Richard Gerdes
Rahul Purwah
Robert Williams, III

CS 352: RUBT Phase 1

The Program:

Our implementation of RUBT is a rough work in progress. The program initially creates an empty file to store what we download, once the download is finished. Next we create a TorrentInfo object to parse through the torrent file and prepare for information gathering. Our PeerHost class then comes into play and begins attempting to make a connection the the host server listed in the .torrent file. The Tracker class handles a the bulk of the HTTP related work. It connects to our PeerHost and begins to pull the information from the Web Server so that we can gather the Peer information , create Peers and store them into our Peer array which we then use to try to make connections and thus begin our download.

This is controlled by the TorrentHandler class. Torrent Handeler goes through our list of Peers attempting to make a connection to one so that it can begin the handshake. After the handshake. The client begins to download the parts of the file, then once it's done, we disconnect and enjoy our new file.

The Classes:

RUBTClient:

The Driver that runs the program. There isn't much more it it than that. It just gets things going.

PeerHost:

PeerHost creates a socket connection to the host server which is then passed along to the tracker for use.

Tracker:

The tracker performs the first bulk load of work. It performs the HTTP Get requests and obtains the list of Peer IPs that the user is supposed to attempt to connect to. It then creates Peers and sends the handshake message away.

TorrentHandler:

The Torrent Handler does the next heavy lifting phase of the project by doing the work to download the file. The Torrent Handler sends out a request for blocks of data, which It then

saves the pieces into a FileBuilder.The TorrentHandler also checks that a peer is still alive and well.

Peer:

The Peer class abstracts the client's connection to any single peer. It is a wrapper class for the information and socket that define what a peer is. We use Peer to send our messages to and from Peers by working closely with the message class.

Block:

Block is a data wrapper for bute information that is send from the Peers

Piece:

Piece is a block wrapper. Piece also validates the data that is received from the Peers against the SHA-1 Hash

FileBuilder:

This builds the file that the peer(s) have sent us. The data hierarchy goes, Blocks build Pieces, Pieces build files. FileBuilder assumes that all data has been validated before being sent, and thus simply constructs a file out of the given bits.

Message:

The message class contains all base code for the generation of byte[] messages that will be sent to the other tracker. While the message class performs validation checks for the Handshake, it leaves the checking of pieces to the Piece class.

Feedback

One of our biggest issues with creating this project, aside from poor time management, was the seemingly lack of documentation. The provided materials and protocols were great at providing us with the concept of how to implement a Bittorrent at a very high, abstract level, however when it came down to actual implementation, we found that there was very little written guidance, and this made it difficult to find direction. Much of our knowledge came from picking at the brains of those who have come before us, but it feels terrible, personally, to repeatedly ask for assistance from others on the same topic. (Makes me feel like a nuisance)