Phase 3 Write Up

Group 20: Aishwarya Gondhi (ag1201), Harshil Shah (hhs30), Krupal Suthar (kss123)

How to run the program

cd Group20
java -cp . bittorrent/RUBTClient Phase2.torrent <filename to save>

Brief Description of Each Class

RUBTClient
RUBTClient.java opens and reads the metadata file. It throws an exception if no file is found. It uses TrackerInfo to deal with the information inside the file and tells the TrackerInfo to make a GET request. It handles any exceptions or null values in response to the TrackerInfo's GET request. It also makes a file specified by user and writes the bytes downloaded from peers. It is responsible for spawning all the threads and is the controller of them. This class also stores numerous global static instance of few classes and some global static private data structures. **We added a private class called OptimizedChoking that determines which peers to choke and which peers to unchoke based on their transmission rate.**

MainView
MainView contains formats the GUI user interface. It creates a white, 900 by 600 bit panel titled 'RU Bittorrent Client'. The panel is created with a menu that has items such as file, options and help.

FilePanel
FilePanel gets images from the images folder. It creates start, stop, and pause buttons, and a progress bar. It has the actionlisteners for stop and start buttons, where users can stop the program and resume the program.

TrackerInfo
TrackerInfo.java has several purposes. It is used for retrieving tracker information from torrentInfo (metadata file) -- tracker ip address and port number. Once it gets this information, it sends a HTTP GET request, where it sends info_hash (from metadata file), peer_id, ip, port, downloaded, left and event to the tracker to get an array list of all the peers. It handles any exceptions thrown in the tracker response and ensures the tracker response includes peer ids, and the corresponding IP addresses and port numbers.

Download
Download.java was created to make requests to other peers for pieces that we don't already have. Download.java ensures the information in the piece is the information it wanted by comparing the packet number it requested with the packet number on the header of the piece we received. It also times the download, and outputs the total time of download. Once the packet is received, it verifies it's SHA-1 hash against the hash stored in the .torrent file. After the packet is downloaded and verified, it passes it on to Shared.java where it is arranged in an array with an index based on its packet number.

Upload
Upload.java was created to handle the requests for packets from other peers. It uses PeerMsg to communicate with these peers. It uses the requested packet number to locate the packet in Shared.java's array. It only makes a connection to peers with whom we haven't set up a TCP connection.

Shared
Shared.java was created to store the packets downloaded and to locate the packets requested. Shared creates an array the size of the actual file (and the number of packets).

PeerMsg
The purpose of PeerMsg.java file is to communicate with the chosen Peer regarding the file. Here, we start messaging the peer about the piece of the file that we need. There are different kinds of messages that we can send a peer: keep-alive, choke, unchoke, interested, and uninterested. We send the interested message to let the peer know that we want a piece of the file that it completed downloading, and when the peer unchokes the client, the client will start downloading that piece of the file.

HaveMessage
HaveMsg.java is an extension of the PeerMsg class. It is used to acknowledge after we recieve a packet.

RequestMessage
RequestMsg.java is also an extension of PeerMsg.java. It is used when we want a packet from another peer.

PieceMessage
PieceMsg.java is another extension of PeerMsg.java. It is used when we are sending a piece to another peer.

BitfieldMessage
Bitfield.java is another extension of PeerMsg.java. It contains information about all the packets that we have in the Shared.java. It contains a list of all the packets that we have downloaded and verified.

KeepMeAlive
KeepMeAlive was created for Upload. It tells the peer to keep the connection alive.

Announcer
Announcer is an independent class that is used to send updates to the tracker after each interval. It sends status information to the tracker over each interval including uploaded, downloaded, left, etc. When the entire file is downloaded and verified, we send completed event. If the client chooses to exit, stopped event is sent to the tracker. Whether stopped or downloaded completely, TrackerAnnoucer closes all connections.

InputListener
InputListener was created to monitor user activity. It continuously listens to check if the user has requested anything. If user requests to exit the program, it enforces the user's command by having Download save all the verified packets and discard any unverified packets. It also ensures that the TrackerAnnouncer sends a stopped event to the Tracker. When the user requests to restart the program, it ensures that everything starts up where it left off.

Constant
Constant is a Utility class that contains all the constants used throughout the program.

Converter
The Converter class is a Utility class that helps other classes with conversion from one object to another. Most of the methods are for byte conversions.

High-Level Description of the Program

The first thing we need to do is contact the tracker and get the list of peers. To communicate with the tracker, we need the IP address and the port number of the tracker. This information about the tracker is found in the TorrentInfo (metadata file). Hence, we shall first address how the we get Tracker information from TorrentInfo.

In order to get the information on the tracker, RUBTClients opens and reads the file to the TrackerInfo. To interpret the bencode in the TorrentInfo file, we used the Bencode2 file provided to us, if there are any exceptions thrown during this process, we are directed to the BencodeException file, otherwise the tracker information is returned to the TrackerInfo.

With the tracker information, TrackerInfo is told to contact the Tracker and sent a GET request. But to do so, it needs to convert bytes to URL which is done with the help of the Converter file. So, when we send the HTTP GET request, it is converted into a url by the Converter, then send to the tracker. Once, the connection with the Tracker is made and the Tracker gets the request, it sends TrackerInfo the peer list. The TrackerInfo then ensures that the tracker response includes all the elements (peer id, ip, and port) it needs to communicate with the peer.

With the information about the peers, we need to communicate with the Peers to download the file. To do so, TrackerInfo uses the Peer class which opens a port for communication and handshakes the Peer. Once the handshake is complete, it validates the response. Once we have the bitfield from that peer, we request all the pieces that the peer has so that we can finish the download as fast as possible. Then Download uses the PeerMsg file to send messages that correspond with the interaction it wants done between the Peers and the client. Once Download has the entire packet, it verifies it's SHA-1 hash against the hash stored in the .torrent file. After the packet is downloaded and verified, Download passes it on to Shared.java where it is stored in an array with an index based on its packet number.

Peers can request verified packets from us. First step is always making the handshake. When the peer sends an interested message, it responds by unchoking. Since we are using optimized unchocking, we are choosing the peers that have the fastest rates, so we randomly add one peer every 30 seconds, and let go of the peer that we have the slowest connection with. When the peer sends a request for a specific packet, Upload checks in Shared classes array if the slot that it suppose to hold that specific packet is empty. If it is empty, it sends the peer an error message, if not, it sends the packet to the peer.

So, we created a GUI user interface, to start the program. When the user clicks file, open, and selects the torrent file, the program starts. The GUI shows a progress bar of how much was downloaded, it contains other data such as amount of data uploaded, the number of the seeders, etc. It even contains a scrolling list of actions taken such as when a peer contacts us, when we handshake, etc. It also allows the peer to stop and resume the program. These are in the form of button on the top of the interface. Whent the user clicks the stop button, all verified packets are stored and all the unverified packets are discarded. The tracker is informed that we stopped. If the user clicks the resume button, the program resumes from where it left off and downloading and uploading will start based on the packets we already have stored. Once the download is complete, the user can see the path where the file was stored. To access the file, the user must first close the window (file, close) then follow the path to find the file.

Once the whole file is done downloading, all the ports for the peers it was communicating with, will be closed and once the Tracker is informed that the download is completed, the port with the Tracker will be closed as well.

Contributions

Aishwarya Gondhi: PeerMsg, RUBTClient, TrackerInfo
Harshil Shah: Download, InputListener, Announcer
Krupal Suthar: Upload, Shared, Piece

Dependency Diagram