CMSC 417 Final Project

1. **List of supported features:**

Our Bittorrent Client supports the core features of a bittorrent client. It is able to ingest a torrent file, parse through it and communicate with a tracker. It can receive a compact response if requested, but otherwise will take the list of peers returned from the tracker and create peer objects then place those in a list to be iterated through. We then prepare the serving aspect of the client by starting a server and handle any incoming requests for pieces as we download pieces from other peers. Additionally, we are able to download a file from an instance of our own client. Lastly we are able to download a file from other bittorrent clients.

2. **Design and implementation choices that you made**

Throughout the process of building the client, there were a few major decisions as well as some minor ones that had to be made that greatly impacted the capabilities and efficiency of our client. The first major decision was the programming language. We decided to go with Python, because of this, we were able to access many useful libraries such as asyncio, and bencoding. Though we probably could have made our program smaller and more efficient using a language like C, Python has many useful libraries that help abstract away smaller details so we could focus on the bigger picture.

Another design choice we had to make was how to handle multiple connections to all the peers. We ended up deciding to use asynchronous functions provided by Python's standard asyncio library, though we could have used multi-threading, we went with asynchronous functions because we don't have to worry as much about writing thread-safe code and avoiding race conditions. In general, it seemed that asynchronous code was easier to manage.

We chose to keep things simple by only having classes for a peer and a torrent. We felt that this kept things minimal while also managing and organizing our functions.

Generally, we set compact to 0 in our parameters, to validate and identify the peer ID. This allowed us to verify where our pieces were coming from and test our code.

Since we started with the download function from specific peers and we were able to get that to work, we ended up reusing our code for the uploading portion of the protocol. In general we reused code wherever we could to minimize our quantity of code/complexity.

When creating our request messages for fellow peers, we discovered that the maximum piece size we were able to request was 16384 (0x4000) bytes. This means our BitTorrent client is required to download larger pieces in sub pieces of size 16384 bytes until the whole piece is completely downloaded. We also had to take special care of the last piece of any given file because the length of that piece may be smaller than all other pieces.

3. Problems that you encountered (and if/how you addressed them)

We encountered an issue trying to download from other instances of our client running on the same machine on UMD's eduroam WiFi. When connected to our home routers we were able to download from other instances of our client, but on the school's WiFi our clients could not communicate with each other. In order to fix this issue, we saved the external ip address of our machine and checked if the peer that we were connecting with to download had the same ip address. If the ip address matched, then we converted the external ip address to the local ip address and the download was done through the local machine.

While trying to download from another instance of our client, we encountered an issue where the second client would stop and hang once it downloaded all the pieces that the first client advertised in its initial bitfield. However, the second client did not continue to download pieces from the first client despite the first client eventually getting all the rest of the pieces. To address this issue, we changed our client to send a "Have" message to each of the peers it is seeding pieces to as soon as it finishes downloading a new piece itself. This way, its peers will know it has those new piece(s) available and it can continue to request them.

When implementing the compact format, the data sent back to us only contained the ip address and the port of the peers. This was an issue because our code stored and verified the peer id, so our existing verification method for the handshake had to be altered to accommodate peers without a peer id, currently we verify solely using the info hash.

4. Known bugs or issues in your implementation

- 'Uploaded'/'downloaded' flag is always set to 0 in our http request to the tracker
- Occasionally face an error where the data in a response to the http get request is completely empty for a second client
- Hex string conversion error when sending an empty bitfield (no pieces) to peer

5. Contributions made by each group member
- Madhava - Seeding
- Matt - Contribute to sending messages, debugging multi-client runs
- Mark - Downloading pieces and subpieces
- Nigel - Parsing torrent/compact, communication with tracker, merging code