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

Computer Network (CO3094)

ASSIGNMENT 1

Building a network application

Lecturer: Nguyen Le Duy Lai.
Students: Tran Xuan Hao - 2252191.
Nguyen Minh Quan - 2252683.
Nguyen Minh Tien - 2150033.

HO CHI MINH CITY, NOVEMBER 2024

1 Member list & Workload

No.	Fullname	Student ID	Tasks	Percentage of work
1	Tran Xuan Hao	2252191	- Upload	100%
2	Nguyen Minh Quan	2252683	- Tracker	100%
3	Nguyen Minh Tien	2150033	- Download	100%

Contents

2 Introduction

This project involves the development of a simple P2P file-sharing application, modeled after the BitTorrent protocol, as part of the Computer Networks course. The application aims to provide users with the ability to download and upload files across a network of peers efficiently. A centralized server, acting as a tracker, facilitates peer coordination but does not directly handle the file data.

The application will include the following core components:

- **Tracker Protocol**: Manages peer connections and provides information about available file fragments.
- **Peer-to-Peer Communication**: Enables direct file exchange between clients.

3 Basic Theory

3.1 What is Protocol?

A protocol is a set of rules, guidelines, or procedures designed to govern specific activities or processes, ensuring they are carried out in a systematic and consistent manner. Protocols exist in many fields, each tailored to the needs of that domain.

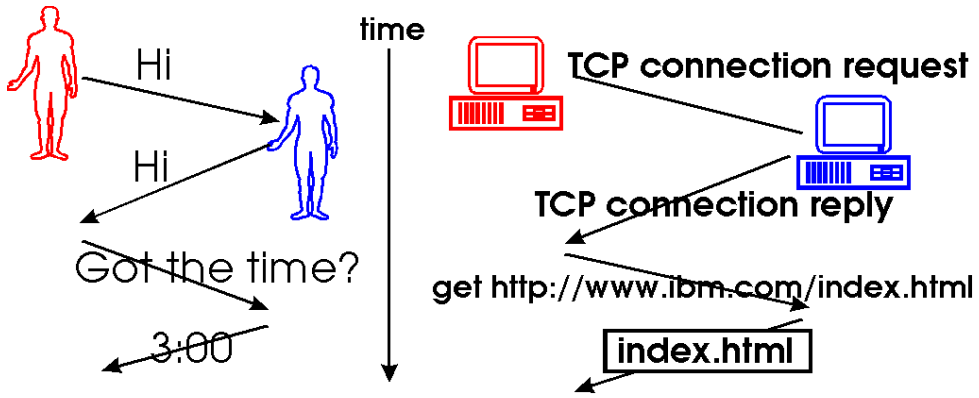

Figure 1: Protocol

3.2 Some popular Protocols

3.2.1 TCP/IP Protocol

TCP/IP (Transmission Control Protocol/Internet Protocol) is a suite of communication protocols used for connecting devices on the internet. It serves as the backbone of the internet and most modern networks.

3.2.2 Socket Protocol

The socket protocol provides a programming interface for network communication. It's not a protocol in itself but rather an API that enables processes to send and receive data over a network using underlying protocols like TCP or UDP.

3.2.3 HTTP Protocol

HTTP (Hypertext Transfer Protocol) is a protocol used for transferring hypertext (e.g., web pages) over the web. It operates at the Application Layer of the TCP/IP model.

4 Tracker HTTP Protocol

4.1 Overview

In a peer-to-peer (P2P) file-sharing system like torrents, the tracker plays an important role in connecting peers (devices or computers participating in file sharing) with each other. Although the tracker itself does not contain file content, it helps peers discover each other and establish the necessary connections for efficient file sharing.

When a new peer joins and requests to download a file, the tracker provides it with a list of other peers that possess parts of the file so it can begin downloading. For new P2P networks, the tracker helps new peers joining the network find other peers, thereby initiating the download process. Once connections between peers are established, the peers can directly transfer data to one another without the need for further involvement from the tracker.

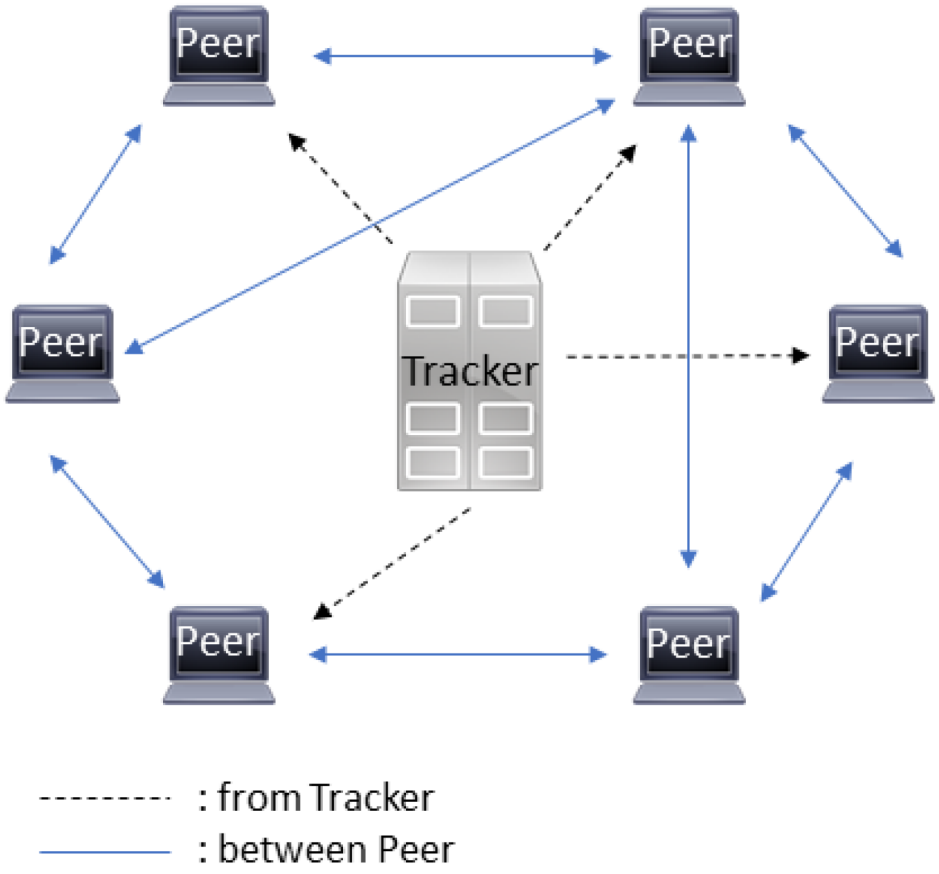

Figure 2: Tracker

4.2 Implementation

```
    def get_peers_from_trackers(self):
        self.send_tracker_request('started')
        self.try_peer_connect()
        return self.connected_peers

    def send_tracker_request(self, event):
        for i, tracker in enumerate(self.torrent.announce_list):
            if len(self.dict_sock_addr) >= MAX_PEERS_TRY_CONNECT:
                break

            tracker_url = tracker[0]

            if str.startswith(tracker_url, "http"):
                try:
                    self.http_scraper(self.torrent, tracker_url, event)
                except Exception as e:
```

```
                    logging.error("HTTP scraping failed: %s " % e.__str__())
            else:
                logging.error("unknown scheme for: %s " % tracker_url)

    def try_peer_connect(self):
        logging.info("Trying to connect to %d peer(s)" % len(self.
    dict_sock_addr))

        for _, sock_addr in self.dict_sock_addr.items():
            if len(self.connected_peers) >= MAX_PEERS_CONNECTED:
                break

            new_peer = peer.Peer(int(self.torrent.number_of_pieces),
    sock_addr.ip, sock_addr.port)
            if not new_peer.connect():
                continue

            self.connected_peers[new_peer.__hash__()] = new_peer
            print('Connected to %d/%d peers' % (len(self.connected_peers),
    MAX_PEERS_CONNECTED))
```

5 Download

5.1 Step 1: Read the torrent file

A torrent file (also known as a metainfo file) contains a bencoded dictionary with the following keys and values:

- announce: URL to a "tracker", which is a central server that keeps track of peers participating in the sharing of a torrent.

- info: A dictionary with keys:

- length: size of the file in bytes, for single-file torrents
- name: suggested name to save the file / directory as
- piece length: number of bytes in each piece
- pieces: concatenated SHA-1 hashes of each piece

To calculate the info hash, you'll need to:

- Extract the info dictionary from the torrent file after parsing

- Bencode the contents of the info dictionary
- Calculate the SHA-1 hash of this bencoded dictionary

```
def initial(file_path):
    file_path = sys.argv[2]
    with open(file_path, "rb") as file:
        torrent = file.read()
    torrent_content = bencodepy.decode(torrent)
    hash_info_hex = hashlib.sha1(bencodepy.encode(torrent_content[b"info"]))
    .hexdigest()
    hash_info_dig = hashlib.sha1(bencodepy.encode(torrent_content[b"info"]))
    .digest()
    return torrent_content, hash_info_hex, hash_info_dig
```

5.2 Step 2: Get list peers

In this stage, you'll make a GET request to a HTTP tracker to discover peers to download the file from.

Tracker GET request You'll need to make a request to the tracker URL you extracted in the previous stage, and include these query params:

- info_hash: the info hash of the torrent

- 20 bytes long, will need to be URL encoded
- Note: this is NOT the hexadecimal representation, which is 40 bytes long

- peer_id: a unique identifier for your client

- A string of length 20 that you get to pick.
- port: the port your client is listening on
- You can set this to 6881, you will not have to support this functionality during this challenge.

- uploaded: the total amount uploaded so far

- Since your client hasn't uploaded anything yet, you can set this to 0.

- downloaded: the total amount downloaded so far

- Since your client hasn't downloaded anything yet, you can set this to 0.

- left: the number of bytes left to download

- Since you client hasn't downloaded anything yet, this'll be the total length of the file (you've extracted this value from the torrent file in previous stages)

- compact: whether the peer list should use the compact representation

```
paras = {
        "info_hash": hash_info,
        "peer_id": client.id,
        "port": 6881,
        "uploaded": UPLOADED,
        "downloaded": DOWNLOADED,
        "left": left,
        "compact": COMPACT
    }
```

Tracker Response The tracker's response will be a bencoded dictionary with two keys: - interval:

- An integer, indicating how often your client should make a request to the tracker.
- You can ignore this value for the purposes of this challenge.

- peers:

- A string, which contains list of peers that your client can connect to.
- Each peer is represented using 6 bytes. The first 4 bytes are the peer's IP address and the last 2 bytes are the peer's port number.

5.3 Step 3: Handshake

In this stage, we'll establish a TCP connection with a peer and complete a handshake.

The handshake is a message consisting of the following parts as described in the peer protocol:

- length of the protocol string (BitTorrent protocol) which is 19 (1 byte)
- the string BitTorrent protocol (19 bytes)
- eight reserved bytes, which are all set to zero (8 bytes)
- sha1 infohash (20 bytes) (NOT the hexadecimal representation, which is 40 bytes long)
- peer id (20 bytes) (generate 20 random byte values)

After we send a handshake to our peer, we should receive a handshake back in the same format.

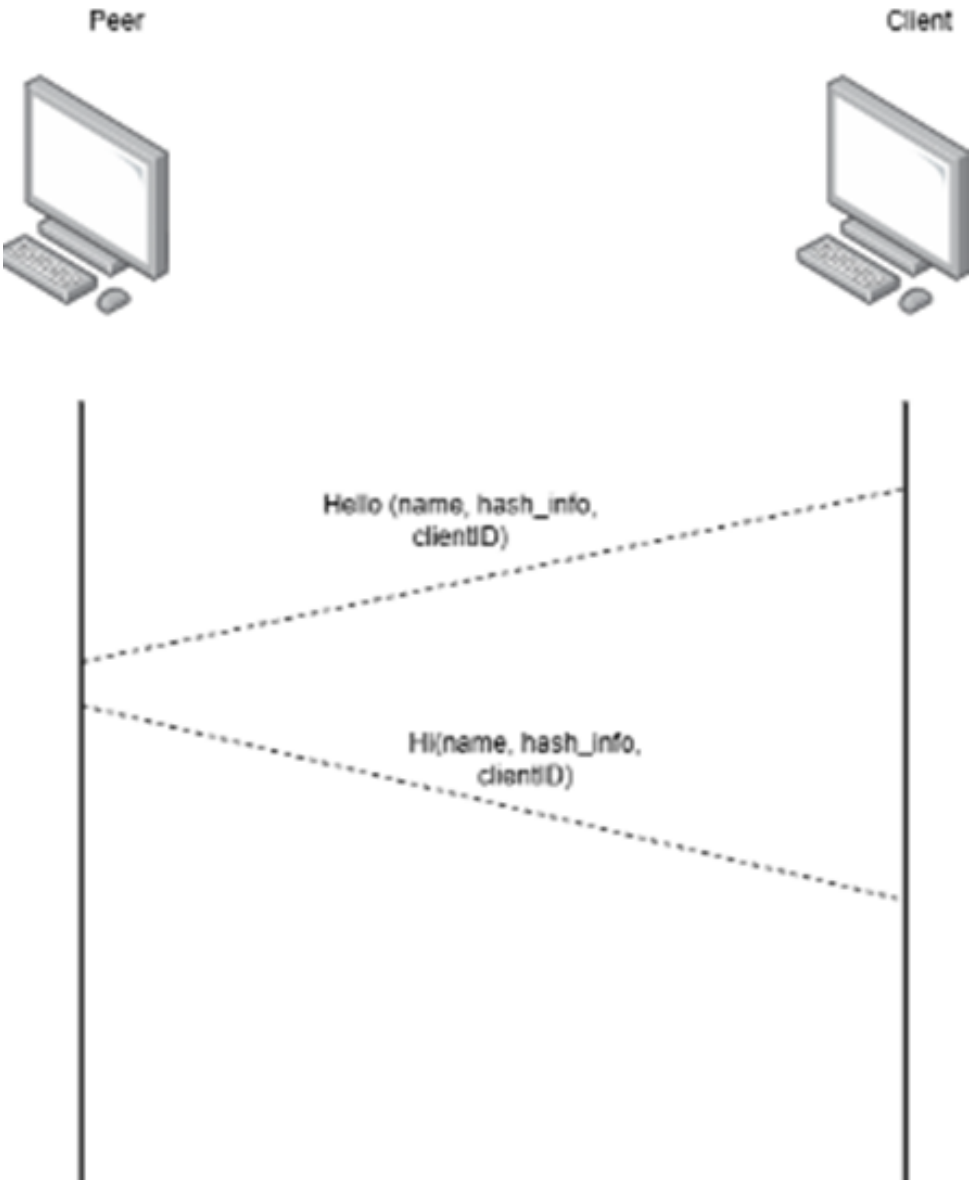

Figure 3: Handshake

5.4 Step 4: Download pieces

Peer Messages: consist of a message length prefix (4 bytes), message id (1 byte) and a payload (variable size).

Here are the peer messages we'll need to exchange once the handshake is complete:

- Wait for a bitfield message from the peer indicating which pieces it has

- The message id for this message type is 5.

- You can read and ignore the payload for now, the tracker we use for this challenge ensures that all peers have all pieces available.

- Send an interested message

- The message id for interested is 2.
- The payload for this message is empty.

- Wait until you receive an unchoke message back

- The message id for unchoke is 1.
- The payload for this message is empty.

- Break the piece into blocks of 16 kiB (16 * 1024 bytes) and send a request message for each block

- The message id for request is 6.
- The payload for this message consists of:
  + index: the zero-based piece index
  + begin: the zero-based byte offset within the piece
  + length: the length of the block in bytes

• - Wait for a piece message for each block you've requested

- The message id for piece is 7.
- The payload for this message consists of:
  + index: the zero-based piece index
  + begin: the zero-based byte offset within the piece
  + block: the data for the piece, usually 2^{14} bytes long

After receiving blocks and combining them into pieces, we'll want to check the integrity of each piece by comparing its hash with the piece hash value found in the torrent file.

In this assignment, we use **Rarest First Method**: The Rarest First Method is a strategy used in peer-to-peer file-sharing systems, particularly in the BitTorrent protocol, to optimize the distribution of file pieces among peers in a swarm. This approach prioritizes downloading the rarest pieces of a file first, rather than choosing pieces at random or the most available pieces.

Index	Num
0	2
1	1
2	2
3	3
4	1
5	1
6	2
7	3

Index	Num
1	1
4	1
5	1
0	2
2	2
6	2
3	3
7	3

Figure 4: Rarest First Method

5.5 Step 5: Download file

You can start with using a single peer to download all the pieces. You'll need to download all the pieces, verify their integrity using piece hashes, and combine them to assemble the file.

6 Conclusion

The development of this P2P file-sharing application is an opportunity to apply theoretical networking concepts in a practical context. By simulating a simplified version of a torrenting system, the project underscores the complexities of peer-to-peer communication and the management of network resources.

Through teamwork and a structured approach, this project fosters critical thinking, collaborative problem-solving, and hands-on experience with protocols like TCP/IP and the BitTorrent specification. The inclusion of optional advanced features offers room for creativity and innovation, encouraging students to explore beyond the baseline requirements. Ultimately, the successful completion of this assignment will provide a deeper understanding of modern file-sharing technologies and their underlying principles, preparing students for real-world applications in network development.

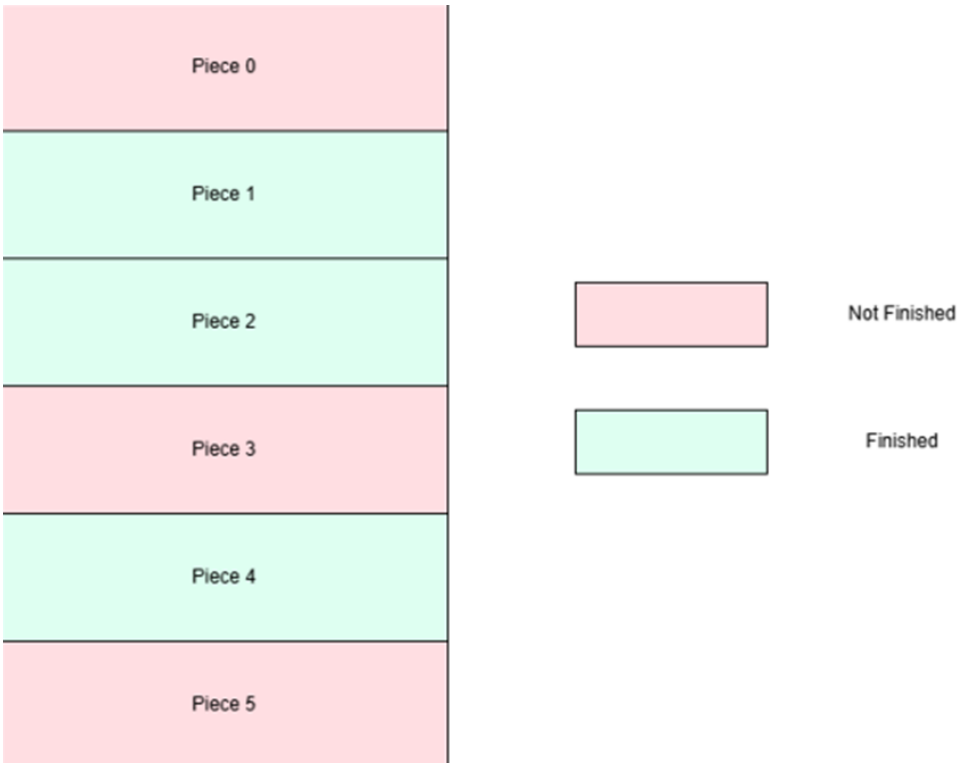

Figure 5: Download File

7 References

1. https://wiki.theory.org/BitTorrentSpecification
2. http://www2.ic.uff.br/~michael/kr1999/1-introduction/1_02-protocol.htm
3. https://www.mdpi.com/2079-9292/12/1/165