Group 5:
Paul Tai
Alex Zhang
Anthony Wong

---

usage:

java -cp .src.RUBTClient project3.torrent outputFilename

Abstract Overview

RUBTClient, given a torrent file, will use Bittorrent protocol to connect to a tracker. Using the information provided by the tracker, it will establish connections to Peers and follow Bittorrent protocol to download pieces of the file. If the program is given an output file that is already completed or partially completed, it will resume the download or begin seeding (respectively).

Our client keeps track of the download and upload speed of any active peers and maintains the TCP connection until either the peer closes or RUBTClient shuts down. It performs optimistic unchoking on peers to (hopefully) increase file acquisition rate.

RUBTClient updates the tracker of it's current status based on the interval provided by the tracker and upon download completion or program exit.

Upon completing a download, it switches to seeder mode, and will attempt to prioritize high-downstream-Peers to improve swarm acquisition rates.

RUBTClient Classes:

RUBTClient.java:

This class primarily performs error checking, initalizing, and shutdown procedures.
It uses TorrentInfo to ensure that the given torrent file is a torrent file.
Calls the constructor for Client
Starts running the main thread of Client
If client is not seeding, RUBTClient will start the PieceRequester of Client.
Lastly, sets up the GUI and the shutdown hook.
When the quit button or the shutdown hook function is called, it will shutdown client.

Client.java

Client handles the internal state of the datafile and all of the data exchange between peers and the tracker.
It has two primary threads:

PieceRequester, which just has a LinkedBlockingQueue of peers and will continuously check for peers who are unchoked and not downloading a piece.

Main thread starts off by scheduling the timer tasks and then continously reads a LinkedBlockingQueue that contains all messages sent to the peer and then updates the internal state of the client and peer based on the message.
Client also has a few supporting threads and timer tasks:

ServerSocket, which will listen for incoming connections and spawn new Peers.
requestTimer, which will contact the tracker with an update every interval, based on what the tracker has set as the interval, as well as analyze and choke poor performing peers every 30 seconds.

Peer.java

Peer handles the internal state, data streams, and sockets of a peer.
It consists of a PeerWriter and a main thread.
The PeerWriter reads from a LinkedBlockingQueue any messages that need to be sent to the remote peer.
The main thread starts off by scheduling the supporting timer tasks, and then continuously reads from the incoming socket.
The timerTasks check for UL/DL rates every 2 seconds and check if it should send a keep alive every 10 seconds.

Message.java

Deals with the internals of a message sent between peers using the BT protocol.
Has a few public static variables that are used every time choke, unchoke, keep alive, interested, or uninterested messages are needed.

MessageTask.java

Wrapper class used to hold a message and the peer it's from in order to store it in the Client's incoming message LinkedBlockingQueue.

Piece.java

Class that handles a single piece while it is being downloaded from a peer.
Stores data and piece index in an internal buffer for the Client and Peer to interact with.

Tracker.java

Class that handles formatting HTTPGetMessages to the tracker, parses the response, and then returns it to the Client.