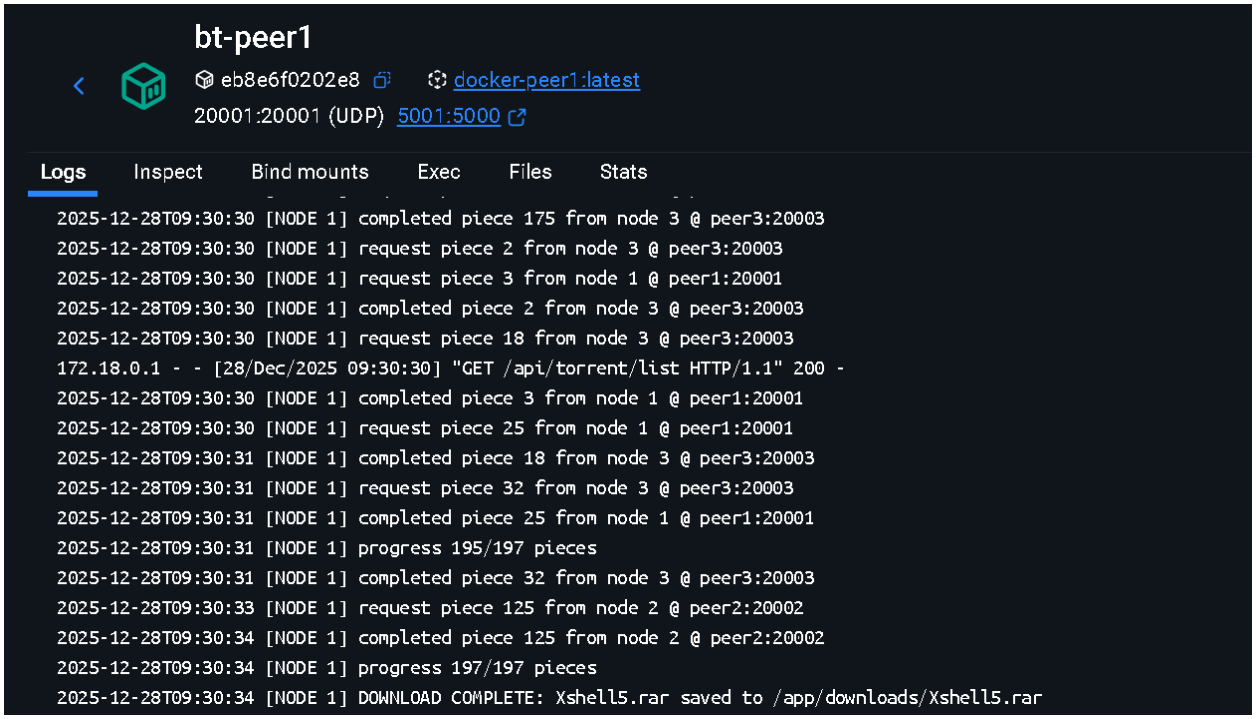
Mục tiêu: Kiểm chứng peer tải được file từ P2P

Bước 1: Thực hiện gửi yêu cầu download file từ mạng P2P

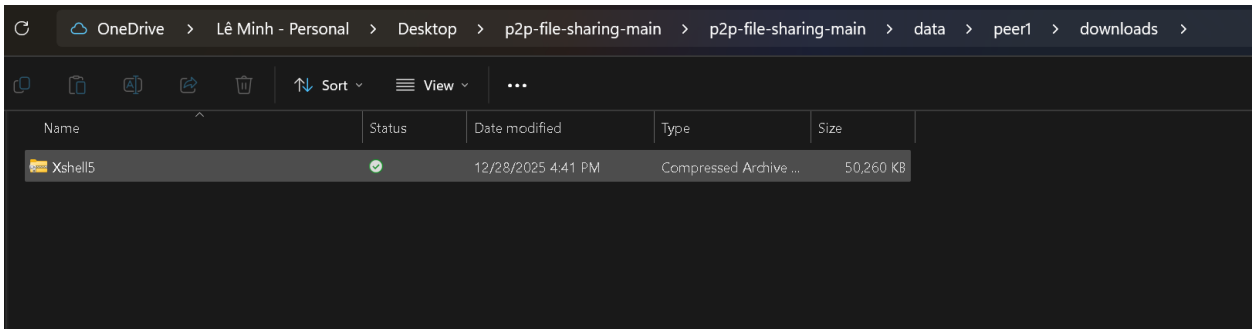


Bước 2: Kiểm tra quá trình download file qua log của peer

Log hệ thống hiển thị quá trình download file



File hoàn chỉnh xuất hiện trong thư mục downloads

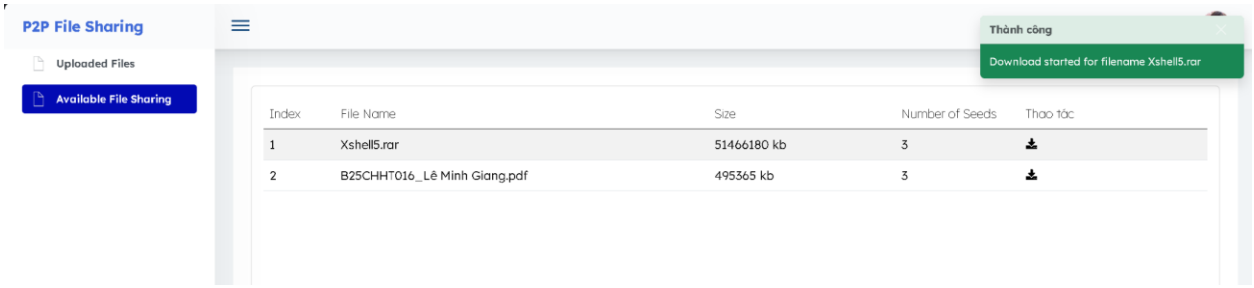


Kết quả: File tải về giống nội dung file gốc

3.3. Kịch bản 3: Thử nghiệm Resume Download

Mục tiêu: Kiểm chứng khả năng tiếp tục tải khi bị gián đoạn

Bước 1: Bắt đầu tải file trên peer3



Bước 2: Khi tải được một phần, dừng container peer3

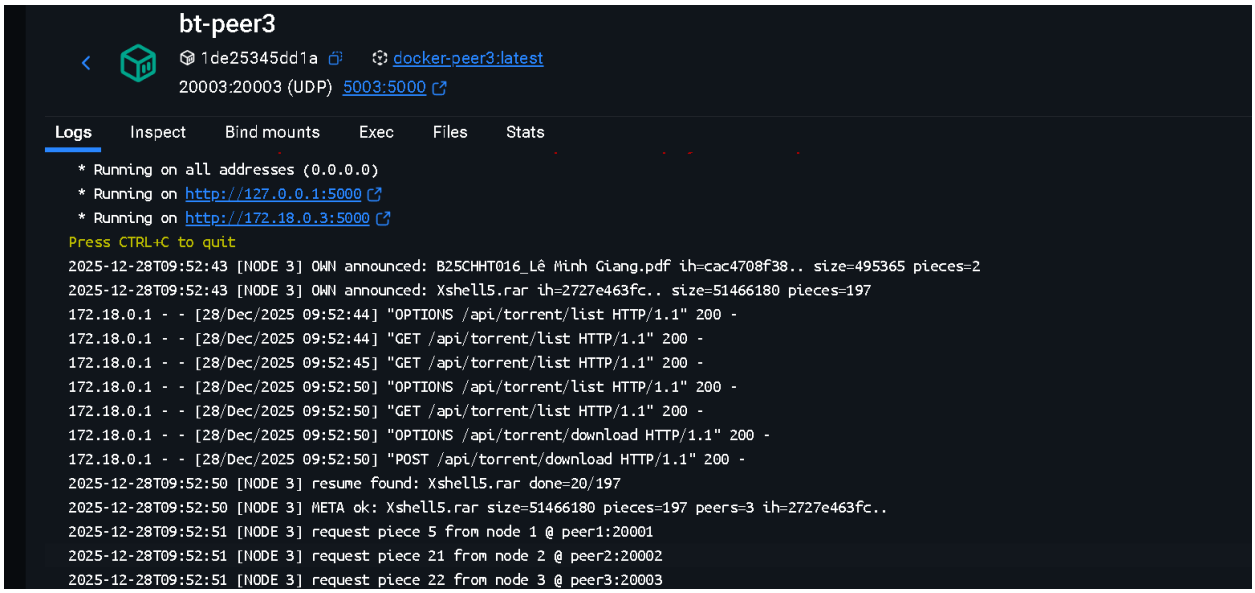
Bước 3: Khởi động lại peer3

Bước 4: Tiếp tục download file trên peer3 (kiểm tra và tiếp tục download dựa trên file *.part và *.resume.json)

Bước 5: Hoàn tất download file

Kết quả:

- Peer3 tiếp tục tải từ checkpoint, không tải lại từ đầu



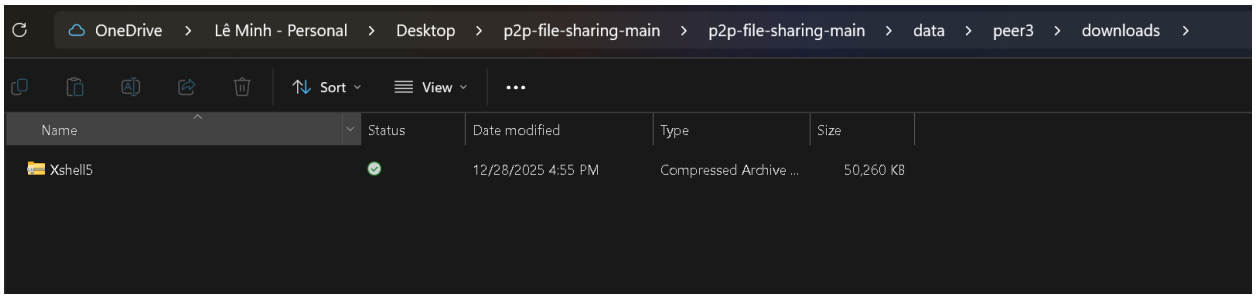
- Peer3 hoàn tất công việc download, file download về giống file gốc

```
2025-12-28T09:54:58 [NODE 3] completed piece 81 from node 3 @ peer3:20003
2025-12-28T09:54:58 [NODE 3] request piece 195 from node 3 @ peer3:20003
2025-12-28T09:54:59 [NODE 3] completed piece 125 from node 2 @ peer2:20002
2025-12-28T09:54:59 [NODE 3] request piece 39 from node 2 @ peer2:20002
2025-12-28T09:54:59 [NODE 3] completed piece 140 from node 1 @ peer1:20001
2025-12-28T09:54:59 [NODE 3] request piece 41 from node 1 @ peer1:20001
2025-12-28T09:54:59 [NODE 3] completed piece 195 from node 3 @ peer3:20003
2025-12-28T09:54:59 [NODE 3] progress 195/197 pieces
2025-12-28T09:54:59 [NODE 3] completed piece 39 from node 2 @ peer2:20002
2025-12-28T09:55:04 [NODE 3] request piece 41 from node 1 @ peer1:20001
2025-12-28T09:55:04 [NODE 3] completed piece 41 from node 1 @ peer1:20001
2025-12-28T09:55:04 [NODE 3] progress 197/197 pieces
2025-12-28T09:55:04 [NODE 3] DOWNLOAD COMPLETE: Xshell5.rar saved to /app/downloads/Xshell5.rar
```

- Peer3 hoàn tất công việc download, file download về giống file gốc


```
2025-12-28T09:54:58 [NODE 3] completed piece 81 from node 3 @ peer3:20003
2025-12-28T09:54:58 [NODE 3] request piece 195 from node 3 @ peer3:20003
2025-12-28T09:54:59 [NODE 3] completed piece 125 from node 2 @ peer2:20002
2025-12-28T09:54:59 [NODE 3] request piece 39 from node 2 @ peer2:20002
2025-12-28T09:54:59 [NODE 3] completed piece 140 from node 1 @ peer1:20001
2025-12-28T09:54:59 [NODE 3] request piece 41 from node 1 @ peer1:20001
2025-12-28T09:54:59 [NODE 3] completed piece 195 from node 3 @ peer3:20003
2025-12-28T09:54:59 [NODE 3] progress 195/197 pieces
2025-12-28T09:54:59 [NODE 3] completed piece 39 from node 2 @ peer2:20002
2025-12-28T09:55:04 [NODE 3] request piece 41 from node 1 @ peer1:20001
2025-12-28T09:55:04 [NODE 3] completed piece 41 from node 1 @ peer1:20001
2025-12-28T09:55:04 [NODE 3] progress 197/197 pieces
2025-12-28T09:55:04 [NODE 3] DOWNLOAD COMPLETE: Xshell5.rar saved to /app/downloads/Xshell5.rar
```


- File hoàn chỉnh xuất hiện trong data/peer3/downloads



V. KẾT LUẬN

Kết quả đạt được: Qua quá trình nghiên cứu lý thuyết và triển khai thực nghiệm, nhóm thực hiện đề tài đã xây dựng thành công ứng dụng chia sẻ tập tin dựa trên kiến trúc mạng ngang hàng (P2P). Đối chiếu với các yêu cầu phân tích ban đầu đặt ra ở Chương 1 và Chương 2, hệ thống đã đáp ứng được các mục tiêu cơ bản sau:

- **Về mặt chức năng:** Hệ thống đã hoàn thiện các chức năng cốt lõi của một ứng dụng P2P:
  - Các peer (nút mạng) có thể khởi tạo kết nối với nhau (hoặc với Tracker server tùy theo kiến trúc bạn chọn) thông qua giao thức TCP/IP
  - Người dùng có thể khai báo danh sách file muốn chia sẻ. Chức năng tìm kiếm trả về kết quả chính xác về địa chỉ IP và Port của peer đang lưu trữ file
  - Truyền tải tập tin (File Transfer):
    - Thực hiện thành công việc gửi và nhận file trực tiếp giữa hai peer.
    - Hỗ trợ chia nhỏ file (chunking) để truyền tải, giúp quản lý bộ đệm hiệu quả hơn.
  - Tích hợp cơ chế kiểm tra toàn vẹn file sau khi tải xong (sử dụng hàm băm MD5 hoặc SHA-256) để đảm bảo file không bị lỗi trong quá trình truyền
  - Xây dựng được giao diện (GUI/CLI) trực quan, cho phép người dùng dễ dàng thao tác kết nối, tìm kiếm và theo dõi tiến trình tải xuống
- **Về mặt phi chức năng (Non-functional Requirements)**
  - Ứng dụng xử lý được đồng thời nhiều kết nối. Một peer có thể vừa đóng vai trò là Server (upload file cho người khác) vừa là Client (download file từ người khác) cùng một lúc mà không bị chặn (blocking)
  - Hệ thống hoạt động ổn định trong môi trường mạng LAN với số lượng peer thử nghiệm hạn chế
- **Các hạn chế tồn tại:** Mặc dù đã hoàn thành các mục tiêu cơ bản, hệ thống vẫn còn tồn tại một số hạn chế về mặt kỹ thuật và quy mô triển khai:
  - Hệ thống hiện tại chỉ hoạt động tốt khi các peer có địa chỉ IP công khai (Public IP) hoặc nằm trong cùng một mạng nội bộ (LAN). Hệ thống chưa xử lý được việc kết nối trực tiếp giữa hai peer nằm sau hai bộ định tuyến (Router/NAT) khác nhau
  - Nếu một peer ngắt kết nối đột ngột (churn) trong quá trình đang truyền file, tiến trình tải xuống sẽ bị hủy bỏ thay vì tự động tìm kiếm một peer khác (resume/failover) để tải tiếp phần còn thiếu
  - Vấn đề bảo mật:
    - Dữ liệu truyền tải giữa các peer hiện đang ở dạng rõ (cleartext), chưa được mã hóa, dễ bị nghe lén (sniffing)
    - Chưa có cơ chế xác thực danh tính (Authentication) chặt chẽ, dẫn đến nguy cơ giả mạo peer (Sybil attack) hoặc phát tán file độc hại
  - Chưa tối ưu hóa thuật toán lựa chọn mảnh (piece selection strategy). Hệ thống tải tuần tự thay vì sử dụng thuật toán "Rarest-first" (mảnh hiếm nhất trước) như BitTorrent, dẫn đến tốc độ chưa tối ưu khi mạng lưới có ít seeder

- **Hướng phát triển:** Để nâng cấp hệ thống trở thành một ứng dụng thực tế và mạnh mẽ hơn, nhóm đề xuất các hướng phát triển trong tương lai như sau:
  - **Cải thiện kết nối mạng**
  - Nghiên cứu và áp dụng các giao thức như STUN (Session Traversal Utilities for NAT) hoặc TURN để cho phép các peer nằm sau tường lửa hoặc NAT có thể kết nối trực tiếp với nhau
  - Mở rộng khả năng định danh và kết nối trên nền tảng IPv6 để đón đầu xu hướng mạng tương lai
- **Nâng cao tính bảo mật**
  - Tích hợp giao thức TLS/SSL (Transport Layer Security) cho các kết nối socket để mã hóa toàn bộ dữ liệu truyền đi, đảm bảo tính bí mật và riêng tư
  - Xây dựng hệ thống đánh giá độ tin cậy của peer dựa trên lịch sử chia sẻ, giúp người dùng tránh tải file từ các nguồn độc hại
- **Tối ưu hóa hiệu năng và kiến trúc**
  - Thay vì dùng cơ chế tìm kiếm tập trung (nếu có) hoặc Flooding, sẽ chuyển sang sử dụng DHT (như Kademlia hoặc Chord) để tăng khả năng mở rộng (scalability) của hệ thống lên hàng ngàn nút mà không bị tắc nghẽn
  - Phát triển tính năng cho phép tải một file từ nhiều peer cùng lúc (swarming), giúp tăng tốc độ tải xuống tối đa
  - Lưu trạng thái các mảnh đã tải xuống đĩa cứng để có thể tiếp tục tải (resume) sau khi tắt ứng dụng hoặc mất mạng

VI. TÀI LIỆU THAM KHẢO