CS 352 - Summer 2013 - Bittorrent Project 2 - Write Up
==========

Authors:

Isaac Yochelson Robert Schomburg Fernando Geraci

Introduction:

The purpose of this document is to explain the general implementation flow of the program. All the implemetation details were documented in javadocs and are available under this folder as index.html.

This bittorrent client implementation heavily leverages the native strengths of java in the fields of portablility and multi-threading. The portability comes from Java's system of partial compilation to intermediary byte code, which can be contained in secure jar files for each package, and then interpreted by any JVM properly implementing version 7 of Java. The multi-threading is being used to provide scalability in our project.

Implementation:

The program was originally engineered the way it looks know, a multipackage, multithreaded OOP design.

When the RUBTClient's main method is called, it imediately creates instances of Bittorrent and UserInterface, each on independent threads. UserInterface serves as our text-based user interface class, filling the role of controller in our design principle, and operates as an infinite loop until disposed of. Bittorrent is the center of the model of our design principle. The Bittorrent stores some information about the state of the system, and collates the information which will eventually be formed into our final file.

As soon as it is created, the Bittorrent will instantiate a Server. The server sits on a port that we reported to the tracker and awaits incoming connections. In the even of an incoming connection, the server instantiates a Peer to handle that connection and that peer is the placed on its own independent thread of execution. The peer object will act as a server to a particular peer on a specific TCP connection, handling incoming file requests. The peer will, during its creation, instantiate a peer listener on another independent thread of execution, and that peer listener will handle all incoming data on its connection, including handshaking and file transfers.

The bittorrent instantiates a TrackerRefresher, which sends updates to the the tracker at requested intervals, and feeds updated peer lists back to the bittorrent. So this strategy utilizes 4 base threads of execution, with an additional two for each peer we are in contact with. These threads are protected from one another by synchronized blocks on the various shared memory resources they make use of whenever they are reading to or writing from a shared resource that they did not allocate within the scope of the current method call stack.

In general, this software system is highly event driven. It implements a MVC design paradigm, having a Utils package for public static utility methods, Model package for backbone implementations, a View package, and an Exceptions package for custom built exceptions. The model consists of object representations of peers, as well as necessary state and file information we maintain in the classes of

the bt.Model package. The view consists of the text we output through the system standard output channel. The controller is a combination of command line

arguments and text based user input.

Classes: base: RUBTClient is the constructor in our system. Its function is to create the model and view, then instantiate a UserInterface to provide for user control through text input.

bt.Model: BitTorrent is the top level of our model. This class is instantiated to house all state information related to a particular .torrent file. This class has its own thread, where it determines what messages to send to the peers in the network. The BitTorrent handles communications with the tracker and saves the file to disk.

Peer is the next level of our model. This class is instantiated once for each peer that we are in contact with for each instance of BitTorrent. Peer handles serving requested file pieces to the peer it represents. It also handles handshaking and bitfields. It contains the methods for receiving incoming messages, but those are called by the PeerListener. Each Peer object runs on its own thread of execution.

PeerListener is part of the model, and every instance of PeerListener is linked to an object of type Peer, which is its parent. Peer Listener receives messages from a particular peer relating to a particular .torrent file. A PeerListener object operates on its own thread of execution.

Request is a simple class which is used in a manner similar to a struct in C++. A request is created as an immutable data container related to a single request message.

WeightedRequest is a subclass of Request which adds an extra field to store an integer weight value, and a native compare to function. This class is used so that we can give priority in our selection of requests to make in our downloading algorithm.

Server is a server to handle incoming requests for communication from peers. Once contacted, Server instantiates a Peer object for that peer to handle communications. This design is, unfortunately extremely vulnerable to DoS attacks, but that is not a major concern at the moment.

TrackerRefresher maintains a timer, and at a requested interval updates the tracker with our status, and retrieves an updated copy of the peer list.

bt.Exception BencodingException is an exception thrown by methods in Bencoder2.

DuplicatePeerException is thrown if a there is an attempt to create a peer representing an already known peer. This is possible if a peer we have already contacted attempts to initiate contact with us prior to receiving our connection request.

NotifyPromptException is used to interrupt the blocking read calls we use for user input with text output for our view.

UnknownBittorrentException is thrown if there is a call for an instance of the bittorrent class before any bittorrent objects have been properly instantiated.

bt.Utils Bencoder2 and TorrentInfo were provided for our use in this project. They are used for decoding binary bencoded messages during our interaction with the tracker.

CommandParser is part of the controller in our system. It parses text input to determine the actions the user has requested.

Utilities is a class which is never instantiated. It contains public static methods for use by other classes in the system.

bt.View

UserInterface is a text input parser, and is the controller in our system.

Project 2 additions:

A full graphic interface is now wrapping the main Bittorrent.java. This shell will interact with the different threads via direct calls to the classes.

The Download Algorithm utilized will queue pieces based on their rarity (weighted pieces as we call them) for then populating a common pending request queue which will be assign to the connected Peers throughout the algorithm's execution. Each Peer can have up to three pending requests to be fulfilled at any given time, and these will be updated in accordance to their arrival and verification.

A separate class on a separate thread called PeersSpooler, will be responsible for measuring download rates every 30 seconds for then ranking the peers as per download rate to decide which peers are going to be choked and unchoked accordingly. There is a limit set by the client which will be critical in the number of peers choked and unchoked at any given time. On the same token, upload choking will work in the same fashion, all uploading peers will be judge by their upload rate and choked accordingly.

Execution Notes:

1. The program is a GUI based interface which runs from RUBTClient.java