Julia Sutula (js1592)
Stephen Jin (sj408)
Sehaj Singh (ssb116)

GITHUB LINK TO CODE: https://github.com/stepjin/BitTorrent-Client

BitTorrent Phase 3 Readme

Our implementation of the BiTorrent Client starts off by parsing the torrent file using the given TorrentInfo class. After decoding the torrent data, the Tracker class comes into play where it does most of the HTTP related work like sending a GET request to the tracker using the IP and port specified within the torrent. This allows us to gather the peer information by extracting a peer list from the tracker. We then connect to the peer by using the BT peer protocol and a TCP socket. After a successful connection, the client sends a handshake and verifies it so they are ready to exchange messages. The peer class works closely with the message class as it is used to send messages to the peer in order to download and upload necessary pieces of the torrent file. The peer class runs as multiple threads, with a thread for each peer. These threads use shared and synchronized methods in order to properly ensure simultaneous uploading and downloading.

Since Phase 2, we attempted implementing a manager, gui, the rarest piece algorithm, and optimistic choking and unchoking. Our attempts at these are included in our code. The rarest piece algorithm and optimistic choking and unchoking successfully worked. The gui was also partially implemented (but not added in to the final result, just included into the project file to show for the work). We also were able to connect to multiple peers however many peers would send incorrect codes forcing us to close connections with those peers, so it was difficult to thoroughly test our program.

** In order to exit the client, the user must type 'exit'. To resume, the program must be started again using the same file name as the file to resume downloading.**

Class Descriptions

RUBTClient.java – This is the main class that takes in two arguments. The first one being the name of the torrent file and the second one as the saved output file. This is where we parse/decode the torrent file and check to see whether the peer_id prefix is the one we want. Essentially, a TorrentInfo object is created which contains the metadata of the torrent file. This metadata of the torrent file is then used by other classes in order to send and receive messages to extract the proper pieces of the torrent file. The client also spawns multiple Peer threads, and properly closes them out when the user exits the client. It also keeps track of the time it takes to fully download a file. It also spawns a thread to keep track of user input, which will be used in order to exit the client accordingly.

Public methods:
main, run() (for user input thread), parseTorrentFile(), getPeerMatch(), getFileName(), getPeers()

Tracker.java – Performs the HTTP GET request to the tracker so we can obtain an ArrayList of peers that the user is can connect to. It ensures that the tracker announce request url is properly encoded. This class is also responsible for generating a peer id and making sure that it is indeed unique. The tracker class also contains method for URL encoding as well as SHA-1 hashing
.
Public Variables: All constant byte buffers used for retrieval of tracker fields

KEY_INTERVAL, KEY_MININTERVAL, KEY_PEERS , KEY_PEERID, KEY_IP, KEY_PORT

Public Methods: all used by peer to successfully decode tracker and verify hashes
connectTracker(), createURL(), decodeTracker(), makePeerID() , uniquePeerId(), bytesToHex() , computeSHA1Hash(), getPeerId() (for retrieval of private variable) setPeerId(), setEvent(), setRequestString()

Peer.java – This class essentially connects and maintains communication with each peer. Each peer works closely with the Message class to send and receive messages to and from the peer. It executes a run method, where the exchanges of messages occur. Run() initially sends the handshake and verifies the responded handshake. Afterwords, the client sends a bitfield, and also extracts and decode a bitfield message sent by the peer. If the bitfield contains a piece that is needed a message is sent indicating interest in the peer's pieces. If the peer is interested in a piece the client has, the peer indicates interest, is unchoked, sends a request, and then is sent a piece from the client. The client performs rarest piece algorithm in order to request pieces. If pieces were requested by the client and sent to the client from the peer, then an exchange of messages in order to extract pieces of the torrent file occurs. The byte array of each block is appended to an output file until each piece necessary is successfully obtained.

Public variables: all constants used in message sending
keepAliveLength, keepAlive, chokeLength, choke, unchokeLength, unchoke, interestedLength, interested, uninterestedLength, uninterested, have, request

Public methods: all used in order to create and maintain peer connections
Peer constructor, connectPeer(), sendHandshake(), verifyHandshake(), handshake(), keepAlive(), choke(), unchoke(), interested(), uninterested(), extractPiece(), requestPiece(), verifyHash(), decodeIncomingMessage(), decodeBitfieldMessage(), decodeRequestMessage(), sendPiece(), getPieceFromFile(), decodePieceMessage(), byte array conversion methods, writePiecetoFile(), along with multiple accessor methods for the private variables.

Message.java – The message class declares the various types of byte[] messages that are used with the peer class allowing for interaction between the peers. It also contains static extensions of the message class including including have, request, bitfield, and piece. Messages here are formatted following <length><messageID><payload>, with the payload only applying to messages of the type have, request, and piece.
Public variables: all constants used for sending messages
keepAliveLength, have, request
Public methods: have message constructor, request message constructor, makePayload() (for each message type), piece message constructor, bitfield message constructor, sendMessage(), sendKeepAliveMessage()

PeerList.java-- This simple class provides access and modifier method for the list of Peer threads. This is used in Peer in order to send have messages to all the Peer threads accordingly.
Public methods: access and get methods for private variables peer list and resume

Manager.java- This class acts like a manager and implements the optimistic choking and unchoking of peers.
Public methods: access and get methods for private variables
PeerConnections.java- This class was our attempt at accepting incoming connections from peers

Gui.java- This class is our attempt at a gui for the extra credit, only partially implemented

Feedback:

Finishing this project was very difficult due to lack of testing ability. Many peers with which we tried to connect to, and there weren't enough peers that worked to thoroughly test our program.

Work Division:
Note: All team members contributed to planning and coordinating execution of the project. We all sat in the ilabs together working from the same computer discussing what to do throughout the duration of the project.

Julia- Manager(not fully implemented due to time constraints), optimistic choking and unchoking, multithreading coordination, checking for user input, gracefully exiting the program, resuming the program, timer, extracting bit arrays, modularizing the decoding of messages, piece requesting, piece downloading, getter and access methods
Stephen- worked on socket connection(werent able to finish due to time constraints)rarest piece algorithm sending haves to multiple peers, resuming the program, announceTimer, timing download, interval announces, converting bit arrays to booleans, bitfield class, tracker notification and communication
Sehaj- worked on gui (due to time constraints we werent able to finish this implementation) timing download, keep alive timer, readme, commenting and documentation