# CSC 645-01: Computer Networks
# Peer-to-Peer Network (P2P) BitTorrent Protocol
# Programming Assignment #2
# Assigned on: 07/02/2020
# Due on: 08/06/2020

Jose Ortiz

July 19, 2020

In this programming assignment, you will implement a **decentralized** 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, new peers send a request to the **Traker** (normally done via HTTP) requesting a list containing all the IP addresses of the peers that are already connected to the P2P network sharing the same file. The tracker then, replies with a response containing such list. In P2P decentralized architecture, each peer is also a Traker, and it can share only limited resources because it only sees the partial network. On the other hand, P2P decentralized networks avoid single point of failure since they do not depend of centralized trackers. 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**
   Peers are those who are uploading and downloading data at the same time from other peers. Note that they do not posses the whole resource that is being shared, only some pieces of it. Peers become seeders for a specific file when they downloaded all the pieces from that file.

   (b) **Swarm**
   A swarm is a network of peers, leeches and seeders sharing the same file. The more seeders a swarm has, there will be faster downloads rates among peers. In other words, there will be better chances of getting the file faster at high download speed.

   (c) **Traker**
   Trackers in a semi-decentralized network a tracker is a server which only function is to send the ip addresses of all the peers in the swarm. In a decentralized network, all the peers are also trackers.

   (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 file is a file that contains meta-info (not the actual data) of the file that is being shared on the swarm by all peers connected to it. It normally has extension .torrent and contain useful info about the trackers, and the pieces of the file being shared in the network such as length, SHA1 hashes of all the pieces

(f) **Piece**
A file shared in P2P 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. The length of a piece is normally 16KIB, however this is defined by the implementer. For example, a piece could also have a length of 32KIB, 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**
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 byte by byte. For example if a piece size is 16KIB, blocks in that piece will be 2KIB. So, the piece will be split into 8 blocks. The final piece of a file may have eight or less blocks.

(h) **Seeder/Seed**
Seeders are those who uploaded the torrent seeds to others, or those who downloaded the the file already. Note that seeders in a swarm do not download pieces of the file that they are seeding. However, they may download pieces from other files in swarms where they are not seeders.

(i) **Leech/Leeches**
Leeches are peers that do not have the required piece of a file. For example, if there are zero seeders in the network at some point most of the peers will become leeches.

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

Note this section is only informative to show you how the BitTorrent protocol works. Implementation is described in next step.

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. However, decentralized P2P networks using BTP rely heavily in TCP.

(a) High level steps for a client to download a torrent

- Normally, in centralized P2P, 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. For this assignment, you can assume that you will be given the .torrent file (you can find it on iLearn).
- 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).
- 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 for your programming assignment:**

(a) **The Meta-info file** In this assignment, I will provide a .torrent file (the file is on iLearn). However, you can also 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. Your peer needs to have the capacity to decode this file to extract all the info from it. You are allowed to import the python package bencode to perform this task.

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 in centralized) 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**

Recall that in this program, every peer is also a tracker. The Tracker class takes as a parameter the Server instance. It has two functionalities implemented in two different methods

- When a new peer connects, the tracker will broadcast the IP address of the new peer to all the trackers connected in the same swarm.
- Once all the Trackers objects detect a broadcast transmission from another tracker, they will include the new IP of address in their list of connected peers to the swarm.

(c) **Peer**

The Peer class is the main class of this program. It may inherit methods from both, the Client and the Server classes because in a P2P architecture all the clients are also servers. It may also create server and client objects instead of inhering their methods. That is a design decision that you need to make. In addition. the peer class provides the following functionalities:

i. Extract the meta-info 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. The only information that the tracker has is the IP addresses of the peers connected

ii. Once the peer's tracker that is trying to connect, retrieves all the list of all peers connected to the swarm, it will create a TCP connection with all the peers in the swarm from the list provided by the tracker. Take into consideration that there must be a max number of connections set. After the initial handshake with all the peers, and if the new peer is interested, and the other peers are not choked, then the downloading process starts. Note that you can only download data from four peers at the same time in the same swarm (tit-for-tat transfer protocol). Try to download from the rarest peers first. See "Application Layer" slides for more detail about this protocol.

iii. Set the peer's status to leech if it still has missing pieces of the file, 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.

iv. 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 must be updated in real time. You do not need to worry about implementing the progress bars to represent the progress of the pieces downloaded because you can use my python module HTPBS (horizontal threaded progress bars). You can find it here https://pypi.org/project/htpbs/.

v. Take into account that this program must support sharing different files at the same time in different swarms. This will be explained in detail in class labs.

(d) **Additional Classes:** I recommend to think carefully of the classes needed in this program apart from the client, server, tracker and peer ones. You will also need a connection class to keep track of all the files that are being share in the network (multiple swarms). Also, you'll need a uploads class (think about the client-handler class), and a downloads class. In addition, you may need the forwarding and routing classes. The routing class is in charge of routing the blocks downloaded that belong to each file because the peer may be downloading blocks from different files. The forwarding class is in charge forwarding blocks to their belonging pieces related to the same file. Think about how routing and forwarding protocols work in real applications (those protocols were learned in class) and then apply this knowledge here.

(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 chunks (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 run in terminal, 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 leeches or seeders uploading the file requested by $P1$ in the swarm, and $F_n$ be the number of files 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 [[1,1,1,0,1,1,0,1],[1,1,1,1,1,1,1,1],[0,0,0]]) In this example, the 0s and 1s are bits representing blogs of the pieces. If a piece is size 16KiB, then the piece is represented by 8 bits (blocks 2KiB size). Note that the last piece contains the leftover blocks. When a block of a piece is completed it is set 1 bit, when is missing is set to 0 bit. In the above example, in piece 0, blocks 3, and 6 are missing, piece 1 is completed, and piece 2 is missing all its blocks.
- **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. For more detail about this protocol see class labs. For more details: BitTorrent Specifications.

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. In my solution, I use configuration files, but you are free to use any library that will help you to maintain persistent data in disk. Note that pickle can also save data in files in disk.

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 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) You must send me an email (by the deadline of the assignment) to jortizco@sfsu.edu. The email must have the following subject and body:

   - Email Subject: CSC645 Computer Networks: P2P
   - Email Body: your name, student id and Github username. In addition, it must have a direct link to the source code of your project in the master branch of your repository.

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) P2P programming assignment is 25% of your final grade.

   (d) Correct and complete implementation 20%

   (e) Appropriate documentation of the code (README.md), coding style and readability. If I cannot understand your code, then your code is not good. 5%

   (f) If I cannot run the peer.py file, you will get a 0 in this assignment. No exceptions!!!. Make sure that you test this before submission.