Jamie Liao/157006320/jl1806
Jungsoo Park/157005472/jp1326
IT Project Phase 3

# How the Project Works

We first opened the .torrent file to then read the contents inside of the file using the getTorrentInfo() method and stored the information in the TorrentInfo class to be used later. Then we created a HTTP GET request to the URL stored it into the .torrent package. Then, we stored the input stream of bytes to get all the information about the peers and select which peer, specifically those that starts with “-RU,” and narrow down that specific peer’s information. From the list of all peers that start with “-RU,” we then calculated the peer with the lowest RTT by taking the average RTTs over 10 trials. We then connect to that specific peer.

We then separated the port and IP from that specific peer’s information to make a socket to that specific port and IP address. Upon making the port, we sent a handshake to the remote peer that followed the BitTorrent protocol handshake by creating a Handshake object and passing in the relevant information to verify the handshake’s received and expected information.

Upon confirming the handshake, we then continuously read in bytes of piece length. We send the "interested" message to the remote peer, and wait for the "unchoked" message response. If the message is unchoked, local host requests the files and continuously takes in information while comparing the checksums of the pieces and the ones given in the .torrent file. We continuously write to the same file as each piece comes in. If there is no error in the comparisons through the entire process, we then tell the tracker that the download has completed and close the open sockets.

Phase 3 modifications were made with the use of multithreading. The main changes can be described as the following:

1) Taking a simple user input, “quit”, and then save progress
2) Periodically announce statistics to the tracker
3) Upload to multiple peers
4) Download from multiple peers

All of these, as one might imagine, needed to run concurrently with each other. (1) is done by InputThread and Save, (2) is done by TrackerThread, (3) is done by UploadThread, and (4) is done by PeerThread.

The main method in RUBTClient starts a single instance of InputThread and ThreadConnection. Then, ThreadConnection is where the rest of the Thread instances are handled. ThreadConnection spawns as many UploadThread and PeerThread instances as needed. All of these different threads interact with the same torrent via having a custom Torrent instance passed to it. All of them have access to the torrent's status and information.

A rough diagram describing the relationship is as follows:

RUBTClient -> InputThread -> Save
\------> ThreadConnection ---> UploadThread
\----> PeerThread
\---> TrackerThread

# Classes

- Bencoder2.java
  - Decodes the response from the HTTP GET request. Because the request is in bencoding, which formats a string of text in a unique way, so this class helped us to decode that special formatting. This class includes a decode() method that allows us to decode the message to store it into a Map<ByteBuffer, Object> object that stores many peers' information.
- BencodingException.java
  - This is an exception object that would be thrown if there is some error with decoding the bencoding string or trying to create a new TorrentInfo object.
- ToolKit.java
  - Methods to help debug decoding the Bencoding string from the HTTP GET request. The methods help us print out the information that is contained in the List objects, Map objects, Hashmap objects, ByteBuffer object, and Integer objects. This class is a useful way to help us "see inside" what is going on inside an object and already provides printing methods that we probably would've coded ourselves.
- TorrentInfo.java
  - A class that contains the torrent information after being read after decoding the torrent metainfo.
- Peers.java
  - This class contacts the tracker by sending an HTTP GET request to a certain URL. It then decodes the information gathered by sending the information to the

Bencoder which then gathers a list of all Peers via the makeConnection() method.
    - It also then queries all of the peers starting with –RU and stores that in another list of peers with just –RU via the method findPeers()
    - The class has an option to find the lowest peer via the public method findLowestRTT() which calculate the average RTT of each peer by adding up the total RTT over 10 pings. This method is utilized in the main method and then we open a socket to connect to the returned peer.
- Handshake.java
    - Handshake.java makes a handshake object that stores all of the relevant information required of a handshake from the peer. It reads in information from the peer and confirms that the info_hash and peerID are valid per the host and the peer.
- RUBTClient.java
    - If the peer was found and the handshake was made, then the client checks to see if the peer is interested and unchoked. If those conditions are met, the client then prompts the download for the file and writes to the file after every piece has been received and checked against the piece checksum. The client keeps track of how long it takes to download the entire file and print it out at the end. After the file has been successfully received, then the client closes all opened sockets, FileOutPutStream, DataInputStram and DataOutputStream.
- PeerThread
    - A thread downloading from a single peer. Uses the Torrent object to know which piece to download next.
- Piece
    - An object representation of a single piece.
- Save
    - Code that handles writing a file that hasn't completed. It implements the Serializable interface.
- ThreadConnection
    - Handles all of the different threads that connect to another host one way or another, i.e. tracker and peer communication.
- Torrent
    - An object representation of the torrent being downloaded. Keeps the table of pieces that needs to be downloaded.
- TrackerThread
    - A thread running from ThreadConnection that handles periodically announcing to the tracker.
- UploadThread
    - Similar to the PeerThread, except now it's uploading rather than downloading. Listens for a connection and then uploads the requested pieces.