Christopher Rios
Ben Green
Manuel Lopez

Project Overview

The ‘main’ method of the program is contained in RUBTClient.java. Arguments taken in are the torrent filename and the desired output filename. The torrent file is parsed into a TorrentInfo object. Once parsed and our random access file is initialized (Destfile.java), our RUBTClient spawns the main client thread that will handle the main event loop of the program.

In the client thread, the connection listener (ConnectionListener.java), shutdown hook (ShutdownHook.java), and input lister threads are initialized. The tracker is then contacted with the start event by using Tracker.java. The extracted info_hash, our randomly generated peer_id, uploaded, downloaded amounts and amount left are composed into a GET request and sent to the provided Ip and port, also extracted from the .torrent file.

The response to the GET request is decoded and stored in a Response object (Response.java) which contains an array of Peer objects. After valid peers are extracted from the list of peers, RUBTClient creates a new Peer thread for each valid peer. During this process, each separate peer thread does various checks to vet the peer connection, such as the torrent info hash of the handshake and if the peer is already connected to the client. If all tests are passed, the peer is added to the clients list of connected peers.

After this initial set of checks for the first round of peers for the tracker, the Optimistic Choke timer is set to execute 30 seconds later. Once 30 seconds pass, the timer run function checks all the unchoked peers and selects the one with the lowest sent bytes per second if client is not seeding, or lowest received bytes per second if client is seeding. The performance of each peer is measured within the peer thread with a timer that takes the amount of bytes sent and received over ever 2 second interval and does a weighted average with the already established bytes-per-second amount. When the lowest-performing peer is selected, it is choked and replaced and a random choked peer is unchoked in its place to maximize throughput.

In our programs main event loop, MessageTasks (MessageTask.java) are popped off the tasks queue and the message id of the message determines what case is executed by the spawned worker thread to maintained concurrent execution. This repeats, handing all uploading and downloading protocol for all the connected peers, responding to each peers task with a specific case in the event loop, constructing a message (Message.java) and sending it on the peers respective thread. In the case of downloading, the piece object (Piece.java) acts as an abstract of the bytes received and are verified and saved to disk by the clients destfile object (Destfile.java)

Periodically an announcement will be sent to the tracker executed by the TrackerAnnounceTimerTask with amounts uploaded and downloaded in the current session. We get back a new list of peers that our client will attempt to integrate into the current list of peers.

The event loop is exited upon the user entering a quit command, which throws a quit MessageTask into the queue of tasks so that task yet to be processed can complete. After the event loop, a graceful shutdown occurs where all peer and worker threads are tied off, all sockets and connections closed, and the main client thread returns, ending the main client thread program. In case of random emergency shutdown, the shutdown hook is invoked and the graceful shutdown cleanUp method is called, ending the program as gently as possible.

Class descriptions

RUBTClient

Spawns the main client thread and houses the event loop that handles message task. RUBTClient's main method parses the torrentfile specified by the command line arguments. Once parsed, RUBTClient spawns the main client thread, which is charged with maintaining input listener, connection listener, message task, and peer threads. After a list of peers is retrieved from initial tracker announcement, the client thread starts the event loop that spawns off worker threads to handle message tasks from the queue. This runs until the event loop is exited by the user, where it closes all connections and threads, then returns on the client thread to end the program.

Message Task

A wrapper that combines a Peer object and the message byte array its associated with. Gets placed in the MessageTask queue for processed by the eventloop. RUBTClient event loop takes MessageTask out of the task queue to perform certain actions on the tasks' respective peers.

Tracker

The Tracker class manages the communication between the client and the tracker. The constructUrl() method takes in the required information from torrentinfo and makes the url that will serve as the get request for the tracker to return a list of peers. Tracker also manages the creation of the random alphanumeric peer_id for the client, as well as the event messages that are sent to the tracker when the client has begun, finished, and stopped the download process. Tracker is called by the main method in RUBTClient.java

Peer

The Peer class manages the communication between the client and the peer, including the socket connections and input/output streams. Peer act on their own separate threads and take in messages sent by the client and send them through the output stream to peer and reads in the data on the input stream. When peer gets a message back from a remote peer, it sends the message up to the client thru the task message queue to be appropriately handled, however if it is sent a bitfield, it sets that field to its own as well. Peer also contains timers that see if a message has not been sent or received after a set amount of time. If these timers run out, it either composes and sends a keep alive message, or closes down the connection. A performance time keeps track of the bytes-per-second received and sent for use in the Optimistic Unchoke protocol. Peer threads are spawned from RUBTClient main thread

Message

The message class handles the composition of the different messages that will be sent to the peer, stored as a byte array. This classes methods are called by RUBTClient.java to compose all messages that will be sent to a remote peer.

Piece

The piece class builds a piece object, containing the piece data, piece number, offset. A piece is created every time peer gets a successful piece response. Once a piece is made its hash is verified by DestFile, and is then added into our destination file to compose the final download.

The assemble() method allows chunks to be placed into the piece's preallocated data array at any time, until the piece is ready to be placed into the final file.

DestFile

This class manages a RandomAccessFile used to create the final output file as well as a torrentinfo object for reference. Various other objects include: an array indicating the status of each piece in the torrent, a bitfield, and the parent client.

Methods include: addPiece to write Piece data, generateBitField to make bitfields more accessible, and verify, which verifies a Piece's data hash against the hashes within the DestFile's torrent info. Other methods are included to manage the bitfield, derive which pieces should be fetched next, and manually create and modify bitfields for peers who only provide have messages.

Response

This class analyzes the tracker's response to the initial GET request and extracts the bencoded dictionary containing peers, intervals, etc. Functions to print the peer list in a readable form and to select peers with the 'RUBT' prefix and IP are included

ConnectionListener

Runs on its own thread spawned from the main client thread. Opens a socket and listens on it for peers trying to connect to the client. Spawns a peer thread to handle handshaking before its added to RUBTClient's list of connected peers

ShutdownHook

Handles the unexpected shutdown that may occur while client thread is running by calling the clients graceful shutdown method, which ties up all client's peer and worker threads and closes all open sockets and connections.

*event loop, message task, and listening stuff borrowed from Rob Moores skeleton code on bitbucket.