Group 22
Manan Doshi - 145000508
Mehul Doshi - 145000507
Christopher Klinchik – 141002235

## Classes:

Peer – Stores peer information in one object so that it may easily be accessed. It stores the port, IP, and the peerID.

Message – Stores the messages sent to and received from the peer in one object. It stores the actual byte message, and also the message type for easy reference.

Downloader - This class extends Thread, and is a thread that downloads from a peer. The constructor takes in a peer, TorrentInfo, and a byte array. After getting the peer list, a Downloader array is made, and a Downloader is made for each peer and added to this array. It utilizes the ReentrantLock class to lock and unlock, so that multiple threads are not accessing the same objects at the same time. The downloaders first send the peer a handshake. After receiving their respective peer's handshakes, they verify the hash, and begin messaging by sending repeated 'interested' messages and 'request' messages. This is done in Downloader's download() method, which loops until all pieces are downloaded in the run() method. After all pieces are downloaded, the RUBTClient class stops all the Downloaders that are still running.

Uploader - This class extends Thread, and is a thread that uploads to a peer. The constructor takes in a peer, TorrentInfo, and a byte array. After getting the peer list, a Uploader array is made, and an Uploader is made for each peer and added to this array. The uploaders first send the peer a handshake. After receiving their respective peer's handshakes, they verify the hash, and send a 'have' message so that the peer knows which pieces we have. Next, it loops and waits for 'interested', 'request', 'bitfield', or 'keepalive' messages. This is done in Uploader's upload() method, which stops only once the user inputs '1' to terminate, or if all the socket connections close because no one is downloading from us.

Terminator - This class extends Runnable, It's sole purpose is to keep track of whether the user wants to pause or quit. If the user enters '1', the program terminates and all threads are stopped. If the user enters '2', the program suspends all threads, and resumes them when the user enters '2' again.

Tracker - This class extends Thread. It was created to store information about the tracker, and also to communicate with the tracker. The constructor takes in the scrape URL, min, and regular intervals and performs a tracker scrape by sending an HTTP GET request to the scrape URL.

RUBTClient – This is the main class that does all the client/peer connections and communication. After taking in the .torrent and .mov file name in arguments, it opens and decodes the torrent file using Bencoder2. It then sends a GET to the tracker and decodes the list of peers. It chooses the given peer that has the file, and creates a Downloader thread to download it. It also creates Uploaders after the Downloaders have been created to then start uploading to other peers. This class also has many helper methods like whatNext(), parseMessage(), parseHex(), and setHex() which both threads can use for the messaging part. The method addPiece() takes the ArrayList of downloaded pieces (which is a global variable) and puts them all into the file with the given file name. The global int numActiveThreads tracks how many threads are still active, and only stops running once all started threads are inactive. This is very important to ending the program properly. Also, in the main execution block, a Terminator object (called terminator) loops and reminds the user that they can quit with '1' or pause with '2', but automatically quits the program if all the uploaders and downloaders have already finished.

## Class Interactions:

The RUBTClient interacts with Peer in both the extract() and download() methods. The Peer object stores peer information, and uses it in the download method to create a socket and establish a connection to the peer.

The RUBTClient interacts with Message using the message() method, after the socket connection and handshaking with the peer is already done. The message() method is used to easily call the correct message that is going to be sent to the peer. For example, message(1,2) returns an 'interested' message.

The RUBTClient also interacts will all of the given classes. It uses Bencoder2 to decode the .torrent file, and uses BencodingException to catch errors that may occur with the Bencoder. The TorrentInfo class is used in the download() method. The method takes a torrentinfo object in, and can then access all the information and parse it. Finally, the ToolKit is used by RUBTClient to print out byte arrays and ByteBuffers.

The RUBTClient interacts with Downloader and Uploader when messaging. After getting the peer list, it creates Downloaders and Uploaders to interact with the peers, and only ends once all

the Downloader/Uploader threads are no longer active, or if the user inputs '1' to Terminator. These two threads are the crux of the entire program.

Downloader and Uploader both interact with Peer. The constructors for both require a Peer, so that it knows which peer to download from or upload to, and can easily communicate with it.

Terminator interacts with Downloader, Uploader, and RUBTClient. It needs to interact with RUBTClient because that is the main program, and it needs to exist within it to ask the user whether they want to quit or pause. It also interacts with Downloader and Uploader, because if the user wants to quit, those threads all need to be stopped as well. Of course, those threads also need to be suspended when the user wants to pause.

Tracker interacts with RUBTClient. RUBTClient provides the min and regular intervals to the tracker class as well as transforming the announce URL provided by the tracker into the scrape URL by the scrapeURL class. Tracker is then able to perform scrapes.

## Workload Distribution:

Manan - Made Uploader and Downloader class. Worked on big download method, helped with hex encoding and messaging. Created and implemented locking. Created sockets and input/output streams for peer messaging. Got the pause/resume for terminator working. Fixed and optimized multi-threading, got program to gracefully end and prevented deadlocking. Made arrays of threads to easily stop and start them when necessary.

Mehul - Did most work in big download() method, whatNext(), addPiece(), and main(). Created the ArrayList that stores the pieces, and got it to put it all together and make the .mov file. Did most peer communication work. Did most work on big upload() method, as well as any helper method used by upload(). In general, did a lot of the messaging stuff for both upload and download, which took a while to get working. Did most of bit manipulation stuff.

Chris - Did the whole handshake() method, most of tracker communication, extract() method, Went through and organized some of the code and also added some more exception handling. Made the Peer class for easy storage of peer information. Created Terminator class and got the end program part for it working. Created Tracker class.

Note - We started using Github kind of late, so the contributions on Github may look skewed in my (Manan's) favor because I uploaded everything from stage 1 and 2 to Github. In reality, this was a good group and we all contributed equally to this assignment.