Introduction:

This document is to be used to give developers (and TA!) a guide to the layout of this Bit Torrent Client. Each section will provide details about implementation and use of each class. This document shall go in ground up order, meaning it shall start with primitives.

Primitives:

Piece → this primitive manages a particular torrent piece and allows for completion of the piece thtrough the addBlock() method. You can get the next missing block by calling getNextBlock().

BitMap → this primitive manages the organization and mapping of completed pieces. Its also may be responsible for ordering piece rarity among clients.

DownloadFile → This primitive manages the output file of the torrent. It keeps its self interms of pieces so that it may make its self useful to other classes.

Request → This primitive represents a request for block [index,begin,length]. Immutable.

MagnetLink → it was GOING TO BE :-(a class that would fetch the meta data from peers in the DHT.

Utilities:

Utilities provide helping functions to other classes.

Bencoding → Simple class that can be parsed bencoded byte array. It can also convert its self to a byte array.

ByteUtils → Simple class that provides static functions for reading and writing data from/to byte and char arrays.

TorrentParser → Contains a static function parseTorrentFile() which takes a file path. It produces the base class of the torrent. It utilizes the bencoding class.

Torrent:

These provide key files used to maintain and complete torrent downloading

PieceManager → This class provides an easy to use interface to store completed pieces. It's also responsible for maintaining file and piece interaction.
Note: that because all torrent data cant fit on ram, this class using a caching system of pieces

Torrent → This class provides the torrents state [peers,downloaded,uploaded, files, etc].

Networking:

MessageParser → while it's mostly utility, its the talking piece for a connection. It parses torrent messages from an input stream. It also provides the function of writing proper torrent messages into an output stream.

ManagedConnection → this is the connection to our peers. It keeps track of requests (from peer and from client), it also keeps track of peer bitMap using the BitMap class. It also keeps information like bytes downloaded and bytes uploaded. It also keeps track of maintenance bytes (bytes used for protocol control).

Tracker → This is a generic class that encapsulates the essence that trackers find peers, and peers are added to a torrent.

DHT-Tracker → this is an implementation of what i'll call a deferred DHT tracker. It wont actually hold a peer list. It merely searches the network for peers have have the required hash_info. It's much like a person stumbling around asking anyone if they've seen some one named bob. It does seem to always yield peers for popular torrents.

HTTP-Tracker → this class allows getting of peer list form a HTTP tracker.

UDP-Tracker → this class allows getting of peer list form UDP tracker

Logic:

The idea behind these classes were to make it easy to switch out different kind of peering logic so that it would be easy to interchange different kinds of logic. This is after all the interesting part of making a bittorrent client. However we ended up only really have one basic type of logic. What I was trying to build toward was being able to switch of different kinds of peering logic on the fly (without re-complication)

BasicPeerLogic:

Follows these rules:

- if someone asks for something that we can give, give immediately.
- Download rarest pieces first.
- If an outstanding request hasn't been downloaded within 10 seconds it gets canceled and the max number of outstanding blocks for the connection is reduced by a factor of 1/2.
- The max number of outstanding blocks is based on history of the connection and either grown or shrunken by the following:

if(# of received > 3* max outstanding && # of currently outstanding < .5){
    grow max outstanding by 1.5 times

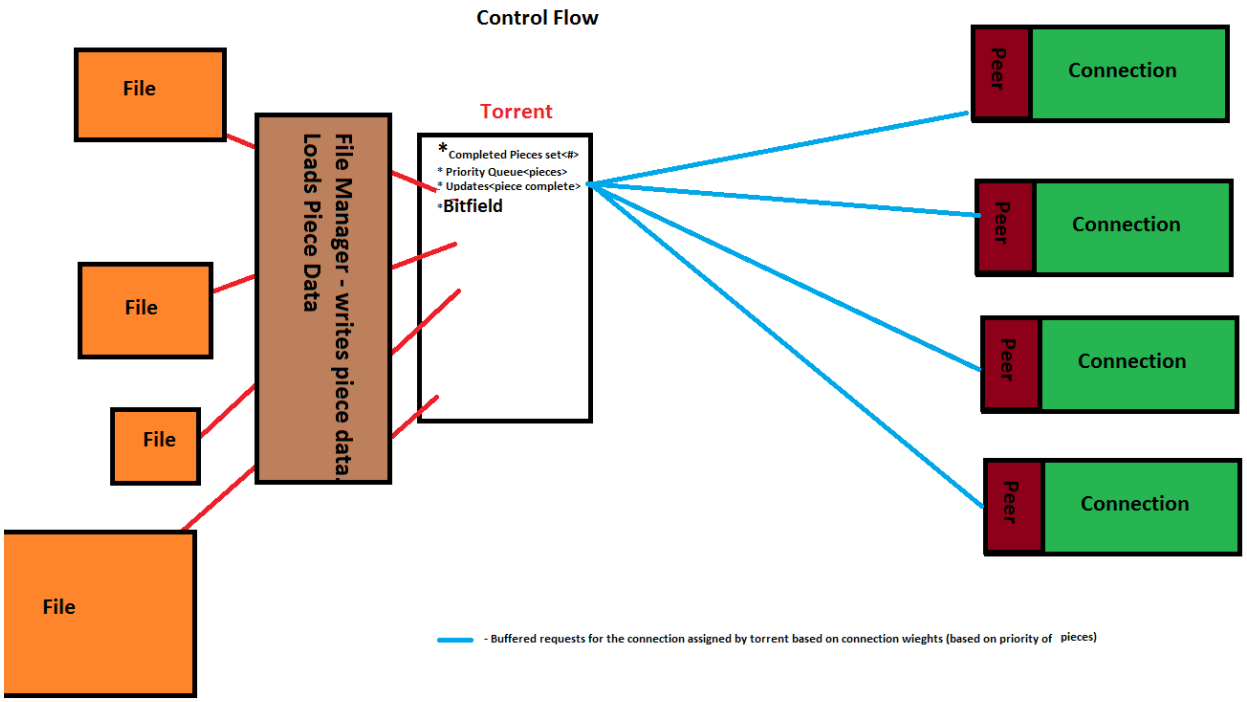
I felt this was a good scheme because it was a bit like tcp and its window growth. It allows for to utilize more bandwith if it can. Perhaps a more aggressive backoff should have been inorder :-/

I've noticed that the connection slows drastically after a few minutes of downloading. And that after restarting the client it speeds up again.

Disseminator → class that helps queue pieces across multiple connections.

- Keeps a list of whats actively being worked on.

Here is rough image of piece relation between connections and files:



Here is what our DHT tracker sorts through: