Andrew Lee (al696) – 144008285

This program performs the function of a Bittorrent client by maintaining connections with the tracker and a peer, parsing and sending messages as specified by the Bittorrent protocol, verifying pieces of data using SHA1 hash techniques, and reading and writing data to and from an output file at random points in the file. The main execution of the program begins in RUBTClient which is responsible for parsing command line arguments about the files used in the download. It creates an output file if it does not exist, and if it does exist, it reads from that file to determine pieces that were downloaded before and verifies them in a bitfield, if any exist. Then it will start a TorrentClient object which is the main overseer for the tracker and peer connections. TorrentClient is responsible for handling information between the TrackerConnection and Peer classes which are connections to the tracker and remote hosts, respectively. Once it contacts the tracker and receives a list of peers, it determines which peers are suitable to connect to and performs ten ping operations to them to determine an average RTT. After calculating the peer with the smallest RTT, it then commences the download procedure.

After the client has sent handshake messages with the peer and verified them, it sends a series of messages in order to retrieve data about the file. These messages are defined in the Message class with several static classes representing the messages used for this project, such as BITFIELD, UNCHOKE, INTERESTED, PIECE, HAVE, etc. Messages are coupled with a Peer using the PeerMessage class in the future that downloading from multiple peers should be supported, otherwise they identify an incoming message with a peer. Messages are stored in a queue which are then read in a FILO order. The connection to the peer first exchanges bitifield messages, and then the local host sends an interested message and waits for an unchoke message before sending requests. The local host sends request messages using 2^{14} length blocks as specified in the Bitttorrent protocol, and after receiving two blocks to comprise a full piece, the client verifies the piece with a SHA1 hash function against the metainfo file, and then it writes it into the output file using a RandomAccessFile. When the client has completely downloaded the file, it outputs the total time taken to download in nanoseconds and seconds (typically a little less than 20 minutes), closes connections with the peer and tracker, and terminates execution.

HexBytes.java
This class provides a utility method to convert a byte array into a hexadecimal string. BytesToHex() takes a byte array and uses bitwise operations and shifts to change it into a hexadecimal format.

Message.java
This class defines the different types of messages used in the Bittorrent protocol and contains several static classes for communication between peers. Some message types, such as keep-alive, choke, unchoke, interested, and uninterested, have respective messages already defined as public static variables since the contents of the message do not change between users. For larger messages such as Piece and Have, the necessary payload information is encoded using an overridden method that changes depending on the message.

Peer.java
This class represents the connection with a peer and manages the message communication between both hosts. When blocks are received by the peer, the piece is reassembled, verified against its hash in the metainfo, and subsequently written into the file. Peer connections are threads, and the main command executed is decode() which parses a Message and responds accordingly. The client mainly downloads files, so after the initial handshake and interested message, it will request a block of a piece, receive the said block, verify SHA1 hashes, write pieces into a file, and send have message afterwards.

PeerKeepAliveTask.java
This class runs a thread that after sleeping for some interval, prompts the TorrentClient to send a keep-alive message to the peer to maintain the connection. Other classes with prompt the thread to interrupt and start again if another message is sent before then.

PeerMessage.java
This class is simply a wrapper class to associate a peer to a message for identification. While only one peer is used for connections for this phase of the project, in the future multiple peers would speed up the download. This class helps to identify incoming messages from specific peers and avoid misinterpreted operations.

RUBTClient.java
This class is responsible for parsing command line arguments, reading from file, and starting the TorrentClient object.

TorrentClient.java
This class creates, manages, and closes connections between the tracker and a list of peers that it maintains. It is responsible for initializing them which includes generating a peer ID to identify itself, receive data from a peer and perform SHA1 hash checks on pieces, and write verified pieces into the output file.

TrackerAnnounceTask.java
This class is used to send blank announce messages to the tracker after a specified interval in order to remain connected to the tracker. The TorrentClient object that starts this class will restart the timer if it sends some message before the interval has passed.

TrackerConnection.java
This class connects to the tracker, sends messages in order to maintain a connection, and retrieves the peer list for the Peer object.

Utils.java
This class provides several static methods which are used as utility methods relevant to the operation of a Bittorrent client such as parsing a bitfield byte array into a Boolean array and hashing pieces to verify their integrity.