Group 16
Harshil Shah
Tedd Noh

Project RUBTClient – Part 2

Description:
Part 2 of this project extends the work done if Part 1. All the tasks (parsing the torrent file, contacting the tracker, connecting to and downloading from a peer, etc.) done by the Part 1 client is also done by the Part 2 client, but the Part 2 client performs addition tasks. In addition to the Part 1 tasks, the Part 2 client uses multi-threading to connect to multiple peers and uploads/downloads from them. It does this through the use of "controller" classes (Controller.java and IncomingController.java). Specifically, at the start of execution the main RUBTClient class creates an instance of the Controller class. For each peer that must be connected to, Controller creates a new instance of PeerController (which is a subclass of Thread), assigns it to a specific peer, and invokes its run() method. All instances of PeerController will then simultaneously communicate with their respective peers by using the Peer class (from Part 1). Our client also is able to accept incoming connections from new peers using the IncomingController class. IncomingController essentially acts as a server that responds to incoming peer connections. For each incoming peer connection, Incoming Controller will create a new PeerController and assign it to the incoming peer.

Classes:
RUBTClient.java
- Starts up the main client.
- Parses the torrent file.
- Creates Controller thread.
- Provides methods for file writing (using FileChannel for random access)
- Maintains state information and saves it to a file upon normal program termination

Controller.java
- Performs initial handshake and verifies handshake response with peers
- Creates a new PeerController object for each outgoing peer connection it has a waiting queue of peers, waiting to be assigned a PeerController.
- Starts up the Tracker Controller
- Controls all threads.

PeerController.java
- Responds to messages from its assigned peer
- Can upload and download pieces

IncomingController.java
- Acts as a server to respond to incoming peer connections
- Verifies incoming handshake and responds with its own handshake

• Creates a new PeerController object for each incoming peer connection

Peer.java

• A prototype of a peer.

• It has methods to perform all peer messages like handshake, alive, interested, and other messages.

• Has the ip and port information for the peer.

Tracker.java

• A prototype of a tracker

• Has the necessary keys to retrive data from the tracker's response dictionary.

• Has one method called connect which takes in the necessary information to make contact with
the tracker.

• It has the trackers annnouce url

TrackerController.java

• Uses the Tracker class to communicate with the tracker

• Contacts the Tracker at regular intervals

• Sends messages such as “stopped”, and “completed”

Miscellaneous:

– The documentation on the BitTorrent protocol is very well documented and easy to find.

– The challenges faced were to write downloaded pieces to a file using FileChannel for random access. Also converting a BitSet to a byte array was an issue.

– For the future: It is possible to modify our code to accommodate multiple torrents each with their own tracker and peers.

Please Note: The client will continue to run even after it completes its download (waiting for piece requests). To properly terminate the program, you should type “q” or “quit” in the command prompt. Upon re-execution, our client should pick up from where it left off in the download process.