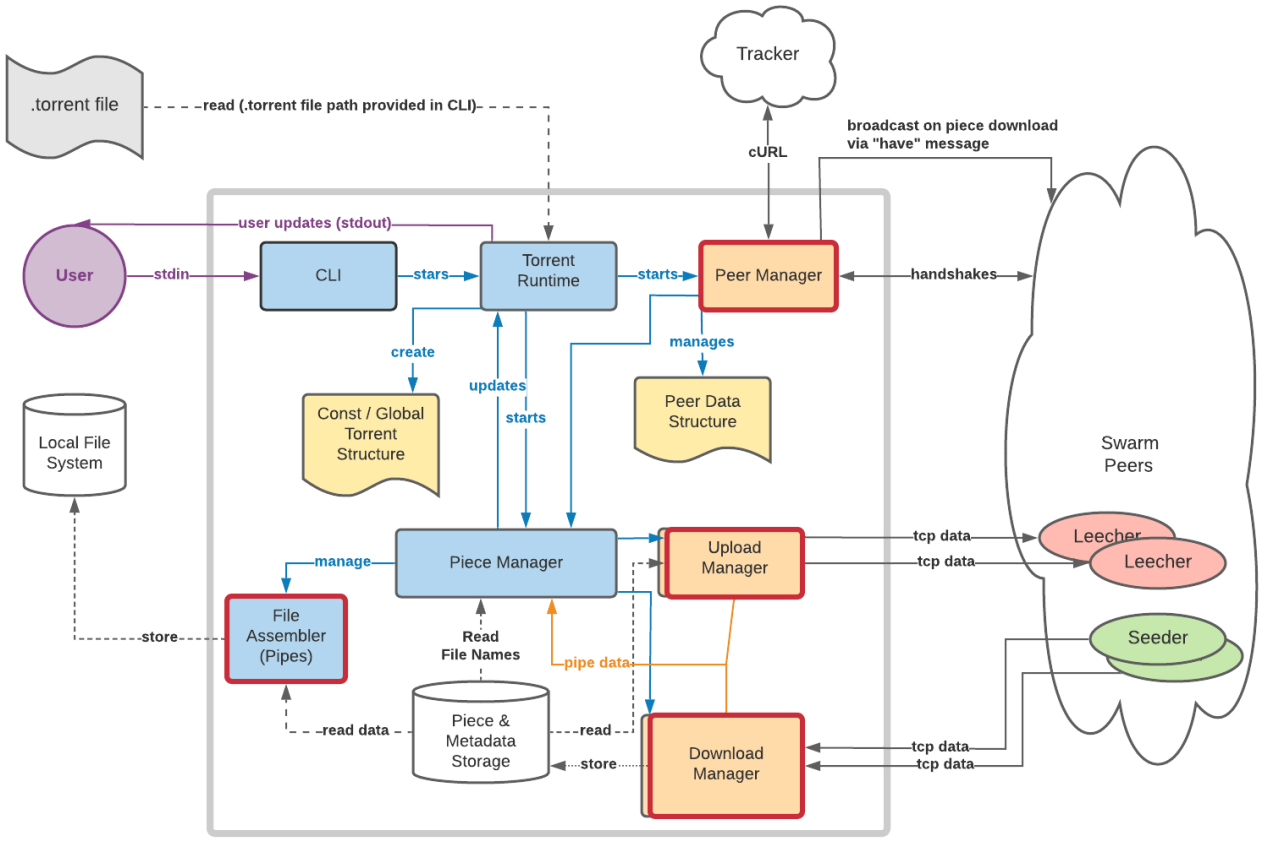
Bittorrent Report

Group Member: Dazhi Peng, Thong Do, Zachary Wynegar

1. List of Supported Features

- Connect to the tracker to request peers to communicate with
- Request and download sub-pieces of length 15000 max each time to complete download a piece from a peer.
- When the client shutdown before completely downloading the files, the client can restart with the pieces they had already downloaded
- Support using a torrent file that would download multiple files
- Create folder for files downloaded according to the paths in the torrent file
- Optimistic unchoking
- Uploading (not tested)

2. Design and Implementation



Questions:
Do you have to upload and download on the same descriptor / socket, or can you make one for download and one for upload?
What if a seeder is too slow at sending the file? How does it cut off?

The first thing we did on the project was diagram an architecture so our code was modular and we can collaborate without issue. When the program begins, it creates a new Torrent Runtime instance. This instance initializes all code and a global Torrent Structure. The Piece Manager is then initialized and reads from the file system all existing pieces associated with this torrent that was already downloaded. The Peer Manager is then initialized which is responsible for all peer communication. The Peer Manager implements the choking algorithm and calls a periodic function within the Piece Manager so the Piece Manger receives execution time. The Piece Manager runs the Piece Selection Algorithm to select a piece to download from a remote peer we are interested in which is unchoking us. The Peer Manager will communicate with a remote peer to initiate a download, and once we receive a piece, the Download Manager (in a separate thread), will download the piece and store it onto the file system. In the event of a peer disconnects, fails to send the correct amount of data, or the piece hash does not match the expected hash provided in the .torrent file, the Download Manager will throw an error message to the Piece Manager, which will propagate to the Peer Manager, and then sever the connection with the peer.

The program is written entirely in C.

3. Problems Encountered

One problem we encountered is how to use threads without needing to use locks to synchronize. Threads would be responsible for downloading and uploading sub-pieces to and from a peer. We do not want to use locks since debugging multithreaded applications is incredibly hard, and if we began using locks in one part of our code, due to the chaining effect of locks, we would require locks in nearly all our code. Our solution was to use a pipe to allow the threads to communicate the success or failure for uploading or downloading sub-pieces and download rate to the main threads. The main threads would have a periodic call that would check the pipe and read data from any child threads, similar to data from a peer.

A second issue was we initially designed the application to request a whole piece instead of sub-pieces. It was only a few days before the deadline that we found that peers will only send a piece of size at most 16000. This requires us to change the data structure for the peers to keep track of the piece index and sub-piece index. Besides, the code for requesting and downloading a piece, and the response to a complete downloading of a piece need to be changed to allow sub-piece

A third issue we had was that the peers would choke the client when the client only needed the very last piece to completely get the file. In this case, all peers choked the client and the client has no one to request the last piece from. We solved this code by going over the request message and found that the initial request message was requesting a sub-piece of fixed size, 15000 bytes, which was not guaranteed for the last piece. We solve this issue by changing the request piece size to reflect the needed and allowed byte size.

4. Known Bugs or Issues

We can run the 1056.txt.utf-8.torrent and 1184-0.txt.torrent but is not able to run the filezilla_multifile.torrent. We get an unsupported protocol error since they were not able to interrupt the protocol part of the URL.

We didn't have testing for the upload. Also, we were not sure how to test it with the tracker since we were only able to connect with peers who are seeders.

Also, we implemented the rarest first algorithm but we are not able to test if it works since we were only able to connect to 1 or 2 peers that the tracker sent us. Also, those peers that accepted the connection were seeders which lead to our application downloading pieces in order.
We are not sure about how to become a seeder though we did send the correct downloaded number of bytes information to the tracker.

5. Contributions made by each group member

Dazhi Peng worked on implementing the peer manager which would communicate with the tracker and read and send messages to other peers. The peer manager would also keep track of information about peers like a socket, choke status, and interested status. The peer manager implements the choking algorithm using the download rate. Also, the peer manager would have periodic functions that would check the piece manager.

Zachary Wynegar did the initial diagramming work and helped to build the architecture. Also worked on the upload and download manager which handles sending and downloading sub-pieces to and from peers. The upload and download manager would run on individual threads for each sub-pieces and would use a pipe to communicate status. He also works on reading the torrent file using the bencode library and storing the information into a data structure to allow easy access to information like piece length, file path, and piece hash.

Thong Do worked on the piece manager and file assembler. The piece manager keeps track of the peer and piece the application is currently requesting or sending sub-pieces to. The piece manager would read the information that the download and upload manager threads send and process it by sending a notification to the peer manager. The piece manager would implement the rarest first strategy where it will find the unchoking peer with the rarest currently accessible piece. The file assembler combines the pieces to make the files.

All team members worked on parts outside their particular area to fix bugs in the application and ensure there was overall integration between the components.