Describe each specific function of the application

Bất kỳ một loại ứng dụng mạng máy tính nào cũng cần có những cơ chế để người tham gia vào mạng có thể tương tác với nhau và với hệ thống để thu được kết quả mong muốn. Không ngoại lệ, ứng dụng mạng Peer-to-Peer mà nhóm hiện thực để chia sẻ tài nguyên thông qua giao thức BitTorrent để có thể truyền gửi và tải dữ liệu đồng thời từ nhiều peers khác nhau cũng cần hiện thực những cơ chế để người dùng có thể dễ dàng chia sẻ dữ liệu cho nhau.

Người dùng (client) có thể sử dụng giao diện dòng lệnh để giao tiếp với hệ thống đóng vai trò là một peer trong mạng P2P nhằm thực hiện một số tác vụ dùng trao đổi dữ liệu giữa các peers trong mạng. Một số chức năng hữu dụng mà client có thể yêu cầu như:

Download: mỗi peer trong mạng đều có thể sử dụng lệnh download để tải các mảnh của một file hoặc một thư mục chứa nhiều file từ phía các peer khác về máy tính của mình. Các mảnh sau đó được tập hợp và lắp ráp theo đúng thứ tự để thu được file hoàn chỉnh. Đối số đầu vào tương ứng của lệnh download là đường dẫn đến file torrent, cổng (port) mà client muốn dùng để thực hiện việc tải file, nếu không được chỉ định hệ thống sẽ dùng cổng mặc định là 6881, đường dẫn đến vị trí của file dùng để lưu dữ liệu được tải về.

Seed: đây là lệnh được dùng khi một peer muốn chia sẻ các mảnh đã có của một file đang được chia sẻ trên mạng P2P này. Đối số đầu vào của lệnh seed là đường dẫn đến file torrent, đường dẫn đến vị trí của file muốn chia sẻ, cổng mà client muốn dùng để thực hiện quá trình chia sẻ, cổng mặc định là 6882.

Peers: dùng để gửi yêu cầu scrape tới tracker để lấy thông tin về số lượng peers hiện tại đang tham gia trên mạng Bittorent và sau đó in thông số và trạng thái của các peers ra màn hình của client theo một định dạng nhất định. Đối số của lệnh này là

Stop: khi client dùng lệnh stop, đồng nghĩa với việc client không muốn tiếp nhận bất kỳ yêu cầu chia sẻ file nào nữa. Lúc này peer sẽ ngừng chia sẻ file và sẽ ngắt kết nối với các peers khác. Đối số đầu vào của lệnh stop là đường dẫn đến file torrent của file đang được chia sẻ.

Create: dùng để tạo file torrent cho file cần chia sẻ. Để tạo file torrent ta cần những đối số sau: đường dẫn đến file hay thư mục chứa dữ liệu muốn chia sẻ, địa chỉ của tracker, đường dẫn đến file torrent được tạo ra, độ dài mỗi mảnh file được cắt ra từ file dữ liệu gốc (mặc định là 512 KB).

Exit: lệnh dùng để thoát khỏi giao diện dòng lệnh mà client dùng để giao tiếp với hệ thống.

- Không chỉ phía client mới cần những câu lệnh để tương tác trực tiếp với hệ thống, ngay cả tracker cũng phải có một số câu lệnh để cập nhật trạng thái của các peers để hỗ trợ quá trình truyền tải dữ liệu trong mạng.

Ping: dùng để gửi một thông điệp đến tất cả các peers có trong danh sách các peers đang hoạt động trên mạng nhằm xác nhận lại trạng thái hiện thời của các peers. Khi đã biết được peer nào không còn hoạt động trên mạng (offline), tracker sẽ loại peer đó ra khỏi danh sách những peers đang hoạt động mà tracker quản lý.

Detailed application design (architecture, class diagrams, main classes ...)

Cấu trúc của ứng dụng mạng này của nhóm được chia thành 3 phần chính: Client, P2P và Tracker. Mỗi phần được nhóm hiện thực một thư mục riêng đóng một vai trò nhất định trong ứng dụng mạng này. Sau đây là cấu trúc và chức năng cụ thể cho mỗi phần:

**Thư mục Client chứa 2 file là client\_c8li.py và client\_node.py. Thư mục này chứa code hiện thực cho hoạt động của một peer trong mạng và cách thức để người dùng giao tiếp với hệ thống.**

- File client\_node.py hiện thực các chức năng của một peer trong mạng như đã trình bày ở phần trước. Trong file này chứa một lớp là ClientNode, lớp này đại diện cho một đối tượng peer trong mạng và có những thuộc tính và phương thức đặc trưng cho một peer đển hiện thức các chức năng nói trên. Các phương thức của ClientNode bao gồm:

    def \_\_init\_\_(self):

        self.tracker\_url = None

        self.tracker\_id = None

        self.torrent\_file = None

        self.torrent\_data = None

        self.has\_announced = False  # Track if the client has announced to the tracker

        self.peer\_id = \_generate\_peer\_id()

        self.download\_port = 6881

        self.upload\_port = 6882

        self.announce\_port = 6883

        self.ping\_port = 6884

        self.downloadding\_manager = None

        self.uploading\_manager = None

        self.stop\_event = threading.Event()  # Event to signal the server thread to stop

        self.seeding\_files = {}  # Dictionary to store seeding files info

        self.announced\_trackers = set()  # Set to store announced trackers

        self.ping\_thread = threading.Thread(target=self.start\_ping\_server)

        self.ping\_thread.start()

Phương thức \_\_int\_\_ để khai báo các biến của đối tượng ClientNode

    def start\_ping\_server(self):

Phương thức \_load\_torrent\_file được gọi trong \_\_int\_\_ để tạo một kênh giao tiếp giữa peer và tracker, luôn lắng nghe và phản hồi khi có thông báo ping từ tracker

def \_load\_torrent\_file(self, torrent\_file):

Phương thức \_load\_torrent\_file dùng để trích xuất nội dung trong file torrent của đối số đầu vào thành các trường và dữ liệu tương ứng của trường đó. Sau đó, lưu nội dung vào biến kiểu dictory với key là tên trường và value là dữ liệu của trường đó, sau đó trả về biến kiểu dictory để tiện sử dụng cho mục đích khác.

    def announce(self, info\_hash, port, event='started'):

Phương thức annouce hiện thực việc peer gửi thông báo cho tracker về trạng thái hiện tại của mình để tracker biết peer đang muốn gì. Tùy theo trạng thái của hiện tại của peer mà tracker sẽ trả về những nội dung gì. Nếu như peer đang muốn tải file thì tracker sẽ trả về chuỗi byte chứa địa chỉ của các peers đang giữ file mà cần tải, sau đó annouce sẽ biến đổi và trả về danh sách chứa địa chỉ các peers đó.

def \_parse\_compact\_peers(self, peers):

Phương thức \_parse\_compact\_peers phân tích chuỗi byte chứa địa chỉ của các peers được gửi về từ tracker thành danh sách chứa nội dung tương đương và trả về danh sách này.

    def download\_torrent(self, torrent\_file, port=None, download\_dir=None):

Phương thức download\_torrent là hiện thực cụ thể cho lệnh download để xử lý quá trình tải về file dữ liệu gốc.

    def seed\_torrent(self, torrent\_file, complete\_file, port=None, upload\_rate=None):

    def \_start\_seeding\_server(self, port):

Hai phương thức seed\_torrent và \_start\_seeding\_server dùng để xứ lý quá trình chia sẻ file. seed\_torrent giúp tính toán các giá trị cần cần thiết và khởi tạo các biến cần trong quá trình chia sẻ file, còn \_start\_seeding\_server lắng nghe các yêu cầu chia sẻ file từ các peer khác và xử lý cách yêu cầu đó.

    def scrape(self, info\_hash):

    def scrape\_peers(self, torrent\_file):

Hai phương thức trên đây dùng để hiện thực cho lệnh peers trong giao diện dòng lệnh với chức năng đã nêu trên

    def sign\_out(self):

Phương thức sign\_out để client thông báo even “stopped” mà không cần mã info\_hash với tracker khi thoát chương trình để tracker ngầm hiểu và xóa thông tin về client khỏi danh sách client của mọi info\_hash.

- File “client\_li.py” là nơi để chạy code để peer hoạt động và chứa hiện thực giao diện dòng lệnh cho ứng dụng mạng. Trong file này, một biến kiểu ClientNode đại diện cho peer đó được khởi tạo để tiến hành chạy các lệnh thông qua các phương thức của đối tượng ClientNode.

**Thư mục P2P chứa hiện thực các lớp quản lý giao tiếp, các lớp xử lý trực tiếp cho quá trình truyền và nhận dữ liệu giữa các peer trong mạng. Thư mục gồm 8 file là bitfield.py, handshake.py, message.py, piece.py, peer.py, peer\_communication.py, upload\_manager.py, download\_manager.py.**

- File bitfield.py chứa hiện thực class BitField cho một đối tượng bitfield, được dùng để lưu một chuỗi bit (kiểu byte) mà mỗi bit đại diện cho một mảnh của file. Nếu như peer đã có mảnh file đó thì bit tương ứng sẽ có giá trị 1 (ngược lại sẽ có giá trị 0). Bitfield được dùng để xác định số mảnh mà một peer đang có, các peers trong mạng có thể trao đổi bitfield để biết được peer kia đang có peer nào và có thể xin tải. Class BitField chứa các phương thức sau:

    def has\_piece(self, index):

Để kiểm tra xem bit thứ index (tính từ 0) của bitfield có bằng 1 hay không.

    def set\_piece(self, index):

Đặt giá trị bit thứ index (tính từ 0) của bitfield bằng 1.

- File piece.py chứa class Piece là lớp đại diện cho đối tượng mảnh của file. Mỗi mảnh sẽ có những thuộc tính như thứ tự mảnh, độ dài, và mã hash của mảnh đó.

class Piece:

    def \_\_init\_\_(self, index, length, hash):

        self.index = index

        self.length = length

        self.hash = hash

- File peer.py chứa hiện thức class Peer cho đối tượng địa chỉ của peer trong mạng

class Peer:

    def \_\_init\_\_(self, ip: str, port: int):

        self.ip = ip

        self.port = port

- File handshake.py chứa hiện thực class Handshake cho đối tượng gói tin Handshake. Class Handshake có hai biến thuộc tính chính là info\_hash và peer\_id. Info\_hash là mã hash của trường info của file torrent được dùng để tải file dữ liệu về peer, peer\_id là mã định danh của peer đó trong mạng. Hai thông số này được gửi kèm trong gói tin handshake để hai peer có thể nhận dạng được nhau và biết sẽ phải trao đổi dữ liệu nào. Class Handshake được định nghĩa các phương thức sau:

    def serialize(self) -> bytes:

Phương thức serialize dùng để đóng gói các dữ liệu cần gửi đi thành chuỗi bytes.

    @classmethod

    def read(cls, r) -> 'Handshake':

Read là phương thức dùng để đọc nội dung gói tin Handshake được các peer khác gửi đến từ socket r sau đó read trả về gói tin Handshake của peer hiện tại.

- File message.py gồm hai class là class MessageID và class Message.

Class MessageID đùng để liệt kê các loại thông điệp như là biến static của class và gán cho biến đó một giá trị cố định, đóng vai trò như enum. Trong ứng dụng này, ta có một số loại tin nhắn như: choke, unchoke, interested, not interested, have, bitfield, request, piece.

Class Message hiện thực đối tượng tin nhắn giữa các peer, đối tượng này có hai thuộc tính là message\_id để xác định loại tin nhắn và payload là biến chưa dữ liệu mà peer cần được gửi đi. Class Message chứa các phương thức dùng để khởi tạo các loại tin nhắn và cũng tương tự như handshake (cùng là gói tin nhưng khác cấu trúc), các tin nhắn này cũng cần có phương thức serialize để đóng gói tin nhắn thành chuỗi bytes và phương thức read để đọc và giải mã gói tin.

- File peer\_communication.py chứa class communicator là đối tượng mà peer dùng để giao tiếp với peer khác. Các thuộc tính của class này chứa thông tin cần thiết cho việc giao tiếp với một peer khác và các phương thức được đóng gói hỗ trợ việc giao tiếp như phướng thức gửi các loại tin nhắn, kiểm tra tính toàn vẹn của dữ liệu nhận được, giúp hai peer giao tiếp dễ dàng hơn.

    def connect(self, peer: Peer):

Phương thức connect dùng để kết nối với peer đầu vào.

- File upload\_manager.py chứa hiện thực class UploadingManager quản lý việc chia sẻ file của client. Đối tượng UploadingManager có những phương thức quan trọng như:

    def build\_piece\_to\_file\_map(self):

Xây dựng một directory để ánh xạ một mảnh với một bộ những thông số như: chỉ số file của mảnh, độ dời của mảnh trong file, và kích thước của mảnh trong file. Các thông số này giúp xác định vị trí của mảnh đó trong file hoặc thư mục gốc cần lưu.

    def upload\_piece(self, communicator, index, begin, length):

Upload\_piece dùng để trích xuất một khối dữ liệu của mảnh thứ index, bắt đầu tại vị trí begin và có độ dài length được yêu cầu. Sau đó dữ liệu được gửi cho peer khác bằng đối tượng communicator (đã được định nghĩa trước đó).

    def add\_peer(self, peer, client\_socket):

Add\_peer là phương thức có chức năng thêm một peer vào danh sách các peers đang được chia sẻ dữ liệu từ peer hiện tại. Khi add\_peer được gọi, hệ thống sẽ cấp phát một luồng mới để thực hiện chuyển dữ liệu cho peer được truyền từ đầu vào của phương thức.

    def handle\_peer\_requests(self, communicator):

Đây là phương thức được dùng để cụ thể hóa việc chuyển dữ liệu cho peer nhận thông qua đối tượng communicator được truyền từ đầu vào. Phương thức sẽ phân tích yêu cầu từ phía peer kia và xử lý các yêu cầu đó. Nếu đó là yêu cầu tải dữ liệu, phương thức upload\_piece sẽ được gọi để gửi dữ liệu phù hợp theo yêu cầu.

- File download\_manager.py là hiện thực cho việc xử lý yêu cầu download từ client. Các chức năng dùng để download dữ liệu được bao đóng trong class DownloadingManager. Đối tượng DownloadingManager có một số thuộc tính lưu trữ các giá trị cần thiết để quản lý việc download. Các phương thức quan trọng của DownloadingManager bao gồm:

    def start\_download(self, peers, pieces, info\_hash, peer\_id, download\_dir, files)

Là phương thức nhận vào các giá trị cần thiết cho việc download. Sau đó, nó sẽ khởi tạo các biến cần thiết và gọi đến các phương thức khác để thực hiện từng bước của quá trình download. Cuối cùng, phương thức sẽ kiểm tra tính đúng đắng của dữ liệu tải về và lắp ráp dữ liệu vào địa chỉ đã được chỉ định để thu được file tải về.

 def download\_piece(self, client, piece):

Download\_piece được dùng để tải mảnh file cụ thể từ peer được chỉ định từ đầu vào. Phương thức sẽ trả về chuỗi bytes là dữ liệu nhận được từ peer gửi.

    def check\_piece\_integrity(self, piece, data):

Kiểm tra tính nguyên vẹn của dữ liệu tải về là một việc không thể thiếu khi download và phương thức check\_piece\_integrity giúp hiện thực điều đó. Phương thức này sẽ so sánh mã hash của data với mã hash của mảnh được lưu trong piece để xem liệu rằng hai mã hash này có giống nhau hay không. Nếu cả hai giống nhau, dữ liệu tải về nguyên vẹn và ngược lại.

    def download\_worker(self, peer, work\_queue, results\_queue, info\_hash, peer\_id, total\_pieces):

Phương thức download\_worker có nhiệm vụ liên hệ trực tiếp với peer để xin tải dữ liệu, rồi sẽ tuần tự gọi phương thức download\_piece và check\_piece\_integrity để tải về và xác thực từng mảnh dữ liệu một. Quá trình sẽ hoạt động đến khi work\_queue trống và mảnh dữ liệu tải về nguyên vẹn được lưu trong results\_queue.

    def assemble\_file(self, results\_queue, download\_dir, files, piece\_length):

Assemble\_file được dùng để lắp ráp các mảnh dữ liệu trong results\_queue theo đúng thứ tự của file dữ liệu ban đầu và lưu file vào đường dẫn được chỉ định từ đầu vào của phương thức.

**Thư mục Tracker chứa code hiện thực cho một tracker. Thư mục có hai file code là client\_list.py và tracker\_server.py.**

- Client\_list.py là file dùng để hiện thực class ClientList để quản lý danh sách các peer tham gia vào mạng. Do vậy class này sẽ được định nghĩa một số phương thức cho việc thao tác với danh sách các peer.

    def update\_peer(self, info\_hash, peer\_id, ip, port, uploaded, downloaded, left, event):

Update\_peer dùng để cập nhật các thông tin của peer được chỉ định trong danh sách bằng các giá trị đầu vào. Nếu peer được yêu cầu cập nhật chưa có trong danh sách, Update\_peer sẽ chèn thêm thông thông tin của peer này vào danh sách.

    def remove\_peer(self, info\_hash, peer\_id):

Remove\_peer dùng để xóa peer có định danh là peer\_id ra khỏi danh sách các peer cùng chia sẻ một file nào đó, file này được xác định bằng giá trị của info\_hash.

    def remove\_peer\_from\_all(self, peer\_id):

Xóa tất cả thông tin của peer có định danh là peer\_id ra khỏi danh sách.

    def get\_peers(self, info\_hash, exclude\_peer\_id=None):

Trả về địa chỉ của tất cả các peer đang chia sẻ file có info\_hash được chỉ định ngoài trừ peer có định danh là exclude\_peer\_id.

    def get\_all\_clients(self):

Trả về danh sách chứa địa chỉ các peer đã từng liên lạc với tracker.

    def get\_scrape\_info(self, info\_hash):

Trả về số lượng client “completed” (seeder), “incompleted” (leecher), “downloaded” (số lượt download thành công).

- File tracker\_server.py chứa code để khởi chạy tracker, có định nghĩa class TrackerServer cho đối tượng tracker. Class TrackerServer kế thừa từ class BaseHTTPRequestHandler, là lớp có sẵn trong module http.server của Python, lớp này định nghĩa các phương thức để xử lý các loại yêu cầu HTTP khác nhau (GET, POST, ...). Tracker quản lý một đối tượng ClientList và có các phương thức để xử lý yêu cầu từ peer cũng như yêu cầu từ người quản lý tracker như:

    def handle\_announce(self, params):

Handle\_announce xử lý yêu cầu annouce từ phía peer và gửi phản hồi đính kèm danh sách chứa địa chỉ của các peers trong mạng chứa file mà peer yêu cầu cần về lại phía peer.

    def handle\_scrape(self, params):

Handle\_scrape xử lý yêu cầu scrape của peer gửi và gửi phản hồi đính kèm danh sách chứa thông tin trạng thái file của các peers trong mạng yêu cầu cần về lại phía peer.

    def handle\_ping(self, params):

Handle\_ping là phương thức hiện thực cho lệnh ping của tracker. Tracker sẽ gửi tin nhắn ping đến tất cả các peers trong mạng và chờ phản hồi từ phía peer. Nếu peer nào không gửi phản hồi đến tracker thì lập tức tracker sẽ loại peer đó ra khỏi danh sách các peers trong mạng.

    def do\_GET(self):

Phương thức do\_GET nhận yêu cầu từ peer gửi đến và gọi phương thức xử lý phù hợp để xử lý yêu cầu cho peer.