

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



Bài tập lớn 1

Mạng máy tính

---

Đề tài

Ứng dụng truyền tải Torrent

---

Giảng viên hướng dẫn: Vũ Thành Tài  
Sinh viên tham gia: 2212243 Trương Quang Nghĩa  
2212027 Đinh Công Minh  
2252260 Phạm Lê Huy  
2212537 Trần Thế Đại Phát

Ho Chi Minh, November 2024



## Contents

<b>1</b>	<b>Phân chia nhiệm vụ</b>	<b>2</b>
<b>2</b>	<b>Mở đầu</b>	<b>3</b>
2.1	Đặt vấn đề . . . . .	3
2.2	Những chức năng được mong đợi . . . . .	3
<b>3</b>	<b>Cơ sở lí thuyết</b>	<b>5</b>
3.1	Mô hình client-server . . . . .	5
3.2	Mô hình Peer-to-Peer . . . . .	5
3.3	Các giao thức được sử dụng . . . . .	7
3.3.1	HTTP . . . . .	7
3.3.2	TCP . . . . .	7
3.4	Multi-Direction Data Transfer (MDDT) . . . . .	8
<b>4</b>	<b>Mô tả dữ liệu được lưu trữ</b>	<b>9</b>
<b>5</b>	<b>Mô tả thiết kế của hệ thống</b>	<b>10</b>
<b>6</b>	<b>Mô tả chức năng của ứng dụng</b>	<b>11</b>
6.1	Client liên lạc tới tracker . . . . .	11
6.2	Phản hồi từ Tracker . . . . .	11
<b>7</b>	<b>Sử dụng ứng dụng</b>	<b>11</b>
7.1	Giải thích giao thức TCP giữa các Peer . . . . .	13
<b>8</b>	<b>Chiến thuật tải</b>	<b>18</b>
<b>9</b>	<b>Mã nguồn của dự án</b>	<b>19</b>



## 1 Phân chia nhiệm vụ

Họ và tên	MSSV	Công việc thực hiện	Mức độ đóng góp
Trương Quang Nghĩa	2212243	Hiện thực các chức năng của Tracker, viết báo cáo	110%
Đinh Công Minh	2212027	Hiện thực các chức năng của Client	110%
Phạm Lê Huy	2252260	Hiện thực các chức năng của Tracker	100%
Trần Thế Đại Phát	2212537	Viết báo cáo	80%

## 2 Mở đầu

### 2.1 Đặt vấn đề

Trong thời đại công nghệ thông tin hiện nay, khi nhu cầu trao đổi và truyền tải dữ liệu ngày càng gia tăng cả về khối lượng lẫn tốc độ, phương pháp truyền dữ liệu Peer-to-Peer (P2P) thông qua torrent đã trở thành một giải pháp thay thế đáng cân nhắc cho mô hình truyền thống client-server. Trong mô hình client-server, tất cả dữ liệu được lưu trữ và truyền tải từ một hoặc một số ít máy chủ. Điều này dễ dẫn đến tình trạng quá tải, đặc biệt khi có lượng lớn người dùng truy cập cùng lúc.

Với torrent, dữ liệu được chia nhỏ thành các mảnh và phân phối giữa các peer (máy ngang hàng). Các peer vừa tải xuống vừa cung cấp dữ liệu cho các peer khác, giảm áp lực lên bất kỳ máy chủ trung tâm nào. Torrent tận dụng băng thông của tất cả các máy tham gia mạng lưới. Một người dùng có thể nhận dữ liệu từ nhiều nguồn cùng lúc, thay vì chỉ từ một máy chủ. Điều này làm tăng tốc độ truyền tải, đặc biệt với các tệp lớn. Khi số lượng người dùng tăng, mô hình client-server cần nâng cấp cơ sở hạ tầng để đáp ứng nhu cầu. Trong khi đó, mạng P2P tự động mở rộng, vì mỗi người dùng mới không chỉ tiêu thụ dữ liệu mà còn đóng góp tài nguyên, tạo thành một hệ thống cân bằng tự nhiên.

Trong mạng client-server, nếu máy chủ trung tâm gặp sự cố hoặc bị tấn công, toàn bộ hệ thống có thể ngừng hoạt động. Mạng P2P không phụ thuộc vào một điểm duy nhất, nên vẫn hoạt động ngay cả khi một số node gặp vấn đề. Với mô hình P2P, không cần đầu tư mạnh vào cơ sở hạ tầng máy chủ hoặc băng thông lớn. Người dùng đóng vai trò là cả người tiêu dùng và nhà cung cấp tài nguyên, giảm chi phí cho nhà cung cấp dịch vụ. Torrent được tối ưu hóa cho việc chia sẻ các tệp lớn như phần mềm, video, hay dữ liệu nghiên cứu. Nhờ khả năng tải xuống song song từ nhiều nguồn, tốc độ và hiệu quả truyền tải được đảm bảo. Dữ liệu được chia nhỏ và không lưu trữ toàn bộ trên một máy duy nhất, giúp giảm nguy cơ bị đánh cắp hoặc bị phá hoại toàn bộ thông tin.

Với những ưu điểm đó, trong đề tài lần này, nhóm sẽ cố gắng hiện thực một ứng dụng sử dụng phương pháp truyền tệp ngang hàng (peer-to-peer) để truyền tải dữ liệu giữa các máy tính cá nhân với nhau. Ứng dụng được phát triển dưới dạng một ứng dụng đơn giản giống Torrent (STA) dựa trên yêu cầu của đề bài. Ứng dụng này được triển khai bằng cách sử dụng bộ giao thức TCP/IP, cho phép quá trình trao đổi dữ liệu diễn ra ổn định và đáng tin cậy. Ngoài ra, ứng dụng cũng hỗ trợ truyền dữ liệu đa hướng (MDDT), đảm bảo việc truyền dữ liệu giữa nhiều máy tính cùng một lúc, tăng hiệu suất và khả năng cộng tác giữa các thiết bị trong mạng lưới.

### 2.2 Những chức năng được mong đợi

Mục tiêu của nhóm khi xây dựng ứng dụng là:

#### Về server

- Server kiểm soát tài khoản người dùng, cũng như thông tin của các peer hiện đang tham gia trong mạng



- Có khả năng phản hồi một cách phù hợp trước những yêu cầu được gửi tới từ client
- Cung cấp chiến lược tải file phù hợp.
- Có khả năng thực hiện ping và discover

#### **Về client**

- Có khả năng đăng ký, đăng nhập và đăng xuất để sử dụng các dịch vụ của server
- Có thể thông báo đến tracker về những file mà nó đang có
- Gửi yêu cầu tải file đến tracker
- Thực hiện được các lệnh publish và fetch

#### **Về truyền tải tệp dữ liệu**

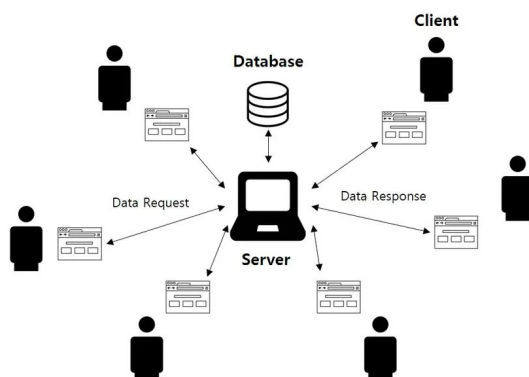
- Ứng dụng có thể truyền tải torrent, sử dụng để phương pháp truyền ngang hàng (peer to peer) giữa các máy tính. Hỗ trợ truyền torrent chứ một hoặc nhiều file.
- Hỗ trợ khả năng truyền tệp giữa các máy tính và đảm bảo tính toàn vẹn của dữ liệu
- Hỗ trợ truyền tệp có kích thước lớn, mà không có sự can thiệp của server.
- Các dữ liệu được tải đồng thời thông qua đa luồng.

### 3 Cơ sở lí thuyết

#### 3.1 Mô hình client-server

Mô hình client-server là một kiến trúc mạng phổ biến trong các ứng dụng hiện nay, trong đó có một máy chủ (server) luôn hoạt động và chịu trách nhiệm xử lý yêu cầu từ nhiều máy khách (clients). Server thường có một địa chỉ IP cố định, cho phép clients liên tục gửi yêu cầu tới máy chủ mà không gặp khó khăn trong việc tìm kiếm địa chỉ. Một ví dụ điển hình là ứng dụng web, nơi máy chủ web phục vụ các yêu cầu từ trình duyệt chạy trên các máy khách. Khi nhận được yêu cầu cho một tài nguyên từ máy khách, máy chủ sẽ gửi trả lại tài nguyên đó cho máy khách.

Trong kiến trúc client-server, các máy khách không giao tiếp trực tiếp với nhau, tất cả các tương tác diễn ra thông qua server. Điều này không chỉ giúp đơn giản hóa quá trình quản lý dữ liệu mà còn đảm bảo tính bảo mật và kiểm soát dễ dàng hơn. Các ứng dụng nổi tiếng như Web, FTP, Telnet, và email đều sử dụng mô hình này, trong đó server đóng vai trò trung tâm để quản lý và phân phối thông tin.



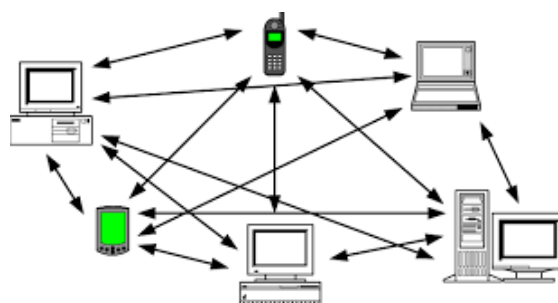
Hình 1: Mô hình client-server.

Một trong những thách thức của mô hình client-server là khi một máy chủ duy nhất không thể xử lý đủ yêu cầu từ tất cả các máy khách. Chẳng hạn, một trang mạng xã hội nổi tiếng có thể nhanh chóng bị quá tải nếu chỉ có một máy chủ.

#### 3.2 Mô hình Peer-to-Peer

Mô hình mạng ngang hàng (Peer-to-Peer) là một kiến trúc phân tán, trong đó các máy chủ và máy khách không cần phụ thuộc vào một máy chủ trung tâm để giao tiếp. Thay vì thông qua một server cố định, các thiết bị cá nhân như máy tính để bàn, laptop từ nhà, trường học hoặc văn phòng trực tiếp kết nối với nhau để trao đổi dữ liệu. Vì lý do này, mô hình này được gọi là P2P, cho phép các máy giao tiếp tự do mà không có sự can thiệp của máy chủ. Một trong những ứng dụng nổi tiếng của kiến trúc này là BitTorrent, nền tảng chia sẻ file phổ biến. Điều này giúp loại bỏ gánh nặng về việc quản lý một hệ thống máy chủ lớn như trong mô hình client-server truyền thống.

Một trong những ưu điểm quan trọng của mô hình P2P là khả năng tự mở rộng. Trong các ứng dụng P2P, mỗi máy không chỉ tạo ra lưu lượng công việc bằng cách yêu cầu các tệp tin mà còn góp phần vào năng lực xử lý của hệ thống bằng cách chia sẻ tệp tin với các máy khác. Điều này có nghĩa là khi số lượng máy trong hệ thống tăng lên, dung lượng và khả năng phục vụ của hệ thống cũng tăng theo. Khác với mô hình client-server, nơi máy chủ có thể bị quá tải khi có quá nhiều yêu cầu từ máy khách, mô hình P2P khắc phục vấn đề này bằng cách cho phép mỗi máy trở thành một phần của hạ tầng phục vụ. Điều này làm cho P2P trở nên hiệu quả và tiết kiệm hơn về mặt chi phí hạ tầng, vì nó không đòi hỏi một trung tâm dữ liệu lớn hay băng thông máy chủ mạnh mẽ.

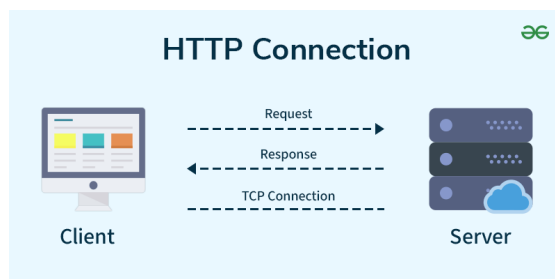


Hình 2: Mô hình Peer-to-peer.

Tuy nhiên, mô hình P2P cũng gặp phải nhiều thách thức, đặc biệt về bảo mật, hiệu suất và độ tin cậy. Với tính chất phi tập trung, hệ thống khó kiểm soát và quản lý, việc ngăn chặn các cuộc tấn công hay bảo vệ dữ liệu trở nên phức tạp hơn. Bên cạnh đó, hiệu suất của một ứng dụng P2P có thể bị ảnh hưởng bởi sự chậm trễ hay không ổn định của các máy trong hệ thống, vì không có máy chủ trung tâm để điều phối và đảm bảo tính nhất quán của dữ liệu. Mô hình này cũng yêu cầu sự tin cậy giữa các peer, điều không phải lúc nào cũng dễ đảm bảo trong các hệ thống lớn với nhiều thành viên không xác định.

### 3.3 Các giao thức được sử dụng

#### 3.3.1 HTTP



Hình 3: Giao thức HTTP.

HyperText Transfer Protocol (HTTP) là một giao thức ứng dụng dựa trên mô hình client-server, được sử dụng để trao đổi thông tin giữa các thiết bị qua mạng. Đặc điểm nổi bật của HTTP là nó phi trạng thái (stateless), các yêu cầu/đáp ứng (request/response) đều độc lập với nhau, không lưu trạng thái giữa các lần giao tiếp. Điều này giúp giảm tải tài nguyên nhưng cần cơ chế bổ sung (như cookies, sessions) để quản lý trạng thái. HTTP truyền dữ liệu qua kết nối TCP/IP. Cổng mặc định là 80 (HTTP) và 443 (HTTPS). Client gửi yêu cầu (request) đến máy chủ. Máy chủ phản hồi (response) chứa dữ liệu như HTML, JSON, hoặc các tài nguyên khác.

Trong đề tài lần này, client sẽ dùng phương thức HTTP để liên hệ đến Tracker để thực hiện các yêu cầu về thông báo và yêu cầu truyền tải. Vấn đề này sẽ được đề cập kỹ hơn bên dưới.

#### 3.3.2 TCP

TCP (Transmission Control Protocol) là một giao thức thuộc tầng Transport trong mô hình OSI và TCP/IP, được thiết kế để cung cấp kết nối tin cậy, đảm bảo dữ liệu được truyền chính xác giữa hai thiết bị trong mạng. Trước khi truyền dữ liệu, TCP thiết lập một kết nối thông qua quá trình "bắt tay 3 bước" (three-way handshake). Kết nối này đảm bảo cả hai phía đồng ý giao tiếp trước khi gửi dữ liệu. TCP sử dụng cơ chế cửa sổ trượt (Sliding Window) để đảm bảo không gửi quá nhiều dữ liệu khiến phía nhận bị quá tải. TCP điều chỉnh tốc độ truyền dựa trên điều kiện mạng nhằm giảm thiểu tình trạng tắc nghẽn. TCP hỗ trợ truyền dữ liệu đồng thời giữa hai bên TCP sử dụng mã kiểm tra (Checksum) để phát hiện lỗi trong quá trình truyền dữ liệu.

Nhờ sự đáng tin cậy trong việc truyền dữ liệu, trong đề tài này, nhóm đã hiện thực giao thức truyền tải file dữ liệu thông qua TCP thay vì phương thức không kém phổ biến là UDP.



### 3.4 Multi-Direction Data Transfer (MDDT)

Truyền dữ liệu đa hướng (Multi-Direction Data Transfer - MDDT) là một tính năng quan trọng giúp các ứng dụng đạt được hiệu suất hoạt động tối ưu. Nhờ MDDT, một node trong mạng có thể tải xuống nhiều tệp từ nhiều nguồn khác nhau đồng thời, thay vì phải chờ từng tệp một hoặc chỉ có thể nhận dữ liệu từ một nguồn duy nhất tại một thời điểm. Cơ chế này không chỉ giúp cải thiện đáng kể tốc độ truyền tải mà còn tối ưu hóa việc sử dụng băng thông mạng.

Với MDDT, khi một node gửi yêu cầu tải xuống, nó sẽ phân tán yêu cầu này tới nhiều nguồn có sẵn trên mạng, có thể bao gồm các server khác nhau hoặc thậm chí các thiết bị khác có cùng dữ liệu. Các tệp sau đó sẽ được chia nhỏ và tải xuống từ các nguồn khác nhau, nhờ đó rút ngắn đáng kể thời gian truyền tải. Quá trình này tương tự như việc tải xuống một tệp từ nhiều phần khác nhau, cho phép hệ thống tận dụng triệt để băng thông của mạng và giảm thiểu tác động của các nút thắt cổ chai trong quá trình truyền dữ liệu.

Thêm vào đó, MDDT cũng giúp tăng cường độ tin cậy và ổn định của kết nối, vì nếu một nguồn gặp trục trặc hoặc ngừng cung cấp dữ liệu, các phần còn lại vẫn tiếp tục được truyền từ những nguồn khác. Điều này đặc biệt hữu ích trong các mạng quy mô lớn hoặc trong các trường hợp mạng có độ trễ cao, nơi việc tải xuống dữ liệu từ một nguồn duy nhất có thể dễ bị gián đoạn. Với khả năng khai thác toàn bộ tài nguyên mạng, MDDT không chỉ cải thiện tốc độ mà còn đảm bảo dữ liệu đến đúng hạn và đúng yêu cầu, tạo ra sự linh hoạt và độ ổn định cao cho các ứng dụng STA trong môi trường truyền tải dữ liệu đa hướng.

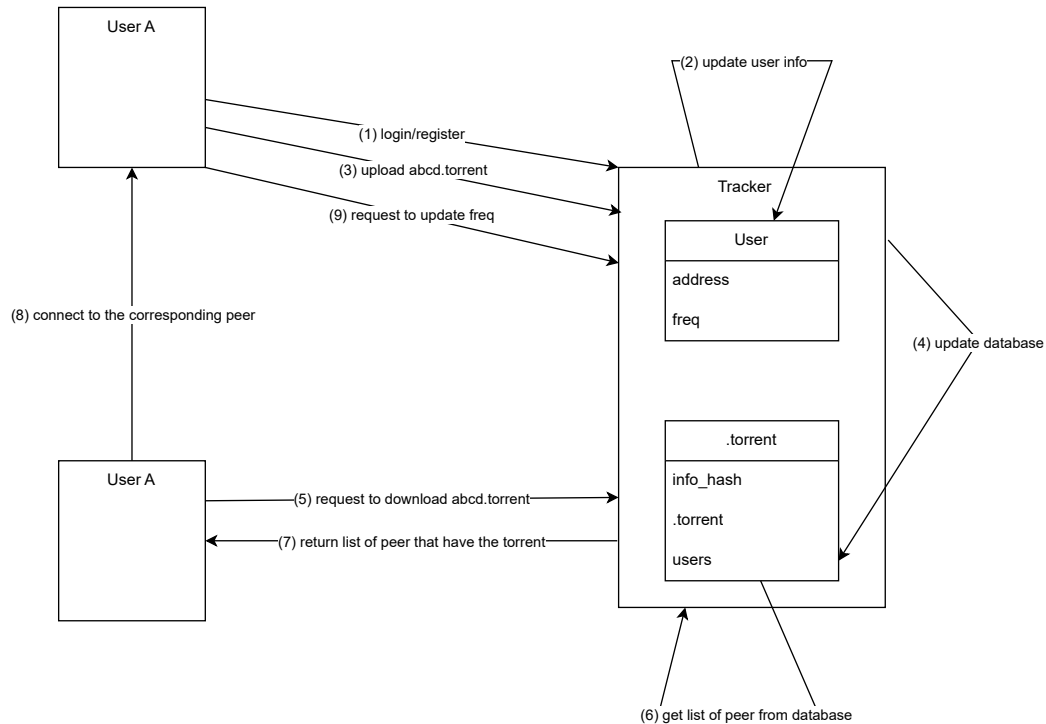
## 4 Mô tả dữ liệu được lưu trữ

User (Tài khoản người dùng)				
No	Attribute	Assigned Code	Unit	Type of data
1	Tên tài khoản	username	String	String
2	Mật khẩu	password	String	String
3	Tần số đăng tải	freq	Number	Numeric

Peer (Node đại diện cho người dùng bên trong swarm)				
No	Attribute	Assigned Code	Unit	Type of data
1	Khóa ngoại trỏ tới User	user	User	Object
2	Mã của Peer	node_id	Number	Numeric
3	Địa chỉ IP của người dùng	address	String	string
4	Danh sách các torrent mà peer đang cho phép upload	torrents	List	List of string
5	Lần cuối truy cập	last_seen	Datetime	Date
6	Trạng thái của người dùng	is_active	True/False	Boolean

Torrent (Thông tin về torrent mà người dùng đã upload)				
No	Attribute	Assigned Code	Unit	Type of data
1	Khóa ngoại trỏ tới Peer	peer	Peer	Object
2	Mã info_hash của torrent	info_hash	String	String
3	Kích thước của một piece	piece_length	Number	Numeric
4	Danh sách các file mà torrent chứa	filenames	List	List of string

## 5 Mô tả thiết kế của hệ thống



1. Người dùng đăng ký tài khoản hoặc đăng nhập vào hệ thống để có thể sử dụng ứng dụng
2. Tracker kiểm tra thông tin người dùng và trả về những phản hồi tương ứng
3. Người dùng sử dụng lên upload để thông báo việc chia sẻ torrent cho Tracker
4. Tracker cập nhập thông tin vào hệ thống
5. Người dùng khác trong Swarm sẽ gửi yêu cầu tải torrent đến Tracker
6. Tracker truy xuất từ hệ cơ sở dữ liệu và lấy danh sách địa chỉ của các Peer có file torrent tương ứng
7. Tracker gửi danh sách phản hồi lại cho người dùng đã yêu cầu
8. Người dùng thực hiện kết nối TCP đến các Peer nằm trong danh sách, sử dụng giải thuật leeching (lựa chọn 4 peer có tần số tải thấp nhất).

## 6 Mô tả chức năng của ứng dụng

### 6.1 Client liên lạc tới tracker

- Client gửi yêu cầu đăng ký đến tracker (register): username, password
- Client gửi yêu cầu đăng nhập đến tracker (login): username, password
- Client gửi yêu cầu đăng xuất đến tracker (logout): username
- Client gửi thông báo về torrent mà nó muốn đăng tải đến tracker (publish): file.torrent
- Sau khi client bắt đầu đăng tải torrent, nó sẽ được xem là vào bên trong mạng P2P, khi nằm trong mạng và ở chế độ cho người khác tải file từ chính nó, mỗi 15 giây, client sẽ gửi một thông báo trên tracker rằng nó vẫn còn nằm bên trong mạng.
- Client gửi yêu cầu torrent mà nó muốn đến tracker (fetch): file.torrent

### 6.2 Phản hồi từ Tracker

- Phản hồi khi có đăng ký: Tracker kiểm tra thông tin trong database và lưu dữ liệu mới của người dùng nếu thông tin là phù hợp.
- Phản hồi khi có đăng nhập: Tracker kiểm tra thông tin trong database và gửi xác nhận cho đăng nhập nếu thông tin yêu cầu là phù hợp.
- Phản hồi khi có đăng xuất: Tracker kiểm tra thông tin người dùng trong database và xóa thông tin về Peer khỏi Swarm.
- Phản hồi khi có yêu cầu đăng tải torrent: Một thực thể Peer được tạo ra nếu Peer chưa từng tồn tại. Peer sẽ lưu trữ danh sách các torrent mà nó đang cho phép các Peer khác tải về và địa chỉ của chính nó.
- Mỗi 15 giây, tracker được kỳ vọng sẽ nhận được những thông báo từ client về việc họ vẫn nằm trong mạng, nếu tracker không nhận được thông báo đó từ phía client, mọi thông tin về client đó sẽ bị loại bỏ khỏi mạng.
- Phản hồi khi có yêu cầu tải torrent: Tracker trả về danh sách các địa chỉ mà nó có thể liên lạc đến để tải torrent.

## 7 Sử dụng ứng dụng

1. Khởi động tracker, tracker chạy ở localhost với port là 8000

```
System check identified no issues (0 silenced).
November 22, 2024 - 07:50:47
Django version 5.1.3, using settings 'tracker.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

## 2. Chạy chương trình cho người dùng

- Chọn cho người dùng một địa chỉ IP và một mã id
- Người dùng có thể lựa chọn giữa đăng nhập/ đăng ký.
- Giao diện khi đăng ký thành công

```
you are running at port 1234
1.register 2.login q.quit: 1
Enter your username: nghia1
Password: 12345
User created successfully: {'message': 'User created successfully', 'username': 'nghia1'}
```

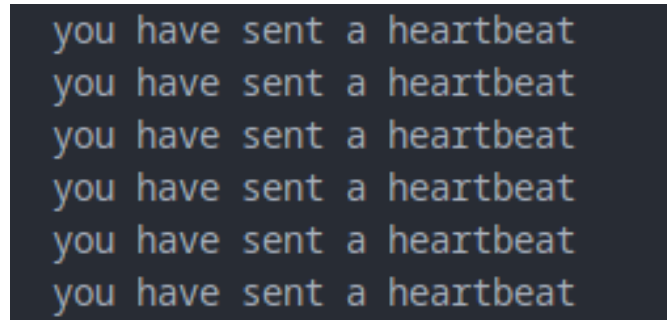
- Giao diện khi đăng nhập thành công

```
1.register 2.login q.quit: 2
Enter your username: nghia1
Password: 12345
Login successful
```

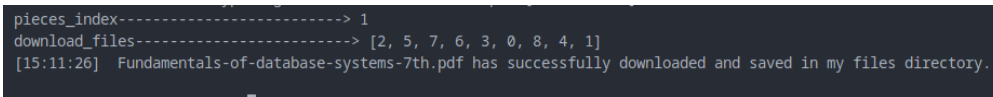
- Giao diện khi publish file torrent.

```
Enter files you wanted to publish: fundamental.torrent
Your trying to add files ['fundamental.torrent']
-----
[15:04:14] Tracker response: {'message': 'Torrent added successfully', 'filenames': ['1e23591b250f27a375680b57a3a6b74d581a424a']}
[15:04:14] Torrent uploaded! Waiting for other node to send requests!
3.publish 4.fetch q.quit: you have sent a heartbeat
```

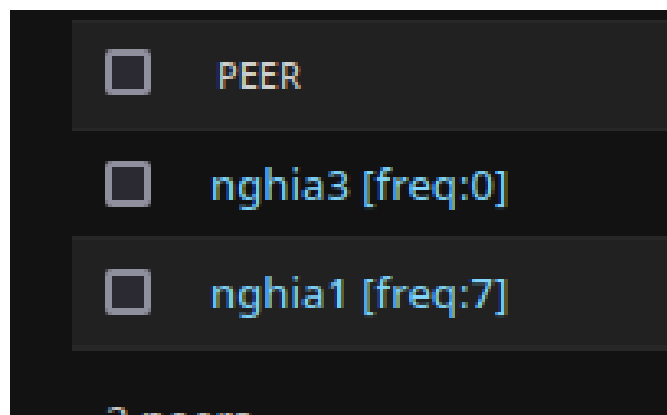
- Sau khi upload torrent thành công, client bây giờ đã là một Peer. Với mỗi 15 giây, Peer sẽ gửi thông báo đến tracker rằng nó vẫn còn tồn tại bên trong mạng



- Peer khác có thể tải torrent đang có bên trong mạng (Thứ tự tải chunk là thứ tự ưu tiên tải những chunk hiếm nhất)



- Frequency của Peer sẽ được cập nhật nếu có Peer khác tải file từ nó



## 7.1 Giải thích giao thức TCP giữa các Peer

### Giao tiếp giữa các nodes có trong swarm:

- Mỗi node khi đã thực hiện hành động upload sẽ khởi động một luồng (thread) mới để chạy hàm listen(), luồng này sau đó sẽ lắng nghe để nhận yêu cầu liên kết từ các node khác.
- Khi node muốn giao tiếp nó sẽ dùng send-segment() để tạo kết nối và muốn gửi request .
- Node sẽ dùng handle\_request() để xử lý tin nhận được.

### Kết nối với các Node khi muốn download file:

- Node sẽ lấy thông tin từ file .torrent về các file có trong đó và piece-length của từng chunk.
- Node sẽ thực hiện download một cách đồng thời các file giữa các node được có trong peer list mà Tracker trả về sử dụng multithread với target là hàm set-download-mode.
- Trước hết, node cần biết rõ là từng file cần tải được chia làm nhiều chunk, các chunk và dữ liệu của nó được track trả về cho node. Khi đó node sẽ giao tiếp với các node khác trong danh sách được Tracker trả về bằng giao thức TCP và bắt đầu truyền dữ liệu cho nhau.

```
elif mode == 'download':
    # t = Thread(target=node.set_download_mode, args=(filename,))
    # t.setDaemon(True)
    # t.start()
    threads = []
    for filename in filenames:
        thread = threading.Thread(target=node.set_download_mode, args=(filename, filenames_0[0], parser_metadata))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()
```

### Giải thuật để tải các chunk:

- Node sẽ lấy thông tin từ file .torrent về các file có trong đó và piece-length của từng chunk.
- Đầu tiên, file sẽ được chia ra thành những chunk chứa một phần nhỏ dữ liệu. Tracker sẽ lưu trữ thông tin tất cả các node có file .torrent mà client cần download và phải trả về danh sách đó cho client, tiếp đó sẽ sử dụng giải thuật rarest-first chunk, tức là khi có được danh sách list node có file torrent bên client sẽ tổng hợp lại từ danh sách piece đó .theo dạng là 0:3,1:2,..., tức là piece 0 có 3 node có, piece thứ 1 có 2 node có, nhờ hàm ask-file-size.
- Bởi khi khởi tạo node sẽ có trường files và trường này lưu trữ các files mà node có và bitfield thể hiện các chunk mà node đó có theo từng file, piece-count tổng số chunk cần chia theo từng file

```
class Node:
    def __init__(self, node_id: int, port: int, node_ip: str):
        self.send_socket = set_socket(port, node_ip)
        self.connection = False
        self.node_id = node_id
        self.node_ip = node_ip
        self.port = port
        self.files = self.fetch_owned_files()
        self.is_in_send_mode = False # is thread uploading a file or not
        self.downloaded_files = {}
        self.shared_file_lock = Lock()
        self.swarm_lock = Lock()
```

```
def psk_file_size(self, filename: str, peer_index, bitfield_pieces_count, to_be_used_owners: list) -> int:
    peer_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    dest_node = to_be_used_owners[peer_index][0]
    # print(dest_node["addr"][0], dest_node["addr"][1])
    msg = Node2Node(src_node_id=self.node_id,
                    dest_node_id=dest_node["node_id"],
                    filename=filename)
    self.send_segment(peer_socket,
                      data=msg.encode(),
                      addr=(dest_node["addr"][0], dest_node["addr"][1]))
    if peer_socket.fileno() == -1:
        self.swarm_lock.acquire()
        print("Socket is closed.")
        to_be_used_owners.pop(peer_index)
        self.swarm_lock.release()
        return
    while True:
        data = peer_socket.recv(config.constants.BUFFER_SIZE)
        dest_node_response = Message.decode(data)
        # print("dest_node_response----->", dest_node_response)
        if dest_node_response is None:
            to_be_used_owners.pop(peer_index)
            return
        size = dest_node_response["size"]
        bitfield_pieces = dest_node_response["bitfield_pieces"] # nhận bitfield_pieces từ node cho và tổng hợp
        target_bitfield_pieces = self.extract_pieces(bitfield_pieces)
        # print("target_bitfield_pieces----->", target_bitfield_pieces)
        update_bitfield.acquire()
        self.update_bitfield_count(target_bitfield_pieces, bitfield_pieces_count, dest_node_response["pieces_count"])
        update_bitfield.release()
        free_socket(peer_socket)
        return {
            "file_size": size,
            "pieces_count": dest_node_response["pieces_count"]
        }
```

```
def fetch_owned_files(self) -> list:
    files = []
    # node files dir = config.directory.node_files_dir + 'node' + str(self.node_id)
    node_files_path = config.directory.node_files_dir + 'node' + str(self.node_id) + "/" + "statusFile.json"
    with open(node_files_path, 'r') as file:
        data = json.load(file)
    for file_id, file_data in data.items():
        filename = file_data["filename"]
        # Use 'ast.literal_eval' to convert the byte string safely
        bitfield_hex = file_data["bitfield_pieces"]
        pieces_count = file_data["pieces_count"]
        bitfield_hex_1 = set(bitfield_hex)
        target_bitfield_hex_1 = self.create_bitfield_message(bitfield_hex_1, pieces_count)
        # bitfield = byte_format = bytes.fromhex(bitfield_hex)

        # Display the result
        target_file = {
            "fileId": file_id,
            "filename": filename,
            "bitfield_pieces": target_bitfield_hex_1,
            "pieces_count": pieces_count
        }
        files.append(target_file)
    print("specific_files:::", files)
    return files
```



```
client.py x node.py M {} statusfile.json ...node1 M {} statusfile.json ...node1 M x
assignment 1 > node_files > node1 > {} statusfile.json > {} 1 > {} bitfield_pieces
1 {
2   "1": {
3     "filename": "file_8.txt",
4     "bitfield_pieces": [
5       0,
6       1,
7       2,
8       3,
9       4,
10      5,
11      6,
12      7,
13      8,
14      9,
15      10,
16      11,
17      12,
18      13,
19      14,
20      15,
21      16,
22      17,
23      18,
24      19,
25      20,
26      21,
27      22,
28      23,
29      24,
30      25,
31      26,
32      27,
33      28
34    ],
35    "pieces_count": 29
36  },
37  "2": {
```

```
def update_bitfield_count(self, bitfield_pieces, bitfield_pieces_count, piece_count):
    for piece in bitfield_pieces :
        if piece in bitfield_pieces_count.keys():
            bitfield_pieces_count[piece] += 1
        else:
            bitfield_pieces_count[piece] = 1
```

- Phần download vòng lặp while khi mà nó có đầy đủ các chunk có trong bitfield-pieces-count thì sẽ dừng lại, nhóm sử dụng thuật toán rarest-first tức nó sẽ lấy ra mảng có tối đa 4 piece hiếm nhất để lấy tải đồng thời với mảng peer list được sắp xếp theo frequency của node. Client sẽ gửi đồng thời request từng piece đến với từng node có trong list sử dụng multithread.

```
while not self.download_complete(filename, length count):
    file_path = f"[config.directory.node_files_dir]node{self.node_id}/{filename}"
    pieces = self.piece_selection_startergy(bitfield_pieces_count)
    downloading_thread_pool = []
    for i in range(min(len(pieces), len(to_be_used_owners))):
        piece = pieces[i]
        peer_index = to_be_used_owners[i]
        #28
        downloading_thread = Thread(target=self.download_piece, args=(piece, peer_index, piece_length, bitfield_pieces_count, file_path))
        downloading_thread_pool.append(downloading_thread)
        downloading_thread.start()
    for downloading_thread in downloading_thread_pool:
        downloading_thread.join()

    return True

def piece_selection_startergy(self, bitfield_pieces_count):
    return self.rarest_pieces_first(bitfield_pieces_count)

def rarest_pieces_first(self, bitfield_pieces_count):
    # check if bitfields are recieved else wait for some time
    while(len(bitfield_pieces_count) == 0):
        time.sleep(5)
    # get the rarest count of the pieces
    rarest_piece_count = min(bitfield_pieces_count.values())
    # find all the pieces with the rarest piece
    rarest_pieces = [piece for piece in bitfield_pieces_count if
                     bitfield_pieces_count[piece] == rarest_piece_count]
    # shuffle among the random pieces
    random.shuffle(rarest_pieces)
    # rarest pieces
    return rarest_pieces[:4]
```

Kết nối giữa các node khi muốn upload file:

- Các nodes khởi tạo chế độ gửi thông báo qua phương thức set\_send\_mode() sau đó node xử lý yêu cầu tạo chunk từ các nodes khác sử dụng đa luồng.

```
##### Send mode #####
if mode == 'send':
    # node.set_send_mode(filename=filename)
    threads = []
    for filename in filenames:
        thread = threading.Thread(target=node.set_send_mode, args=(filename, parser_metadata))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()
```

- Ngoài ra node còn cung cấp thông tin về bit\_field mà nó có và các thông tin cần thiết cho từng file mà nó có thông qua tell\_file\_size().

```
def tell_file_size(self, msg: dict, addr: tuple, peer_socket):
    filename = msg["filename"]
    file_path = f"{config.directory.node_files_dir}{self.node_id}/{filename}"
    file_size = os.stat(file_path).st_size
    bitfield_pieces = next((item["bitfield_pieces"] for item in self.files if item["filename"] == filename), None)
    pieces_count = next((item["pieces_count"] for item in self.files if item["filename"] == filename), None)
    response_msg = Node2Node(src_node_id=self.node_id,
                             dest_node_id=msg["src_node_id"],
                             filename=filename,
                             size=file_size,
                             bitfield_pieces=bitfield_pieces,
                             pieces_count=pieces_count)
    # temp_port = generate_random_port()
    # temp_sock = set_socket(self.port, self.node_ip)
    # self.send_segment(sock=self.send_socket,
    #                   data=response_msg.encode(),
    #                   addr=addr)
    peer_socket.sendall(response_msg.encode())
    # free socket(temp sock)
```

- Và khi muốn gửi các chunk thì sử dụng hàm send\_chunk().

```
def send_chunk(self, filename: str, rng: tuple, dest_node_id: int, dest_port: int, peer_socket, piece_length: int):
    file_path = f"{config.directory.node_files_dir}{self.node_id}/{filename}"
    file_descriptor = os.open(file_path, os.O_RDONLY | os.O_CREAT)
    file_descriptor_position = rng[0] * piece_length + rng[1]

    # move the file descriptor to the desired location
    self.move_descriptor_position(file_descriptor_position, file_descriptor)
    block_data = self.read(file_descriptor, rng[2])
    msg = ChunkSharing(src_node_id=self.node_id,
                      dest_node_id=dest_node_id,
                      filename=filename,
                      range=rng,
                      chunk=block_data)
    peer_socket.sendall(Message.encode(msg))
    log_content = "The process of sending a chunk to node{} of file {} has finished!".format(dest_node_id, filename)
    log(node_id=self.node_id, content=log_content)
```

## 8 Chiến thuật tải

Mô tả tình huống:

- Node 1 có 2 file ban đầu là file A và file B trong torrent sample.torrent , có đầy đủ các chunk của từng file

```
"filename": "1_DatabaseSystemConceptsAndArchitecture.pdf",
"bitfield_pieces": [0, 1, 2, 3, 4, 5, 6],
"pieces_count": 7
```

Hình 4: Dữ liệu trong node 1

- Node 2 cũng có 2 file là file A và file B trong file sample.torrent, tuy nhiên file A node 2 có chunk là 4,5,6 và file B node 2 có đầy đủ các chunk

```
"filename": "1_DatabaseSystemConceptsAndArchitecture.pdf",  
"bitfield_pieces": [  
    5,  
    4,  
    6  
],  
"pieces_count": 7
```

Hình 5: Dữ liệu trong node 2

- Dưới đây là kết quả khi node 3 muốn tải file sample.torrent.

```
"filename": "1_DatabaseSystemConceptsAndArchitecture.pdf",  
"bitfield_pieces": [  
    2,  
    0,  
    1,  
    3,  
    4,  
    5,  
    6  
],  
"pieces_count": 7
```

Hình 6: Dữ liệu trong node 3

- Ta có thể thấy ở bitfield-pieces (là thứ tự các chunk mà node 3 tải về của file) ở file A theo thứ tự là 2,0,1,3 là những chunk mà chỉ node 1 có nhưng node 2 không có, tức giải thuật rearest-first đã hiện thực đúng bởi nó lấy những chunk mà ít node có nhất và sắp xếp lên đầu và request tới các node.

## 9 Mã nguồn của dự án

github: [<https://github.com/quanghia24/bittorrent-like-app.git>]