Ethan Berroa (erb86)
Ivan Chio (ic165)

Phase 3

Description:

Our program, RUBTClient, runs with 2 arguments, the name of the torrent and the file save name. The base of the program runs the Torrent Handler object, which starts by reading the torrent file. It is responsible for handling/ coordinating the different instances of each class and tasks relating from the peer, tracker and messages. It follows a flow of direction from the torrent handler, where as it will download from multiple peers. In addition, if one wants to pause the download simply input "quit" in the terminal or close the window itself. If one wants to continue the download, one simply reruns the program

LAYOUT:

First, TorrentHandler decodes the information from the torrent file using "GivenTools" package. Once decoded it creates a Tracker object from the extracted info, a buffer to hold completed pieces, and a Writer object to write those completed pieces to disk. Then the Tracker builds up the announce URL, establishes the HTTP connection, and then stores the trackers response in a decoded Map for easy access. By using this decoded map, TorrentHandler creates a peer list with peers that contain "-RU" in their ID. In order to connect to multiple peers in an instance, we created a peer object, which the torrent handler can access in order for multiple connections. When a peer is first created, it creates a handshake object and tries to implement the handshake with the remote peer. If it succeeds, it tells the TorrentHandler using the peer delegate interface and now our send/receive loop can begin. The Peer starts reading on it's socket buffer. The TorrentHandler will check the SH1 hash of any completed pieces the Peer has passed up to it. In order to successfully download from multiple peers, we included a method to keep track of each piece to avoid collision. Once the download is complete, TorrentHandler lets the Peer disconnect, uses Writer to write the file in its entirety, and then uses Tracker to announce the completion and subsequent stop.

CLASSES:

RUBTClient.java (main file):
This creates the TorrentHandler, from the shell arguments, which is in charge of operating the rest of the other operations for program. In order to connect to the peers, the generatePeerID() method, created a random peer during the initialization for connection. This also includes reading "exit" from the terminal to close all connections and terminate.

Peer.java:
This is an abstract version of a P2P(Peer-to-Peer connection). The peer maintains information about the connection state, maintains the flow of sending and receiving messages, and has an interface to communicate to the TorrentHandler,known as a PeerDelegate. This interface is the link between Peer and TorrentHandler, therefore it allows the TorrentHandler to verify pieces upon each successful receive, as well as closing the client once completed.

TorrentHandler.java:
This is an abstract overview of the torrent connection. Given a path to a .torrent file, it's in charge of basically monitoring the entire download operation. It will decode the torrent file using the provided tools, create a Tracker instance to communicate with the tracker, and create the chosen Peer from the peerlist. Upon completion of each piece it is verified against its own hash (to ensure that the correct piece is downloaded, utilizes a Writer object to write the file to disk, and contacts the tracker to notify that the piece is successfully downloaded.

Tracker.java
This class used to communicate with the tracker and the torrentHandler, it does do by creating an instance of this object by using the information extracted from the torrent file. The object is in charge of building the announce URL, opening a URLConnection, and then extracting the trackers response into a map. The TorrentHandler also uses this class to initiate communication with the tracker, as well as terminate, by notifying the tracker that file is completed.

Message.java:
This class contains all of the enumerated message types and handles all the information needed to encode and decode peer messages. Encoding is done by constructing a byte array using the type of Message we want (HAVE, PIECE, UNCHOKE, etc.) and the "payload" array (referred to as "tail" in the code). Whereas, decoding is done by creating a MessageData object using the types defined here and then reading from MessageData members.

ListenerServer.java:
This class is an abstract view of the uploading process. It consists of various methods for receiving incoming connections between incoming peers. This class with the conjuction with the TorrentHandler, helps with uploading/seeding to various incoming peers

MessageData.java:
This class contains the messageData objects whereas it maintains connections between the peers and torrentHandler, and constructors for creating messages. To enable encoding it requires the use of Message.java. This class in conjunction with Message.java helps with decoding the messages recieved.

Event.java:
This is an abstract view of peer events, where each event correlates to the creation and handling of the multiple connection. Since connection with multiple peers is required for faster downloads, this class creates a peer event queue which it passes to PeerRunnable. These events are crucial, since it helps keep track of each pieces to avoid collision..

PeerRunnable.java
This class creates two threads for the peer, a sending/receiving thread and a writing thread. With these 2 threads it allows us to download with connections with multiple peers.

Handshake.java
This contains the Handshake object , constructors, and encoder/decoder for the handshakes. The handshake object allows us to communicate with the peer and download each peer.

PeerDelegate.java
This is an Interface that TorrentHandler implements to allow "communication" between it and the Peer. This allows our Peers to invoke methods in TorrentHandler upon certain conditions, like upon successfully performing the handshake or upon reading a Message.

Writer.java
We have utilized this to write to our new file, this helps to avoid any errors to writing the file to the disk. Therefore we use this class along with the TorrentHandler to write our completed and verified pieces to disk.

TorrentSession.java
This class includes the methods that assist in maintaining information about the torrent session in a torrent file. It is beneficial in order for us to keep track of each method, also allows us to easily examine the program itself.

Bitfield.java
This is a helper class meaning it helps parse and maintain information about the torrent file's bit field. It consists of various methods in order to correctly decode the incoming bitfield message.

PieceIndexCount.java
This class is the helper methods for TorrentHandler. Its purpose is to keep track of each piece index since, we are downloading from multiple peers, it helps avoid any complications.

COMPLICATIONS:

While we worked on this phase, we ran into various problems. One of the biggest hurdle was communicating and downloading from various peers. To implement this method correctly, we created a method that threads for reading and writing, instead of one for each peer. We have done this because there could be a possibility of the peers being choked, and if we threads for a choked thread, it would increase the space complexity within the process. It would significantly slow down the download process. Also to avoid collision for the pieces, we have included a method that keeps track of each piece, this helps ensure that we have the correct piece from each peer. The second problem was creating a method which handles the uploading to various peers after download. For this we have included methods in the Listener server.java, which utilized already existing methods, to correctly upload to incoming peers.