CS305 2025 Spring Programming Assignment 1

BitTorrent-based P2P File Sharing

Deadline: Saturday, April 5th, 2025, 23:59:00

1. Introduction

In Peer-to-Peer (P2P) networks, peers directly share files without relying on a centralized server (see Figure 1). In this assignment, you need to implement a simplified BitTorrent-based P2P file-sharing protocol.

Figure 1: P2P File Sharing Network (Left) vs. Client-Server Network (Right)

A BitTorrent network typically consists of four types of entities: **trackers, seeders, requesters, and torrent repository servers**:

- **Seeders:** Peers holding files to share. A seeder generates a torrent file that holds the information (info) of the shared files, etc. Then, the seeder sends its address, port, hash value of the info (`info_hash`), and the names of shared files to the tracker to register the shared files. This document introduces more information about `info_hash` later.
- **Requesters:** Peers downloading files. They must acquire and decode the torrent file of the shared files in advance to obtain the `info_hash`. Then, they can send the `info_hash` to the tracker to get the list of seeders with the requested files. Finally, they can download the shared files from one or more seeders by sending them the requested file names.
- **Trackers:** Servers coordinating interactions between seeders and requesters. They are responsible for managing the `info_hash` and names of shared files, as well as the addresses and ports of related seeders. Additionally, they can search and return the list of seeders when requesters upload an `info_hash`.
- **Torrent repository servers:** Store torrent files (ignored in this assignment).

2. What you need to do

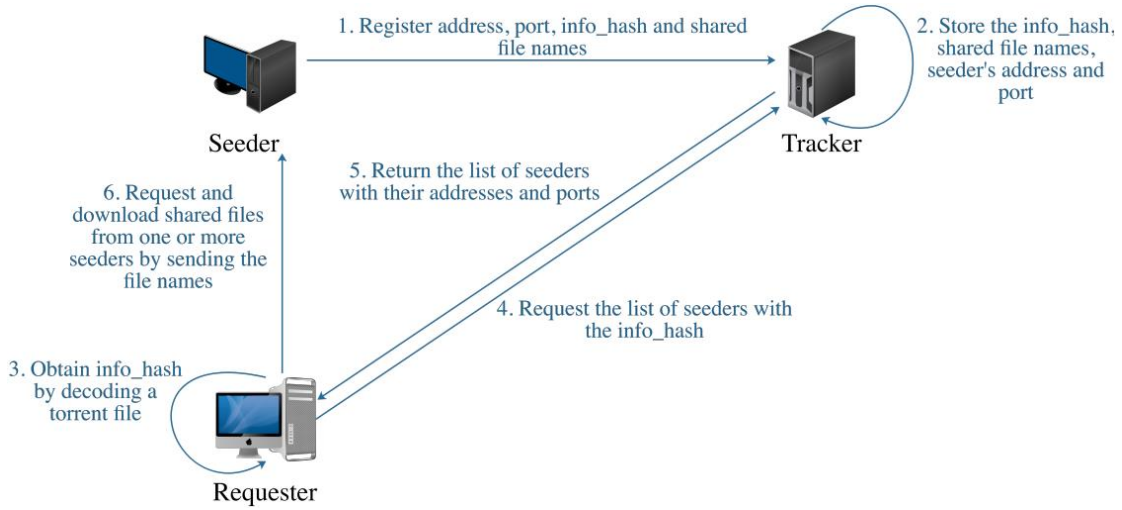

Figure 2. Simplified BitTorrent Model

For simplicity, implement only the **Tracker**, **Seeder**, and **Requester** as depicted in Figure 2. The following are tasks to be completed in this assignment.

- **Tacker.py: Design the functions needed to implement a tracker.**
  - ✓ `announce()`: Handle announcement sent by the seeder using HTTP protocol, and record `info_hash` and names of shared files as well as seeders' addresses and ports. **The `info_hash` should be represented by 40 hexadecimal digits.**
  - ✓ `get_seeders()`: Search the list of seeders based on an `info_hash` and return the list with seeders' addresses and ports to requesters.
  - ✓ `show_tracker_data()`: Return all tracker data when requested.
- **Seeder.py: Design the functions needed to implement a seeder.**
  - ✓ `get_peer_folder()`: Create a seeder folder named "peer_6881." Please note that the seeder's port defaults to 6881.
  - ✓ `create_torrent()`: Create a torrent file for a given file "example.txt" to be shared. Note that a torrent file should be in **bencoded** format, which will be introduced later. Moreover, since we remove torrent repository servers in our model, we assume that the torrent file is shared with requesters after being generated. You can generate the torrent file and copy it to the requester folder.
  - ✓ `get_info_hash()`: Decode a torrent file to compute its `info_hash` and piece list.

- ✓ `announce_to_tracker()`: Send an announcement to the tracker using HTTP protocol to register the shared file.
- ✓ `download()`: Serves a file to other peers upon request.
- ✓ `run_peer()`: the main function to run a seeder, defining its operation logic.

- **Requester.py: Design the functions needed to implement a requester.**
  - ✓ `get_peer_folder()`: Create a requester folder named "peer_6882." Please note that the requester's port defaults to 6882.
  - ✓ `get_info_hash()`: Decode a torrent file to obtain its info_hash and piece list.
  - ✓ `get_seeders_from_tracker()`: Request the tracker for a list of seeders with the given info_hash using HTTP protocol.
  - ✓ `download_file()`: Downloads a file from another peer and saves it in the requester's folder.
  - ✓ `request_file()`: Define the logic to request a shared file by getting the list of seeders and downloading the file from one of them.
  - ✓ `run_peer()`: the main function to run a requester, defining its operation logic.

3. Structure of a torrent file and its info_hash

```
{
    'announce': 'http://bttracker.debian.org:6969/announce',
    'info':
    {
        'length': 678301696,
        'name': 'debian-503-amd64-CD-1.iso',
        'piece length': 262144,
        'pieces': <binary SHA1 hashes>
    }
}
```

Figure 3. An example of a de-bencoded torrent file of a single shared file

In Figure 3, 'announce' represents the tracker's URL. 'Info' contains the metadata for the shared file, where 'length' specifies the file size, 'name' indicates the file name, 'piece length' refers to the size of each piece (as a file can be divided into multiple pieces), and 'pieces' represents the concatenation of SHA-1 hash values for each piece. Here, SHA-1 is a hash function that can map any input of arbitrary size to a unique 160-bit (20-byte) hash value known as a message digest - typically rendered as 40 hexadecimal digits.

`info_hash` is a SHA-1 hash of the **bencoded** `info` section, represented as 40

hexadecimal digits. Use Python's `hashlib` and `bencodepy` for hashing and encoding/decoding.

In the BitTorrent protocol, `info_hash` is not generated directly from 'info'. Instead, 'info' must be converted into **bencoded** format before inputted into a hash function to compute its hash value. The functions `bencodepy.encode()` and `bencodepy.decode()` in `bencodepy` library can convert the info into bencoded format and decode a bencoded file. Moreover, **a torrent file should be a bencoded format of the content shown in Figure 3.**

4. Tips

1. Please refer to the logic of file sharing in Figure 2.
2. All entities interact with the HTTP protocol, and you can use the "`requests`" library.
3. More information about torrent files can be found at https://en.wikipedia.org/wiki/Torrent_file.
4. All entities can run a Flask web server to process HTTP requests.
5. In practice, each peer runs in one host, which may not be feasible for most students. As such, you can run different peers on different threads.
6. Please complete the codes of each function mentioned above.

5. How Your Codes Tested and Graded

Step 1: Execute **Tacker.py** to listen for the registration and get-seeder-list requests from the seeder and requester. **(25 pts)**

Step 2: Execute **Seeder.py** to create a torrent file for a text file and register it with the tracker. **(30 pts)**

Step 3: Execute **Requester.py** to decode a torrent file and obtain the info_hash. Afterward, send the info_hash to the tracker to request the list of seeders and download the text file from one of the seeders. **(35 pts)**

Step 4: **Write a report to** screenshot and briefly explain how you implement the above steps and the results. **(10 pts)**

6. Submission

Please package all your files (including your report) into a zip file named "{Your student ID number}_{Your name}" and submit on Blackboard.