CSC 645-01: Computer Networks
Peer to Peer (P2P)
Programming Assignment #3
Assigned on: 10/28/2019
Due on: 11/18/2019

Jose Ortiz

October 28, 2019

In this programming assignment, you will implement a **centralized** peer-to-peer network architecture (P2P), including the basic implementation of the BitTorrent protocol (BTP). General P2P architectures can be classified into centralized and decentralized, In P2P centralized architectures, a new peer connects to a **Traker**. The tracker, then, sends back, a list of all the peers connected to the network that have the requested resource. In P2P decentralized architecture, each peer is also a Traker, and it can share only limited resources because it only sees the partial network. P2P centralized architecture is more scalable than the decentralized one. Examples of P2P applications that are/were implementing centralizing architectures are Nasper, the Berkeley Open Network Infrastructure (BOINC) and some versions of BitTorrent. Other versions of BitTorrent implement decentralized architectures as well.

1. **General terminology in a P2P BitTorrent protocol (BTP):**

   (a) **Peer**
   A peer is a client to which other clients can connect and transfer data, a peer is usually referred to a client with incomplete data. Note that a specific peer is unique, and it is identified by its peer id (client socket id). Peers can also be classified as seeders and leechers in the swarm

   (b) **Swarm**
   A network of peers that actively operate on a given torrent.

   (c) **Traker**
   A tracker is a centralized server that holds information about one or more torrents and associated swarms. It functions as a gateway for peers into a swarm. Note tha in centralized P2P srchitectures, a tracker is a HTTP service, and it does not provide access to any downloading data nor has access by itself to such data.

   (d) **Metainfo File**
   The metainfo file is also called as a torrent file and has a .torrent extension. This file mainly contains encoded information regarding the URL of the tracker, name of the file, and hashes of the pieces of the file for verifying downloaded pieces of the file.

   (e) **Torrent**
   A torrent is the term given for a file that is being or was downloaded by a peer.

(f) **Piece**
A torrent is divided in pieces by the seeder. All the pieces from a torrent have the same size with the only exception of the last piece which may have the leftover bytes. Info about the file, the length of its pieces, and the hashes of the pieces are found in the metainfo file (the file with extension .torrent)

(g) **Blocks**
Since pieces from a torrent may became huge in size, we need to divide them into blocks. Blocks are the basic structure transmitted in the swarm bit by bit. Normally, blocks are divided in 20 bytes of data. But it depends mostly on the needs of the app

(h) **Seeder/Seed**
A seeder is a peer that has the entire copy of the data (here you can think in terms of files), and it offers it for upload. As you can guess, when a swarm has many seeders, the better the chances of getting better downloading rates.

(i) **Leech/Leecher**
A Leecher is a peer that is downloading data from the swarm. Once a leecher has all the pieces from a file (complete file), then the leecher becomes a seeder. Note that a leecher may be downloading data, and at the same time, avoiding to upload data into the swarm. This will impact negatively in the download rates of the swarm.

2. **Overall operation of BitTorrent Protocol (BTP)**

BTP consist in two main distinct protocols Tracker HTTP Protocol (THP), and Peer Wire Protocol (PWP). THP defines methods used by peers for contacting **Trakers** to join the **swarm**. PWP defines mechanisms that allow communication between peers.

(a) High level steps for a client to download a torrent
- A client must download the metainfo file with .torrent extension via HTTP. Then, once the file is downloaded, the client extracts the IP address of the tracker, and creates a TCP connection. After the mandatory handshaking process between client and tracker, a list of the peers sharing the resource requested in the swarm is sent to the new client. Then the client becomes a peer in the swarm
- The new peer, then creates a connection with all the peers sharing the requested file in the swarm, and the sharing process is initiated between two peers if the uploader is not choked, and the downloader is interested (more about this later). Only when the two peers agree to initiate the downloading process, the downloader becomes a leecher.
- Once a piece of a file is downloaded by a leecher, it will make that piece available for downloading in the swarm. When the leecher completes the download process of all the pieces from a file, it becomes a seeder in the swarm for that file.
- A higher number of seeders seeding the same file to the network will make a huge positive impact in the download rates in the swarm

(b) High level steps to publish a torrent (seed)
- There must be at least a tracker in the swarm. Note that in centralized P2P architectures there may exist more than one tracker. In decentralized ones, there is always more than one tracker (a peer may be tracker too). **For this assignment, you can assume that there is only one tracker in the swarm**
- A meta-info file must exist in the tracker containing information about the structure of the swarm, peers connected to it, and resources shared by each peer and seeds.
- The swarm must have at least a seed with access to the complete file.

3. **High level implementation steps:**

   (a) **The tracker meta-info** In this assignment, you will create (with your preferred text editor) a file with extension .torrent that will contain all the meta-info related to the tracker, and info about the resource that needs to be downloaded. Normally torrent files are encoded in Bencode. However, for this assignment that step is optional. Once the file is created put this file in the root of your project directory.

   Here is what a de-bencoded torrent file (with piece length 256 KiB = 262,144 bytes) for a file example.pdf (whose size is 678 301 696 bytes) might look like:

```
{
    'announce': '127.0.0.1:12000',
    'info':
    {
        'name': 'example.pdf',
        'piece length': 262144,
        'length': 678301696,
        'pieces': <binary SHA1 hashes>
     }
}
```

   The announce value is the IP Address/port or URL of the tracker containing all the info about the peers in the swarm that have the complete 'example.pdf' file or one or more pieces of that file.

   (b) **Tracker**
   Create a **Tracker** class that inherits the following methods from the **Server** class: listen(), accept(), send(), recv(). For the server class, you can reuse the one from assignment #1. Additionally, the Tracker class will handle a list of objects from the class **Resource** that contains all the information about the peers connected to the swarm that are sharing specific resources. Each time a peer is accepted by the tracker, the peer's info is added to the class Resources. This info must be persistent, so you may need to store this information in disk at some point. Also, every time a connected peer request info about the seeds and peers sharing a resource the tracker provide that info to the peer requesting it.

   (c) **Resource**
   This class have all the configuration data related to the resource that is being shared by peers and seeds in the swarm. Note that this class do not provide functionalities to access such data nor save the actual files shared in the swarm. The following is some useful configuration data that must handle this class: the id, and name of the resource. The seeds/peers that have this resource including their role (seed, peer, or leecher), their IP addresses and ports. and their ids in the swarm (client id). Also this class must keep track of the pieces missing for a specific file. You are free to add here more configuration data. The Resource class must have a method that saves configuration data in disk.

   (d) **Peer**
   The Peer class inherits methods from both, the Client and the Server classes because in a P2P architecture all the clients are also servers. In addition. the peer class provides the following functionalities:

      i. Extract the metainfo from the .torrent file, and using that info, create a TCP connection with the tracker. Note that you don't need to send any info about the file requested to the tracker (i.e name or pieces). The tracker do not know anything about it. It only knows about the peers sharing the resource.

ii. Once the handshake is performed. receive the peer id, and the list of peers sharing the requested resource from the tracker

iii. Create a TCP connection with all the peers in the swarm from the list provided by the tracker. After the initial handshake with all the peers, and if you are interested in starting the downloading process.send a message with the value 'interested' set to 1 to each of the peers in the swarm. The downloading process start when a peer responds with the 'unchoked' status.

iv. Then, set your status to leecher and interchange symmetric messages in the swarm using the peer protocol described in section 3e of this document. Those messages contain the blocks from pieces that you are requesting. Messages are explained in detail in the peer protocol section. Once a piece from a file is completed then, the piece can be uploaded to the swarm, so others can download it. When you have the entire file, set your status to seeder.

v. Like in any P2P client app, we need to log and show in console info about the downloading and uploading processes taking place. The below figure illustrates those processes at one specific moment of the application execution. Note that percentages of download and upload data including peers and seeds data must be updated in real time.

```
***** P2P client App *****
Peer Info: id: 12344, IP: 127.0.0.1:13000
Tracker/s info: IP: 127.0.0.1:12500
Max download rate: 100 b/s
Max upload rate: 50 b/s
*** Downloads in progress ***
- File: example.pdf total downloaded: 45%
0.02 b/s peer 127.0.0.1:12000 1234 2 2%
0.01 b/s peer 127.0.0.1:12345 2347 1 34%
0.10 b/s seed 127.0.0.1:12300 4354 7 22%
- File: example2.pdf total downloaded: 15%
0.04 b/s peer 127.0.0.1:12000 1234 4 33%
0.05 b/s seed 127.0.0.1:11034 2333 6 22%
0.00 b/s seed 127.0.0.1:11035 5546 0 0% ch
- File: example3.pdf total downloaded: 100%
*** Uploads in progress ***
- File: example.pdf total uploaded: 33%
0.23 b/s peer  127.0.0.1:14000 23E4 2 1%
0.23 b/s leecher 127.0.0.1:12387 5676 3 30%
0.23 b/s leecher 127.0.0.1:12388 9898 4 34%
```

- **Downloads values** values are: (1) the rate at which the uploader is uploading the file, (2) peer or seed, (3) IP address and port of the peer/seed uploading the data, (4) the uploader id, (5) the piece number that is being downloaded, (5) the percent downloaded of that specific peace of data (6) if chocked set to ch
- **Upload values** values are: (1) the rate at which the data is being uploaded, (2) The receiver role; peer or leecher, (3) IP address and port of the receiver, (4) the receiver id, (5) the piece number that is being downloaded, (5) the percent downloaded of that specific peace of data

(e) **The Peer Protocol**

The BitTorrent's peer protocol operates over TCP or uTP. Peer connections are symmetrical. Messages sent in both directions that look the same, and data can flow in either direction. The following is a general step by step high level description about how to implement this protocol:

i. The peer protocol refers to pieces of the file by index as described in the metainfo file, starting at zero. When a peer finishes downloading a piece and checks that the hash matches, it announces that it has that piece to all of its peers.

ii. Connections, in both ends, must define one of the following states: interested or not interested and chocked or not. When a peer is chocked it won't send any data. Connections **always** start out choked and not interested. The data transfers begins between two peers when a peer is interested and the other peer is not chocked.

iii. Once the data transfer has began the peer will send pieces using different algorithms. In this assignment, you will implement the **tit-for-tat** transfer process, which sends chuncks (bytes from pieces) to the top four peers with highest upload rate, and re-evaluate top 4 each 30 seconds approximately with the goal of find better trading partners, and get the file faster.

iv. For this assignment, after a peer is executed, it must ask the user to set the maximum upload and downloads rates that the P2P app is allowed to use. Downloads and Uploads rates are dynamic and need to be re-evaluated every 30 seconds as follow:
Let $P1$ be a peer trying to download a resource from the swarm, D_{max} and U_{max} be the maximum download and upload rates set by the peer, $UR = (UR_2, .., UR_i, ..., UR_n)$ be the set of upload rates from the leechers or seeders uploading the file requested by $P1$ in the swarm, and F_n be the number of filed being uploaded by $P1$ to the swarm. Then, the download and uploads rates of P1 are computed as follows:

$$D(P_1) = \sum_{i=2}^{n} UR_i \quad where \quad D_{max} >= D(P_1)$$

$$U(P_1) = \frac{U_{max}}{F_n}$$

v. The message sent between peers has the possible values.

- **unchoke** if set to 1, the peer allows data to be downloaded. Otherwise, peer is in choke status and cannot sent data.
- **interested** if set to 1, the requester is interested in downloading data from a peer. Otherwise is not interested
- **have** a positive integer that defines the number of pieces from the file that the peer have.
- bitfield indicates the missing pieces. (i.e [1,0,1,1,1] means that piece 2 is missing)
- **request** messages contain an index, begin, and length
- **piece** messages contain an index, begin, and piece.
- **cancel** messages have the same payload as request messages. They are generally only sent towards the end of a download,

Note that choke, unchoke, interested and non interested values do not contain payloads. Also note that this is a basic implementation of the peer message protocol. Real implementation deals with messages bit by bit, and with offsets.

4. **Libraries allowed in this assignment**
   For this assignment the only libraries/packages allowed are threading, socket, and pickle. In addition, you can use any libraries or packages to help you with the following tasks:
   - Helping with imports problems
   - Make your data persistent. For example, when a peer reconnects to the swarm, it needs to know which pieces are missing from the file it was downloading.
   - Working with bits, since you may need to send bit by bit at a specific upload rate. You can also use such library to help you with the creation of pieces and blocks
   - Any package that will help you to detect the number of tics from the CPU processes. The builtin 'time' package is a really good one for this task.

5. **Submission Guidelines**
   (a) By the assignment due date specified at the top of this document, your complete and tested project must be hosted in /applications/peer-to-peer-app/ folder of your GitHub private class repository. All working code must be hosted in the master branch of the repository
   (b) Your README file for this assignment must contain the following info:
   - Your name and student id
   - Project name
   - Project description
   - Project purpose
   - Clear instructions about how to clone/download/install/execute the project Compatibility issues
   - A paragraph or two about what challenges you have faced in this assignment, and what you have learned from it.

   (c) I will only grade assignments located in the Master branch of your repository.
   (d) Commits done after the assignment due date won't be accepted by any means. So plan accordingly!
   (e) Assignments submitted by email won't be accepted as well.

6. **Grading Rubrics**
   (a) This assignment will be graded based in the correct implementation of all the functionalities described in this document.
   (b) Your code must be readable and self explanatory. Add comments only for some complicated functions or algorithms that need extra clarification.
   (c) This assignment is 10% worth of your final grade.
   (d) Correct and complete implementation 7%
   (e) Appropriate documentation of the code, coding style and readability. If I cannot understand your code, then your code is not good. 3%
   (f) If I cannot run the peer or the tracker python files, you will get a 0 in this assignment. 2% extracredit added to your final grade is given to all the students that will complete this assignment.