File Base:

torrent is a list of files.
Each file keeps starting and ending index of byte.
Client goes and retrieves files doing the follorwing:

- 1. Only load requested files
- 2. Load by rarity and give higher weights to files with preference

Client view:

- I only see torrent in terms of a big map of pieces.
- I only want certain area's of this map
- I prefer area's of this map that are most rare.
- I work in operation of **parts of pieces**

Algorithm Overview:

[Assume you have a list of connections that know their peer states]
Step 1 Torrent loops over client maps to build a priority list (aggrate map)
Step 2:
Queuing:
connections will pull from the priority last based on a weighting of both their peer's state and what is known about the connection [no connection is working on same piece at same time].
Notes-> wiki says to randomize the order of the top few rarest pieces.

Step 3:
Connections will be updated. Some data will be grabbed from connections and will be pumped upstream, some data will come down stream and be updated.

This should be enough to get us started on at least a some what working client. Other implementation will follow.

TODO: what if requests come for things on disk?

-cache? Well all I know is we cant just block till read.

Implementation Notes:
Chocked → no requests
not interested → no
Client weights are based on down and up rates.
Preference is given to clients that have the highest up/down ratio with a slightly stronger weighting to those clients that have given us more up.

Connection Algorithm:

OUT-BOUND LOGIC:

- If the Download/upload > x then client shall remain choked.
- We shall always remain interested
- If something in queue and we havent yet sent request for it, send it now.
- If something in request queue pass it out as long as down/up is above certain x
-

DHT -Tracker:
We associate with clients through ping. Until we get ping response we dont know their ID's thus they are pretty much useless. Queries are sent no more then once per 3 seconds for unresolved clients (unknown ID) and no more then once per 2 minutes for resolved clients. Max Drops for a client is 8. (16 minutes to become disconnected from known ID).

Rules of Speaking:

Dont talk to any known ID more then once per minute.
If in to out ratio is greater then 50 for any connection ban list:
ratio = in/out
Ban by IP. Ok for now we just drop node.
Outstanding request stay in queue for no longer then 30 seconds

Searching Algorithm (searching for a place to insert yourself):

```
if(have no peers){
    PING ROOT
}

if(!found matching ID || Matching List){
    get closest node;
    ask node (assuming you dont break max speaking rules)
}
```

Matching List → node gives us list of peers. This is where we shall announce our selves as well
Of course there could be no peers on the network. So we keep a counter of how deep we are
When this counter reaches 20 we give up and just announce and wait. After a certain amount of time (5 minutes) if we still haven't found peers we start back at root and search down. We always continue searching DHT for more peers. We keep a list of no more then 40 nearest nodes.
The DHT tracker does work no more then 1 times a second (if we've never found a peer). And no more then .05 times a second once we've found a peer.

Extension Protocols:
"metadata extension"

The biggest problem so far with this extension is that I've needed to support the idea of opening a blind connection. Meaning that other then the info hash we know nothing about the torrent. To build moduler I created a metaconnection class that basically handles the initialize and extension based messages. Other then this it pretty much does nothing, to support the concept of turning a metaconnection into a managed connection i've cached all incoming packets from the client. This way a managed connection can in theory be created out of a meta connection once basic torrent info is found (total bytes, piece length)