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

COMPUTER NETWORKING (CO3093)

ASSIGNMENT 1

DEVELOP A NETWORK APPLICATION

Advisor(s):	Le Bao Khanh	
Student(s):	Vuong Thanh Phuong (leader)	2252658
	Vo Nguyen Gia Huy	2252270
	Nguyen Thien Hai	2252189
	Le Hai Long	2252441

HO CHI MINH CITY, NOVEMBER 2024

Contents

1 Members and Workload

Full Name	Student Number	Task Assignment	% Work
Vuong Thanh Phuong	2252658		100%
Vo Nguyen Gia Huy	2252270		100%
Nguyen Thien Hai	2252189		100%
Le Hai Long	2252441		100%

Table 1.1: Member Task Assignment and Workload

2 Lý thuyết về BitTorrent

2.1 Định nghĩa

BitTorrent là một giao thức được thiết kế để tối ưu hóa việc chia sẻ và tải xuống tệp tin lớn thông qua mạng peer-to-peer, cho phép người dùng tải xuống tệp từ nhiều nguồn khác nhau đồng thời, từ đó tăng tốc độ tải và giảm tải cho các máy chủ trung tâm.

2.2 Bencode

Bencode là một dạng mã hóa được dùng cho việc truyền file peer-to-peer trong hệ thống Bittorent, hỗ trợ 4 loại:

- **Strings**
  Mã hóa theo dạng: *<string length>:<string>*
  Ví dụ: 4:spam tương ứng với chuỗi "spam"

- **Integers**
  Mã hóa theo dạng: **i***<integer>***e**
  Ví dụ: i3e tương ứng với số nguyên "3"

- **Lists**
  Mã hóa theo dạng: **l***<bencoded values>***e**
  Ví dụ: l4:spam4:eggse tương ứng với: ["spam", "eggs"]

- **Dictionaries**
  Mã hóa theo dạng: **d***<bencoded string><bencoded element>***e**
  Ví dụ: d4:spaml1:a1:bee tương ứng với "spam" => ["a", "b"]

2.3 Cấu trúc file Metainfo

Tất cả dữ liệu trong metainfo file đều được bencoded. Nội dung trong metainfo file là một từ điển bencoded, trong đó bao gồm các thành phần chính sau:
Info: một từ điển mô tả file (hoặc các file) của torrent.
Announce: URL thông báo của tracker (string).
Creation date: (optional) thời gian tạo torrent, theo định dạng epoch UNIX chuẩn.
Info Dictionary:
Piece length: Số byte trong mỗi mảnh (số nguyên).

Pieces: Chuỗi bao gồm sự nối tiếp của tất cả các giá trị băm SHA1 20-byte, mỗi mảnh một giá trị

2.4 Tracker HTTP/HTTPS protocol

Tracker là dịch vụ HTTP/HTTPS xử lý các yêu cầu GET từ client, cung cấp danh sách peers giúp client tham gia vào torrent. Phản hồi bao gồm một danh sách peers giúp client tham gia vào torrent. URL cơ bản bao gồm "announce URL" được định nghĩa trong tệp metainfo (.torrent).

2.4.1 Tracker Request Parameters

Khi một client gửi yêu cầu GET đến tracker, các tham số sử dụng bao gồm:
Info_hash: Mã băm SHA1 20-byte của giá trị khóa info trong tệp Metainfo, được mã hóa URL. Giá trị này là một từ điển được mã hóa bằng bencode.
Peer_id: Chuỗi 20-byte mã hóa URL được sử dụng như ID duy nhất cho client, được tạo ra khi client khởi động. Giá trị này có thể là bất kỳ giá trị nào, bao gồm cả dữ liệu nhị phân, nhưng cần phải duy nhất trên máy cục bộ.
Port: Số cổng mà client đang lắng nghe. Các cổng dành cho BitTorrent thường nằm trong khoảng từ 6881 đến 6889. Nếu không thể thiết lập cổng trong khoảng này, client có thể bỏ qua.
Uploaded: Tổng số dữ liệu đã tải lên (từ khi client gửi sự kiện 'started' đến tracker), được biểu diễn bằng số ASCII cơ bản. Đây là tổng số byte đã được tải lên.
Downloaded: Tổng số dữ liệu đã tải xuống (từ khi client gửi sự kiện 'started' đến tracker), cũng được biểu diễn bằng số ASCII cơ bản. Đây là tổng số byte đã được tải xuống.
Left: Số byte mà client vẫn cần tải xuống để hoàn thành (để có được tất cả các tệp trong torrent), được biểu diễn bằng số ASCII cơ bản.

2.4.2 Tracker Response

Tracker phản hồi bằng một tài liệu "text/plain" gồm một từ điển bencoded với các khóa sau:
Failure reason: Nếu có, không được có bất kỳ khóa nào khác. Giá trị là thông điệp lỗi dễ hiểu giải thích lý do yêu cầu thất bại (chuỗi).
Interval: Khoảng thời gian tính bằng giây mà client nên chờ giữa các yêu cầu gửi đến tracker.
Tracker ID: Chuỗi mà client nên gửi lại trong các thông báo tiếp theo. Nếu không có

và thông báo trước đó đã gửi một tracker ID, không được bỏ qua giá trị cũ; tiếp tục sử dụng nó.
Complete: Số lượng peers đã có toàn bộ tệp, tức là seeders (số nguyên).
Incomplete: Số lượng peers không phải seeders, còn gọi là "leechers" (số nguyên).
Peers: (dictionary mode) Giá trị là danh sách các từ điển, mỗi từ điển có các khóa sau:

- **Peer ID:** ID tự chọn của peer, như mô tả trong yêu cầu tracker (chuỗi).
- **IP:** Địa chỉ IP của peer, có thể là IPv6 hoặc IPv4 hoặc tên miền DNS.
- **Port:** Số cổng của peer (số nguyên).

2.5 Peer wire protocol (TCP)

Giao thức peer hỗ trợ việc trao đổi các mảnh như được mô tả trong tệp metainfo. Thuật ngữ "block" sẽ được sử dụng trong tài liệu này để mô tả dữ liệu được trao đổi giữa các peers.
Một client phải duy trì thông tin trạng thái cho mỗi kết nối với một peer từ xa:

- **choked:** Cho biết liệu peer từ xa có chặn client hay không. Khi peer chặn client, nó thông báo rằng không có yêu cầu nào sẽ được trả lời cho đến khi client được bỏ chặn.
- **interested:** Cho biết liệu peer từ xa có quan tâm đến những gì client cung cấp hay không. Điều này thông báo rằng peer từ xa sẽ bắt đầu yêu cầu các block khi client bỏ chặn chúng.

Data types: Tất cả các số nguyên trong giao thức peer được mã hóa dưới dạng giá trị 4 byte big-endian, bao gồm tiền tố độ dài trên tất cả các tin nhắn sau khi handshake.
Message flow: Giao thức peer bao gồm một lần handshake ban đầu. Sau đó, các peers giao tiếp qua việc trao đổi các tin nhắn có tiền tố độ dài.
Handshake: Handshake là tin nhắn bắt buộc và phải là tin nhắn đầu tiên do client truyền. Định dạng tin nhắn handshake là:
handshake: <pstrlen><pstr><reserved><info_hash><peer_id>

- **Pstrlen:** Độ dài của <pstr>.
- **Pstr:** Chuỗi định danh giao thức.
- **Reserved:** 8 byte dự trữ, hiện tại sử dụng toàn bộ là số 0.

- **Info_hash:** 20-byte SHA1 hash của khóa info trong tệp metainfo.
- **Peer_id:** 20-byte chuỗi ID duy nhất cho client.

Nếu client nhận được handshake với info_hash mà không khớp, client phải ngắt kết nối. Nếu peer_id không khớp, client khởi tạo cũng phải ngắt kết nối.
Peer_id: peer_id có độ dài chính xác 20 byte (ký tự).

3 Ứng dụng BitTorrent

3.1 Sơ lược

Ưu điểm:

- BitTorrent cho phép tải tệp từ nhiều nguồn cùng một lúc, giúp tăng tốc độ tải xuống so với các phương pháp tải truyền thống.
- Thay vì tải từ một máy chủ duy nhất, BitTorrent chia sẻ băng thông giữa nhiều người dùng, giảm áp lực lên các máy chủ.
- Người dùng không chỉ tải xuống mà còn có thể đóng vai trò như một nguồn cung cấp, giúp tăng tốc độ tải xuống cho người khác.

Nhược điểm:

- BitTorrent có thể bị lợi dụng để phát tán phần mềm độc hại hoặc vi phạm bản quyền, gây nguy cơ bảo mật cho người dùng.
- Tốc độ tải xuống phụ thuộc vào số lượng người dùng đang chia sẻ tệp tin. Nếu số lượng người chia sẻ ít, tốc độ tải xuống có thể chậm.
- Việc chia sẻ file có thể khiến người dùng bị lộ IP và port,dẫn đến dễ bị theo dõi,thông tin cá nhân có thể bị lộ.

3.2 Requirements

Functional Requirements:

a) Đối với Client:

- Người dùng có thể đăng nhập hoặc đăng ký tài khoản để sử dụng hệ thống.
- Người dùng có thể chia sẻ tệp tin mình có lên hệ thống.
- Người dùng có thể tìm kiếm tệp torrent trên hệ thống.
- Người dùng có thể tải tệp xuống từ nhiều nguồn khác nhau cùng 1 lúc.
- Người dùng có thể xem lịch sử các tệp đã tải.

b) Đối với Sever:

- Xác thực khi người dùng yêu cầu đăng nhập.
- Quản lý danh sách người dùng và tệp tin.
- Nhận và phản hồi lại các yêu cầu từ người dùng về các thông tin tệp và danh sách các peer thông qua giao thức HTTP.

Non-Functional Requirements:
Hiệu suất:

- Ứng dụng phải hỗ trợ tốc độ tải xuống và tải lên nhanh nhất có thể, phụ thuộc vào băng thông mạng của người dùng và số lượng peer sẵn có.
- Ứng dụng phải có khả năng xử lý tối thiểu 50 kết nối peer đồng thời mà không làm giảm hiệu suất đáng kể.

Độ tin cậy:

- Ứng dụng phải có khả năng tự động khôi phục các kết nối bị mất với các peer nếu có sự cố mạng xảy ra.
- Dữ liệu tải xuống phải được kiểm tra tính toàn vẹn, và các phần bị lỗi phải được yêu cầu tải lại tự động.

Bảo mật:

- Phải có biện pháp kiểm tra và xác minh mã băm (hash) để đảm bảo tính toàn vẹn của dữ liệu tải về.
- Dữ liệu người dùng và tệp tin cần được bảo vệ.

Khả năng mở rộng:

- Kiến trúc hệ thống phải cho phép thêm các tính năng mới mà không ảnh hưởng đến các chức năng hiện có.
- Hệ thống phải có khả năng mở rộng để xử lý một lượng người dùng và tệp tin lớn mà không làm giảm hiệu suất.

Tính khả dụng:

- Giao diện người dùng cần phải thân thiện và dễ sử dụng, giúp người dùng dễ dàng tìm kiếm, upload và download tệp.
- Cần có phần hướng dẫn hoặc tài liệu hỗ trợ đi kèm để người dùng dễ dàng làm quen với các tính năng của ứng dụng.
- Ứng dụng phải cung cấp các thông báo rõ ràng cho người dùng khi có lỗi xảy ra hoặc khi cần sự can thiệp từ người dùng.

Khả năng duy trì: Hệ thống cần được thiết kế để dễ dàng bảo trì và cập nhật.

3.3 Use-case Diagram

a) Toàn bộ hệ thống

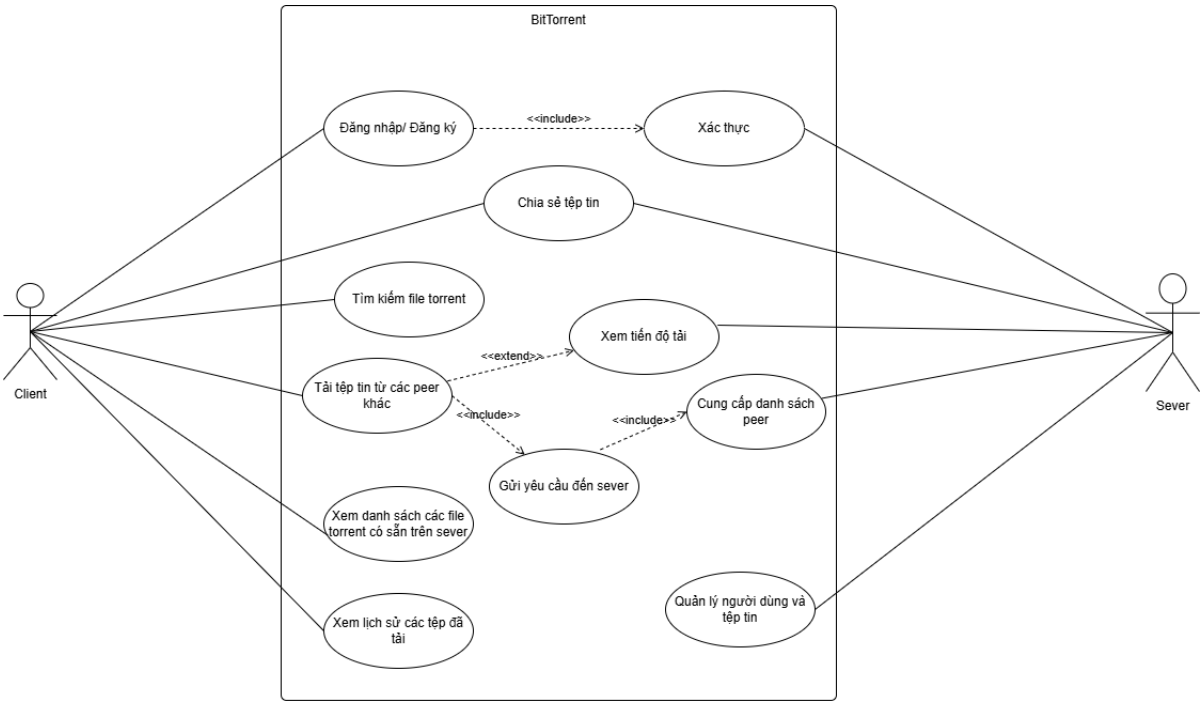


Figure 3.1: Use-case hệ thống

b) Module tải file

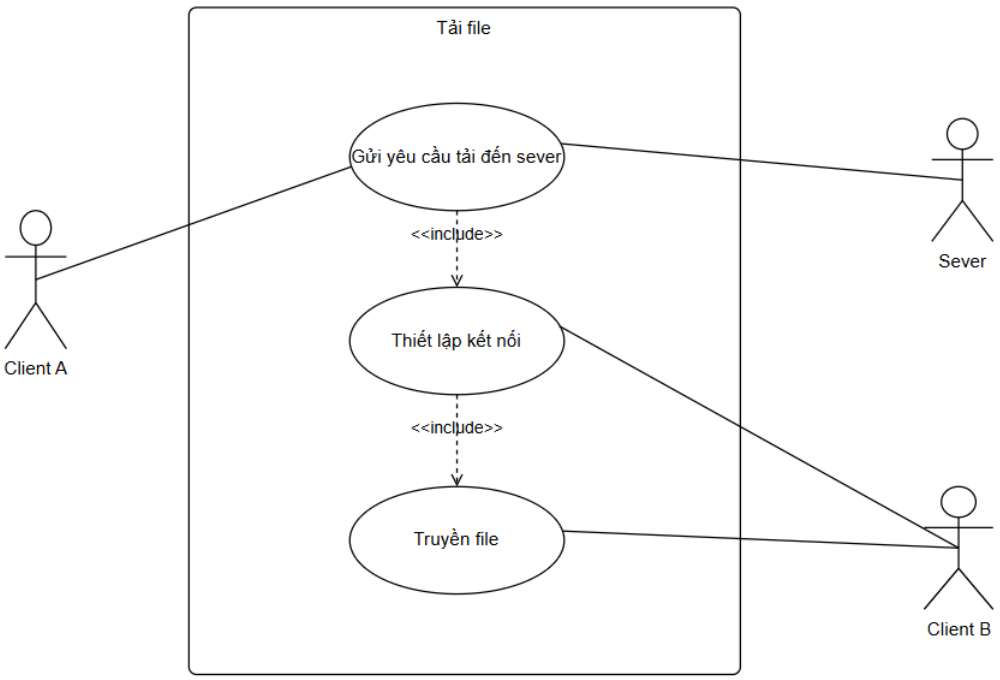


Figure 3.2: Use-case module tải file

Description:

Table 3.1: Usecase table cho module tải file

Use Case	**Description**
Actor	Client A, B và Server
Brief Description	Client cần thực hiện việc trao đổi file thông qua mạng P2P
Preconditions	Người dùng tải file đăng nhập vào hệ thống, người truyền file đã kết nối với Tracker
Postconditions	Người dùng tải file thành công và lưu lịch sử vào hệ thống
Continued on next page	

Table 3.1 – continued from previous page

Use Case	Description
Trigger	Client A yêu cầu tải file
Normal Flow	1. Client A gửi yêu cầu tải file thông qua file `.torrent`. 2. Server nhận tín hiệu, thông qua mã hóa file `.torrent`, tìm kiếm các Client chia sẻ file và thông báo cho client cần tải. 3. Client cần tải file gửi tín hiệu thiết lập kết nội TCP tới các Client chia sẻ. 4. Client truyền file nhận tín hiệu, phản hồi và hai bên tiến hành thiết lập tải file từng pieces theo yêu cầu của bên tải.
Exceptional flow	Ở bước **2** nếu không có Client nào đang share file cần tải, thông báo cho người dùng và ngừng chu trình xử lý
Alternative flow	Ở bước **4** nếu không có tín hiệu phản hồi sau một thời gian, toàn bộ tiến trình tải sẽ bị hủy bỏ.

3.4 Phác họa kiến trúc ứng dụng

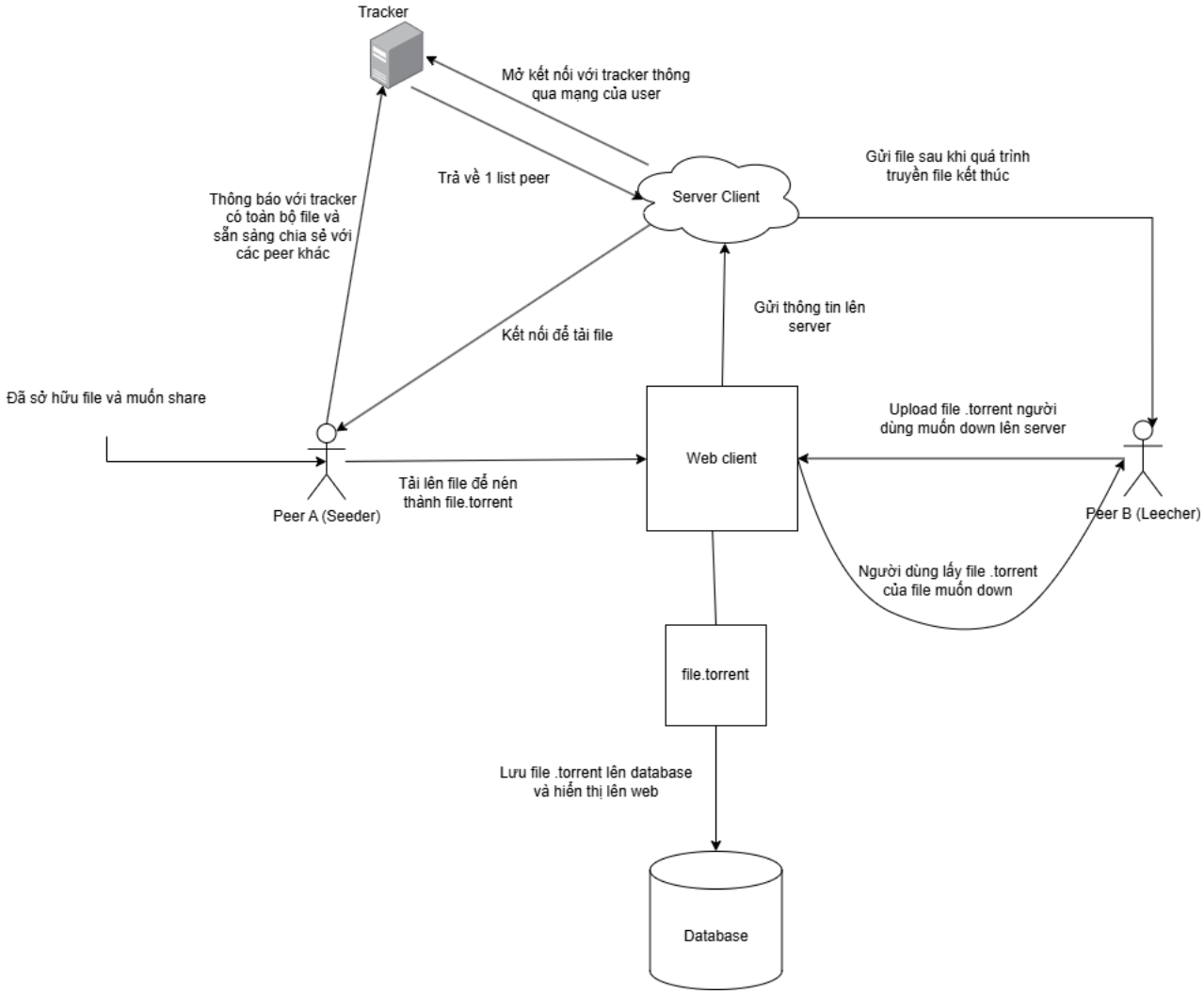


Figure 3.3: Kiến trúc phần mềm

3.5 Các giao thức đã sử dụng

Giao thức TCP: là một giao thức truyền tải dữ liệu quan trọng trong mạng Internet, hoạt động dựa trên mô hình kết nối (connection-oriented). Khi hai máy tính kết nối qua TCP, chúng thiết lập một phiên giao tiếp, trong đó dữ liệu được chia thành các gói tin và gửi tuần tự từ nguồn đến đích. TCP đảm bảo tính toàn vẹn của dữ liệu thông qua cơ chế kiểm tra lỗi và xác nhận (acknowledgment), do đó, nếu gói tin bị mất hoặc hỏng trong quá trình truyền, chúng sẽ được gửi lại.

Trong code, giao thức TCP được sử dụng làm phương tiện giao tiếp giữa các peer trong

quá trình tải xuống dữ liệu từ một torrent.

```
const client = new net.Socket();
let keepAliveInterval;
let overallTimeout;

client.connect(peer.port, peer.ip, () => {
  console.log(`Connected to peer ${peer.ip}:${peer.port}`);
  const handshakeMsg = createHandshake(infoHash, peerId.toString("hex"));
  client.write(handshakeMsg);


  keepAliveInterval = setInterval(() => {
    client.write(Buffer.alloc(4));
  }, 120000);
});
```

Giao thức HTTP: HTTP là giao thức dùng để truyền tải các tài liệu siêu văn bản như trang web từ máy chủ đến trình duyệt của người dùng, hoạt động trên nền tảng TCP. HTTP không có cơ chế bảo mật, nên dữ liệu truyền đi không được mã hóa và có thể bị theo dõi hoặc đánh cắp bởi bên thứ ba. Giao thức này hoạt động dựa trên mô hình yêu cầu-phản hồi, trong đó trình duyệt gửi yêu cầu và máy chủ đáp ứng lại với dữ liệu phù hợp.
Trong code,giao thức HTTP được sử dụng để kết nối giữa các peer với tracker (peer discovery).

```
const server = http.createServer((req, res) => {
  if (req.method === "GET") {
    const parsedUrl = url.parse(req.url, true);
    handleAnnounceRequest(parsedUrl, req, res);
  } else {
    res.writeHead(405);
    res.end("Method Not Allowed");
  }
});
```

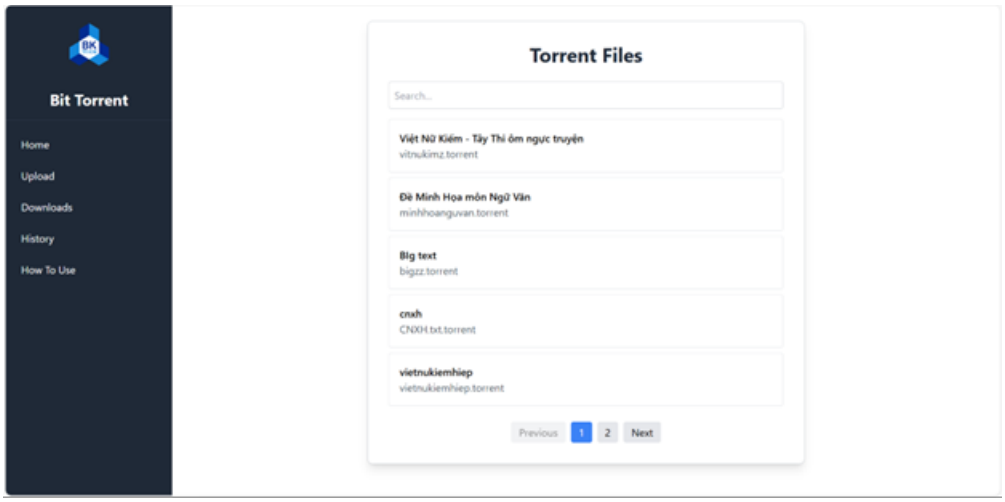
```
http
  .get(
    `${trackerUrl.origin}${trackerUrl.pathname}?${params.toString()}`,
    (res) => {
      let data = [];
      res.on("data", (chunk) => data.push(chunk));
      res.on("end", () => {
        const response = bencode.decode(Buffer.concat(data));
        if (response["failure reason"]) {
          console.error(
            "Tracker announce failed:",
            response["failure reason"].toString()
          );
        } else {
          console.log("Announced to tracker successfully ");
        }
      });
    }
  )
  .on("error", (err) => {
    console.error("Error announcing to tracker:", err.message);
  });
}
```

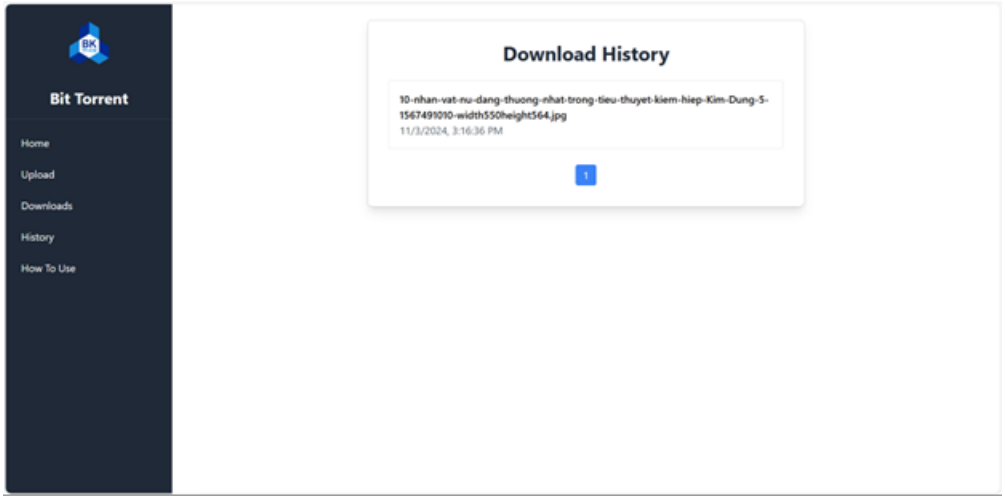
HTTPS: HTTPS là phiên bản bảo mật của HTTP, sử dụng SSL/TLS để mã hóa dữ liệu được truyền đi, giúp bảo vệ thông tin nhạy cảm như mật khẩu và thông tin tài chính khỏi bị đánh cắp. Giao thức này vẫn hoạt động trên nền tảng TCP nhưng có thêm một lớp mã hóa, do đó dữ liệu được mã hóa từ đầu đến cuối, chỉ người gửi và người nhận mới có thể giải mã. HTTPS được coi là chuẩn mực cho các trang web yêu cầu bảo mật, như các trang thanh toán trực tuyến, ngân hàng, và các nền tảng giao tiếp chứa dữ liệu cá nhân.
Giao thức HTTPS được sử dụng trong server backend Node.js. Vì dữ liệu truyền được mã hóa SSL/TLS nên sẽ tăng tính bảo mật của ứng dụng

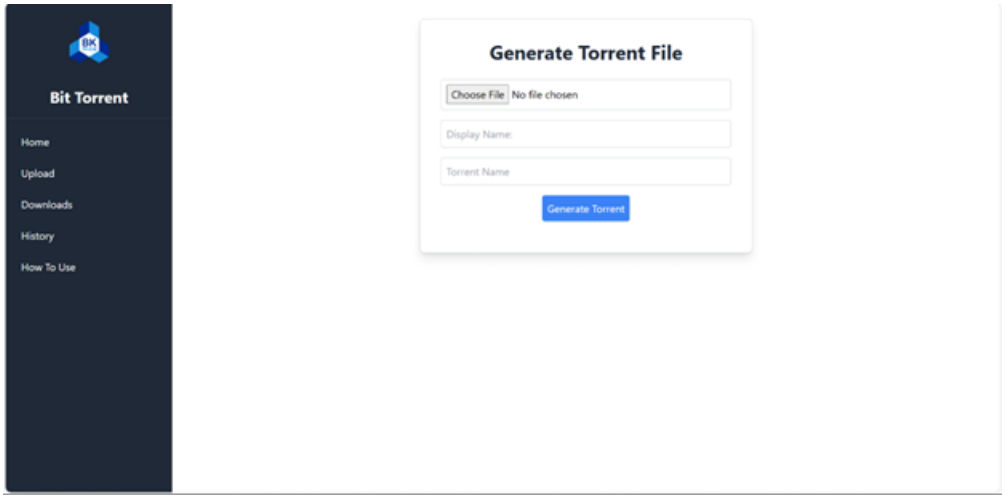
3.6 Thành phần cốt lõi

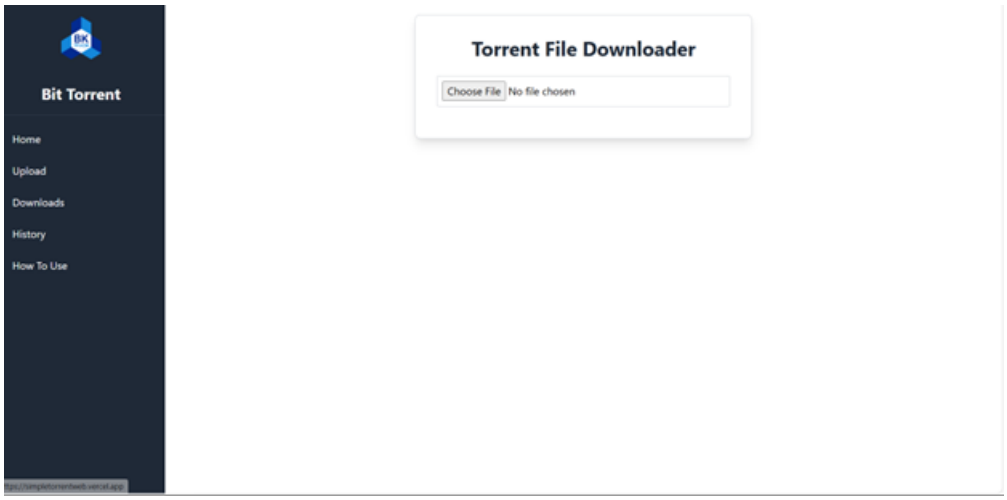
3.6.1 Client

Link deploy website: https://simpletorrentweb.vercel.app
User interface: Sử dụng Reactjs để code giao diện người dùng cho ứng dụng. Bao gồm các trang: đăng ký, đăng nhập, tải xuống file, danh sách file .torrent, lịch sử tải của tài khoản, upload file.


Bit Torrent
Home
Upload
Downloads
History
How To Use
Torrent Files
Search...
Việt Nữ Kiếm - Tây Thi ôm ngực truyện
vitnukimz.torrent
Đề Minh Họa môn Ngữ Văn
minhhoanguvan.torrent
Big text
bigzz.torrent
cnxh
CNXH.txt.torrent
vietnukiemhiep
vietnukiemhiep.torrent
Previous
1
2
Next


Bit Torrent
Home
Upload
Downloads
History
How To Use
Download History
10-nhan-vat-nu-dang-thuong-nhat-trong-tieu-thuyet-kiem-hiep-Kim-Dung-5-1567491010-width550height564.jpg
11/3/2024, 3:16:36 PM
1


Bit Torrent
Home
Upload
Downloads
History
How To Use
Generate Torrent File
Choose File
No file chosen
Display Name
Torrent Name
Generate Torrent



Search: Các file .metainfo được upload lên server sẽ hiển thị ở Homepage. Người dùng có thể download các file bằng cách click vào file đó. Thanh search bar dùng để tìm kiếm trong database.

History: Lưu trữ các file người dùng đã download.

3.6.2 Server

Authentication: Người dùng gửi thông tin tài khoản lên server. Sau khi xác thực với database, server gửi token cho phía người dùng.

```
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem("token");
    const expiry = localStorage.getItem("expiry");

    if (token && expiry) {
      if (new Date().getTime() < parseInt(expiry)) {
        setUser({ token });
      } else {
        // Clear expired token
        localStorage.removeItem("token");
        localStorage.removeItem("expiry");
      }
    }
    setLoading(false);
  }, []);
```

Token được lưu như một context. Để có thể tương tác với ứng dụng, người dùng cần có token hợp lệ. Sau 15 phút, token sẽ bị xóa, kết thúc một phiên đăng nhập. Lúc này hệ thống sẽ yêu cầu người dùng đăng nhập lại.

```
app.post("/download-file", authenticateToken, async (req, res) => {
  const { torrentFile } = req.body;
  console.log("Downloading file from torrent...");

  try {
    const tempDir = "/tmp";
    const { fileName, filePath } = await downloadFileCommand(
      torrentFile,
      tempDir
    );
```

Tất cả các hàm đều cần có token để hoạt động.

Tracker: Tracker được host trên VPS, nên luôn hoạt động.

1. Giúp các peer tìm thấy nhau khi tham gia vào một "swarm".
2. Nhận thông tin kết nối. Chuyển đổi thành info_hash để xác nhận yêu cầu của peer đó. Từ đó tiến hành tìm các peer có file yêu cầu.

```
function handleAnnounceRequest(parsedUrl, req, res) {
  const { pathname, query } = parsedUrl;

  const info_hash = Buffer.from(query.info_hash, "binary").toString("hex");
  const peer_id = Buffer.from(query.peer_id, "binary").toString("hex");

  const port = parseInt(query.port, 10);
  const ip = req.socket.remoteAddress.replace("::ffff:", "");

  if (pathname === "/announce" && info_hash && peer_id && port) {
    addPeer(info_hash, { peer_id, ip, port, lastSeen: Date.now() });

    const peerList = getPeers(info_hash);
    const compactPeerList = compactPeers(peerList);

    const response = {
      interval: 1800,
      peers: compactPeerList,
    };

    console.log(`Announced peer: ${ip}:${port} for info_hash: ${info_hash}`);
    res.writeHead(200, { "Content-Type": "application/x-bittorrent" });
    res.end(bencode.encode(response));
  } else {
    res.writeHead(400);
    res.end(
      bencode.encode({ "failure reason": "Missing required parameters" })
    );
```

Hàm getTrackerPeers: trả về danh sách các peer có chứa file cần thiết

```
async function getTrackerPeers(trackerURL, infoHash, fileLength, peerId) {
  const publicIP = await getPublicIP(); // Fetch the public IP

  return new Promise((resolve, reject) => {
    const params = new URLSearchParams({
      info_hash: infoHash,
      peer_id: peerId,
      port: 6881,
      uploaded: 0,
      downloaded: 0,
      left: fileLength,
      compact: 1,
    });
    const url = `${trackerURL}?${params}`;

    http
```

Download:

Xây dựng class WorkQueue quản lý trạng thái tải của từng mảnh dữ liệu, đánh dấu mảnh nào đã tải, đang tải, hay thất bại.

pendingPieces: Danh sách các mảnh chưa tải.

inProgressPieces: Các mảnh đang được tải.

completedPieces: Các mảnh đã tải xong.

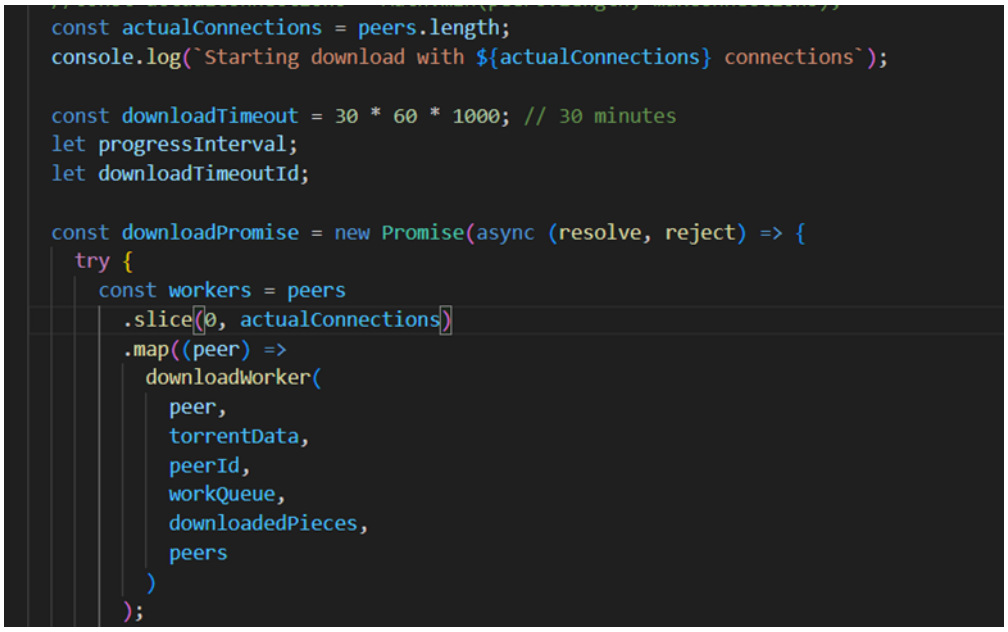
```
class WorkQueue {
  constructor(totalPieces) {
    this.pendingPieces = new Set([...Array(totalPieces).keys()]);
    this.inProgressPieces = new Set();
    this.completedPieces = new Set();
    this.totalPieces = totalPieces;
    this.lastProgressUpdate = Date.now();
  }

  getNextPiece() {
    console.log(`Current state - Pending: ${[...this.pendingPieces].join(',')}, InProgress:

    const now = Date.now();
    if (now - this.lastProgressUpdate > 60000) { // 1 minute timeout
      console.warn("Detected stalled pieces, returning them to pending queue");
      for (const piece of this.inProgressPieces) {
        this.pendingPieces.add(piece);
```

Figure 3.4: Class WorkQueue

Khi bắt đầu tải file,giải mã file torrent để lấy các thông tin cần thiết. Khởi tạo **WorkQueue** và đẩy các piece cần tải vào **pendingPieces**. Gửi yêu cầu tới tracker để lấy danh các peer đang chia sẻ file client cần.



```
const actualConnections = peers.length;
console.log(`Starting download with ${actualConnections} connections`);

const downloadTimeout = 30 * 60 * 1000; // 30 minutes
let progressInterval;
let downloadTimeoutId;

const downloadPromise = new Promise(async (resolve, reject) => {
  try {
    const workers = peers
      .slice(0, actualConnections)
      .map((peer) =>
        downloadWorker(
          peer,
          torrentData,
          peerId,
          workQueue,
          downloadedPieces,
          peers
        )
      );
```

Figure 3.5: Hàm downloadFile

Khởi động các "worker" (mỗi worker tương ứng 1 peer) để tải các mảnh song song từ nhiều peer. Dẩy các mảnh đang tải vào **inprogressPieces** để quản lý trạng thái.

```
async function downloadFile(torrentFile, outputPath) {
  const fileContent = readFile(torrentFile);
  const torrentData = bencode.decode(fileContent);
  const fileLength = torrentData.info.length;
  const pieceLength = torrentData.info["piece length"];
  const totalPieces = Math.ceil(fileLength / pieceLength);

  console.log(`Starting download of ${totalPieces} pieces`);

  const workQueue = new WorkQueue(totalPieces);
  const downloadedPieces = new Map();

  const infoHash = calculateInfoHash(torrentData.info);
  const peerId = generatePeerId();
  const trackerURL = String(torrentData.announce);

  const peers = await getTrackerPeers(trackerURL, infoHash, fileLength, peerId);
  if (peers.length === 0) throw new Error("No peers available");
```

Khi đã nhận đủ dữ liệu của mảnh, xác thực mảnh bằng cách so sánh hash của dữ liệu với hash mong đợi từ torrent. Nếu khớp, lưu mảnh vào **completedPieces** và kết thúc.Nếu thất bại,đẩy mảnh vào **pendingPieces** và thử lại sau.

Seed:

1. Đọc file metainfo và tạo kết nối tới tracker:

```
async function announceToTracker(torrentData, port, peerId) {
  const infoHash = calculateInfoHash(torrentData.info);
  const trackerUrl = new URL(torrentData.announce);
  const publicIP = await getPublicIP(); // Fetch the public IP

  const params = new URLSearchParams({
    info_hash: infoHash.toString("hex"),
    peer_id: peerId.toString("hex"),
    port: port,
    uploaded: 0,
    downloaded: 0,
    left: torrentData.info.length,
    compact: 1,
    event: "started",
    ip: publicIP,
  });

  http
    .get(
      `${trackerUrl.origin}${trackerUrl.pathname}?${params.toString()}`,
      (res) => {
        let data = [];
        res.on("data", (chunk) => data.push(chunk));
        res.on("end", () => {
          const response = bencode.decode(Buffer.concat(data));
          if (response["failure reason"]) {
            console.error(
              "Tracker announce failed:",
              response["failure reason"].toString()
            );
          } else {
            console.log("Announced to tracker successfully ");
          }
        });
      }
    )
    .on("error", (err) => {
      console.error("Error announcing to tracker:", err.message);
    });
}
```

2. Chờ tín hiệu từ các peer muốn tải. Khi nhận được message, lần lượt gửi tín hiệu *handshake*, sau đó sẽ là *bitfield*. Kế tiếp, seeder gửi tín hiệu unchoke và bắt đầu nhận danh sách yêu cầu các pieces từ peer.

```
function sendPieceMessage(socket, pieceIndex, begin, pieceData) {
  const messageLength = 9 + pieceData.length;
  const pieceMessage = Buffer.alloc(4 + messageLength);

  pieceMessage.writeUInt32BE(messageLength, 0);
  pieceMessage.writeUInt8(7, 4);
  pieceMessage.writeUInt32BE(pieceIndex, 5);
  pieceMessage.writeUInt32BE(begin, 9);
  pieceData.copy(pieceMessage, 13);

  socket.write(pieceMessage);
  console.log(
    `Sent piece message for piece ${pieceIndex}, begin=${begin}, length=${pieceData.length}`
  );
}
```

Upload:

1. User tải file cần nén lên server thông qua interface

2. Server xử lí file nhận được, tiếp theo gọi hàm **generateTorrent**.

```
app.post("/generate-torrent", authenticateToken, async (req, res) => {
  try {
    const chunks = [];
    req.on("data", (chunk) => chunks.push(chunk));

    req.on("end", async () => {
      const bodyBuffer = Buffer.concat(chunks);
      const boundary = `--${req.headers["content-type"].split("boundary=")[1]}`;

      let fileBuffer = null,
        torrentName = null,
        displayName = null,
        originalFileName = null;
      const parts = [];
      let start = 0;

      // Parse multipart form data
      while ((start = bodyBuffer.indexOf(boundary, start)) !== -1) {
        const end = bodyBuffer.indexOf(boundary, start + boundary.length);
        if (end === -1) break;
        parts.push(bodyBuffer.slice(start + boundary.length + 2, end - 2));
        start = end;
      }

      parts.forEach((part) => {
        const headerEnd = part.indexOf("\r\n\r\n");
        const headers = part.slice(0, headerEnd).toString();

        const contentDisposition = headers.match(
          /Content-Disposition: form-data; name="([^"]+)"(?:; filename="([^"]+)")?/
        );
        if (contentDisposition) {
          const fieldName = contentDisposition[1];
          const filename = contentDisposition[2];

          if (fieldName === "file") {
            fileBuffer = part.slice(headerEnd + 4);
            if (filename) {
              originalFileName = filename;
            } else {
              originalFileName = "uploaded_file";
            }
          } else if (fieldName === "torrentName") {
            torrentName = part
              .slice(headerEnd + 4)
              .toString("utf8")
              .trim();
```

3. Hàm xử lý sẽ phân chia các data dưới dạng Bufer thành các chunk (hay pieces) với kích thước tối đa của mỗi piece sẽ là 512KB.

3.6.3 Extended Function

a) **Database:** Sử dụng noSQL database Mongodb. Các thông tin được lưu trữ bao gồm:

- User: thông tin người dùng để xác thực.
- Các file torrent: lưu trữ các file .torrent được upload lên server từ người dùng. Người dùng khác có thể download những file này.
- Download history: Lịch sử download của một người dùng, được phân biệt bằng khóa userId.

b **Thiết lập thời gian chờ**:
`Client.setTimeout(60000)`: Thiết lập thời gian chờ cho kết nối socket (TCP). Nếu

không có hoạt động trong vòng 60 giây, kết nối sẽ bị hủy và thông báo lỗi *Connection timed out* sẽ được gửi.

```
client.setTimeout(60000);
```

Figure 3.6: Client.setTimeout(60000)

c) **Retry logic**: Nếu một peer không tải thành công mảnh, mã sẽ chuyển sang peer tiếp theo trong danh sách và thử lại.

```
  }
} catch (error) {
  console.error(`Error downloading piece ${pieceIndex} from peer ${currentPeer.ip}:${currentPeer.port}:`, error);


  currentPeer = getNextPeer();
  console.log(`Switching to peer ${currentPeer.ip}:${currentPeer.port}`);

  maxRetriesPerPiece.set(pieceIndex, retryCount + 1);
  await new Promise(resolve => setTimeout(resolve, 1000 * (retryCount + 1)));
}
```

Với mỗi mảnh, mã chỉ thử lại tối đa maxRetries lần. Nếu sau số lần thử này mà vẫn không tải xong, mã sẽ đánh dấu mảnh đó là thất bại và bỏ qua để thử lại sau.

```
const retryCount = maxRetriesPerPiece.get(pieceIndex) || 0;
let currentPeer = peer;
let success = false;

while (!success && retryCount < maxRetries) {
  try {

  }

  if (!success) {
    console.error(`Failed to download piece ${pieceIndex} after trying all peers`);
    workQueue.markPieceFailed(pieceIndex);
  }
}
```

d) **Kiểm tra khi gửi yêu cầu tải piece**: Trước khi gửi yêu cầu tải piece,cần kiểm tra để đảm bảo rằng không gửi yêu cầu tải piece mà mình đã có hoặc piece mà peer kết nối không có.

```
function requestPieces() {

  function hasPiece(index) {
    if (peerBitfield.length === 0) return true;
    const byteIndex = Math.floor(index / 8);
    const bitIndex = 7 - (index % 8);
    return byteIndex < peerBitfield.length &&
           (peerBitfield[byteIndex] & (1 << bitIndex)) !== 0;
  }

  if (!hasPiece(pieceIndex)) {
    console.log(`Peer doesn't have piece ${pieceIndex}, skipping requests`);
    return;
  }

  if (workQueue.completedPieces.has(pieceIndex)) {
    console.log(`Piece ${pieceIndex} already downloaded, skipping requests`);
    return;
  }
```

e) **Phát hiện tình trạng stalled (bị kẹt)**: Nếu một mảnh đang tải bị kẹt quá lâu mà không tải xong, WorkQueue sẽ chuyển mảnh đó trở lại danh sách pending để thử lại với peer khác. Mã kiểm tra stalled bằng cách xem thời gian tải so với thời gian cập nhật gần nhất của WorkQueue.

```
if (now - this.lastProgressUpdate > 60000) {
  console.warn("Detected stalled pieces, returning them to pending queue");
  for (const piece of this.inProgressPieces) {
    this.pendingPieces.add(piece);
  }
  this.inProgressPieces.clear();
}
```

f) **Kết nối giữa tracker với seeder:** Tracker được thiết lập để xóa các peer không active sau 15 phút. Cứ sau 10 phút, các seeder cần phải gửi tín hiệu keep alive đến tracker để xác nhận kết nối.

g) **Bảo mật:** Người dùng có thể download nhiều file cùng một lúc. Nhưng do giới hạn của server host vercel miễn phí nên việc tải các file lớn còn hạn chế.

h) **Bảo mật:** Như đã đề cập đến, để sử dụng được hệ thống, người dùng cần phải có token đang hoạt động để tương tác được với mọi chức năng của ứng dụng.

References

[1] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 8th edition, 2021.