Jeffrey Rocha 135003441
Gaurav Kumar 122008144
Julian Modesto 133006963
Group 16

Project 2 Write-Up

RUBTClient creates the TorrentInfo object which lays the ground work for the Tracker class to create the list of Peers needed to begin interaction with. At this point, the client runs through the peers with the correct IP addresses, and generates and validates handshakes for each of those in the Peer class. At this point, it will take the decoded message using the read() method in the Message class. Depending on the message returned, whether with choke, unchoke, interested, uninterested, bit field, have, request, or piece, the client will respond changing its settings of receiving and sends pieces of data by sending choke, unchoke, interested, or uninterested. There will be multiple peers running simultaneously now using threads, thus the client must be able to keep track the pieces downloaded from each peer, and put them together to successfully create the file being downloaded. MessageTask is used as a container to organize each peer with the message it has sent in order to keep structure with the multiple peers.

RUBTClient – Where the main method is located to run the program, where most of the action is going on in this program. run() deals with actually beginning the connection between the client and the peers, addPeers() filters out any IP addresses that are invalid, putMessageTask() places the task into a queue for processing, chooseAndRequestPiece() allows the client to find which piece it needs and send the message back to the peer, amInterested() discovers if the client wants to download a specific remote peer, shutdown() essentially kills the connection to the peer, and generatePeerId() generates the randomized peerID the client will be using throughout the program.

Peer – Deals with the messages sent to and from the peers talking with the client. sendMessage() sends the provided message to the remote peer, connect() opens up the socket and data input/output streams for data to be exchanged, run() calls connect() and handshakes are generated and validated in order to begin the exchange of messages, disconnect() disconnects the socket and closes the data streams, checkAndSendKeepAlive() sends a keep-alive message to peer if it was timed out, getHandshake() sets up the handshake between client and peer, and validateHandshake() checks the handshake for equality.

Message – Represents the peer messages that occur after the handshake, used to encode and decode peer messages. read() reads the Peer message passed in, and determines which type of message is sent and what to do according to what message was sent, write() writes out a message to the output stream. PieceMessage, RequestMessage, BitFieldMessage, and HaveMessage are separate classes created in Message to differentiate from the messages sent.

Tracker – where the tracker interface object is created. Here is where the URL connection is set, and decoding and bencoding also occurs here. Mainly, this is also where the list of peers is created and used by the client. The primary method in the Tracker class is announce(), which creates the HTTPGetRequest, opens the URL connection, reads from the input stream, and creates a hash map of all the data received from the input stream. This map is then filtered through using the ByteBuffer keys declared in Tracker, and eventually sets up the min interval for the process and allows for the creation of the list of peers that will be used to talk to the correct peers.

MessageTask – created by Rob, used as a container to hold messages with their corresponding peers in an effort to have organization in the number of messages received. This is used in RUBTClient for processing purposes. Just some simple setter methods to assign peers and messages.