Marinela G. Haldeman (RUID: 142004774)
Aditya Sai (RUID: 125006921)
Elliot Goodzeit (RUID: 054001281)

Project 3 Write-Up

Submitted here is a peer-to-peer client using the BitTorrent protocol, written in java. The client now implements a rarest-piece-first algorithm for piece selection, as well as a method for throttling peers by choking and optimistic unchoke. Included in this write-up is a description of each class, and how interactions between these classes allow the program to function correctly.

The program is initially run from the command line, taking two arguments: the torrent file (*.torrent), and a name to save the file as. The RUBTClient class contains the main method. Main() loads the torrent file from command line argument. If it is not found, the program exits. Likewise, if there is no argument for the file name, the program will exit.

Manager.java contains the most important thread of the application. It's the decision maker. It has references to RUBTClientInfo, and upon instantiation it creates a tracker object and starts it. Manager keeps track of connected peers, and these peers report back to the manager using their specific interface listeners. The manager also contains a thread that listens on a socket for incoming connections. When an incoming connection has arrived, a new Peer object is created and added to the list of connected peers. The manager also listens for when the user has requested the program shuts down, and does so gracefully.

The program uses the observer design pattern, with appropriate listeners to that allow the subjects to notify the manager of any status updates. All listener methods are kept in a RUBTClientListener interface.

Tracker.java contains all the code for handling interactions with the tracker. It contains a thread that sends an HTTP GET request to the tracker every interval seconds for updates on the list of peers that are online. It then notifies the manager that a fresh list of online peers has arrived through the appropriate listener.

When the list of online peers arrives from the Tracker, the Manager goes through the list, finds the correct peers to connect to and tries to add it to it's list of connected peers. Upon adding a peer, the manager first checks if the peer is a duplicate (it is already connected). If the peer is a duplicate then it tests the peer is still connected and if it is it uses the old thread. If the peer is not a duplicate, it attempts to handshake with the peer. If the handshake is successful, it sends a copy of the client bitfield to the peer allowing the peer to know which pieces of the file we currently have. It also adds the peer to list of connected peers, and starts the peer's main thread.

The Peer class contains a thread that is responsible to intercept messages from the peer it's connected to and pass them to the manager (using the MessageListener interface) and send messages back to the peer when the manager makes a decision. The peer stores the timestamp of the last message sent to the peer and if it has passed more than 100 seconds it sends a keepalive message to the peer. To encode and decode the messages to and from the peer it uses the Message class. The Peer class also contains all information on that peer such as the peer's current status (whether it's interested in what we have, or its choked), its bitfield, its peer id, port, ip etc.

The Message class represents a peer to peer message in the bittorrent protocol. It has methods that allow it to encode or decode messages. The messages are captured by the peer and sent to the manager where a decision is made (such as unchoke a peer, request a piece of the file, upload a piece, etc.). Peers handle uploading and downloading pieces. Full pieces of a file are sent to the manager where they are verified and saved to disk.

FilePiece is a class that is a representation of a piece of the file being downloaded. It contains the actual piece data (byte[] downloaded), as well as the index of the piece, and its size. It extends the RequestPiece class, which is a representation of a piece that we request. A RequestPiece is smaller, because request sizes can only be at most 2^14. The Request class holds information on the status of a piece that is being requested.

RUBTClientInfo is a helper class that extends TorrentInfo and has information on the file to be downloaded, as well as methods for checking the file on resume, and parsing the stats file.

Included in the program is code that was written by Robert Moore. Specifically code contained in TorrentInfo.java for parsing the torrent info file, the ToolKit class, code in ToolKit2 for bit-to-boolean[] conversions, and a random peerID generator method in ClientInfo.

Overall, the project went fairly smoothly. There were some spots in which we got stuck, and some hurdles we had to overcome such as coding a program that doesn't burn a large amount of CPU, but in the end we produced something that we're all proud of.