

Initialization (cont'd)

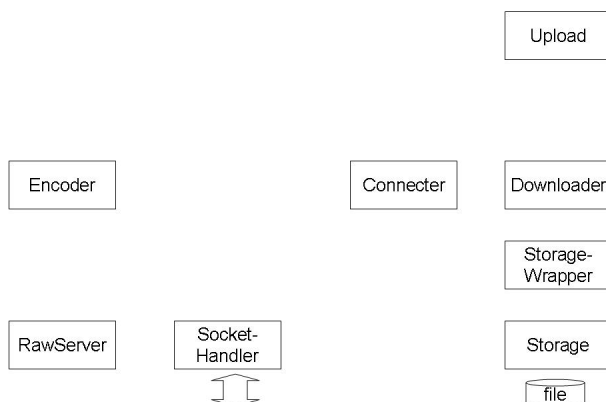


Figure 1:

Document: **I-Share Design Note**
Subject: **How BitTornado Works**
Author: **Arno Bakker**
Date: **September 16, 2005**
Status: *DRAFT*

1 Introduction

This document gives a partial description of the structure and workings of the BitTornado BitTorrent client (www.bittornado.com). In particular, it describes the steps taken when downloading a file, and gives a brief description of what each Python file does. The reader is assumed to be familiar with the BitTorrent protocol specification (www.bittorrent.com/protocol.html).

2 A Download

All modules and classes mentioned are defined in the BitTornado directory, or its BT1 subdirectory.

1. The user calls e.g. `btdownloadheadless.py` in the root directory to start the download.
2. The run method reads the config file `$HOME/.BitTornado/config.gui.ini` and parses command-line parameters.
3. Creates a 20-byte BT peer ID by calling `__init__.py/createPeerID` for this download session.
4. Creates a `RawServer.py/RawServer` object which creates a `SocketHandler.py/SocketHandler` object and schedules a task that checks for timeouts on the `SocketHandler`'s connections. A task is a method that the `RawServer` will execute once at a given time. Most tasks reschedule themselves, making themselves periodic. See Figure 1.

5. Attempts talk to any firewall/NAT box via Universal Plug 'n Play (UPnP) to see if there is one.
6. Calls RawServer to find and bind to a TCP listen port and to open this port on the firewall via UPnP.
7. Reads the torrent from disk if already present or downloads it via HTTP, turning it into a metainfo Python dictionary, as described in the BT protocol spec.
8. Creates a download_bt1.py/BTDownload1 object.
9. The constructor of the BTDownload1 object creates a BT1/PiecePicker.py/PiecePicker object. This latter object controls which piece to download next. It also creates a BT1/Choker.py/Choker object that will execute the optimistic unchoking policy. To this extent it schedules a task with the RawServer.
10. The client then calls BTDownload1.initFiles().
11. initFiles creates a BT1/Storage.py/Storage object, which maps all bytes in the files in a torrent into a single linear address space. All indices used in the BitTorrent protocol refer to this address space. So a piece of data identified by an index is mapped to a particular offset in a particular file by this object. Furthermore, it creates an empty file for each file in the torrent, and handles file locking.
12. initFiles creates a BT1/StorageWrapper.py/StorageWrapper object. Its main task is to execute the disk-allocation policy. Multiple policies are supported. E.g. "sparse" uses sparse files (i.e., the pieces are written on their correct place in a file, but only the actual data written is allocated on the file system). E.g. "pre-allocate" fills any gaps in the file with zeros. The default policy is "normal" which writes data consecutively into the files and moves the data in the right location in the background. In addition, the StorageWrapper schedules a task with RawServer that checks the integrity of any data already on disk in case of a restart. It also schedules a task that automatically synchronizes the data to disk.
13. For a multi-file torrent, if enabled on the command-line, initFiles creates a BT1/FileSelector.py/FileSelector object that enforces user-specified download priorities on the files.
14. The integrity check scheduled by the StorageWrapper object is run.
15. The client calls BTDownload1.startEngine().
16. startEngine() initializes the PiecePicker with information about the pieces already on disk, if any. It creates CurrentRateMeasure.py/Measure objects for up- and downloads. It creates a RateLimiter.py/RateLimiter object. It creates a BT1/Downloader.py/Downloader object that keeps track of the download side of the BitTorrent download (e.g. which pieces have been received, which pieces are available from peers, etc.). It creates a similar object for the upload side: BT1/Uploader.py/Upload.
It creates a BT1/Connector.py/Connector object. It creates a BT1/Encrypter.py/Encoder object which schedules a task with RawServer to send keep alives on all connections. (In the BT protocol, messages of length zero are keep alives). Finally, it may create a BT1/HTTPDownloader.py/HTTPDownloader object which is used by the 'httpseeds' extension to BitTornado that allows seeding of files directly from Web sites.

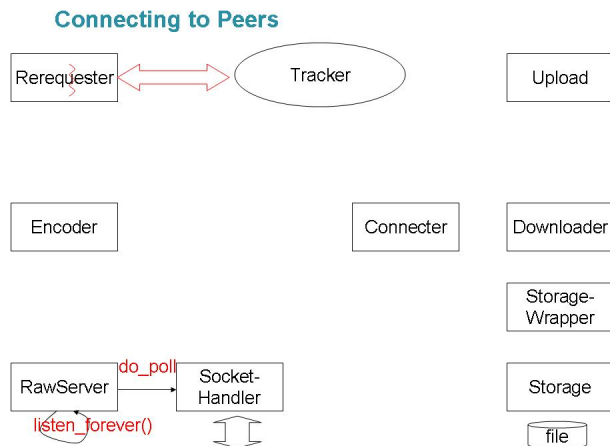


Figure 2:

17. The client creates a BT1/Rerequester.py/Rerequester object which is the tracker client and which uses a separate thread. See Figure 2.
18. When the Rerequester has obtained some peer addresses from the tracker it calls Encoder.start_connections(). See Figure 3.
19. The Encoder calls Rawserver.start_connection() and creates a BT1/Encrypter.py/Connection object for the created connection.
20. The Rawserver calls SocketHandler.start_connection().
21. The SocketHandler creates a Python socket and connects to the peer. It then registers the socket with a poll object from Python's select module, and creates a SingleSocket object for the connection.
22. The client finally calls RawServer.listen_forever(), the object's mainloop.
23. The mainloop calls SocketHandler.do_poll() which calls Python select's poll()
24. When data comes in on a connection, the SocketHandler.handle_events() method is called. This method reads the data from the socket and reports it to the Encrypter.Connection for the connection via the data_came_in() method. See Figure 4.
25. The Encrypter.Connection calls the next.func function pointer to handle the data. In the handshake phase of a BT connection this will call methods in Encrypter.Connection to handle the handshake. If the handshake is successful, the object calls the Connector's connection_made() method. All subsequently received messages are delivered to the Connector via got_message()
26. The Connector connection_made() method creates its own BT1/Connector.py/Connection object which will handle all non-handshake BT messages. It also registers the connection with the

Connecting to Peers (cont'd)

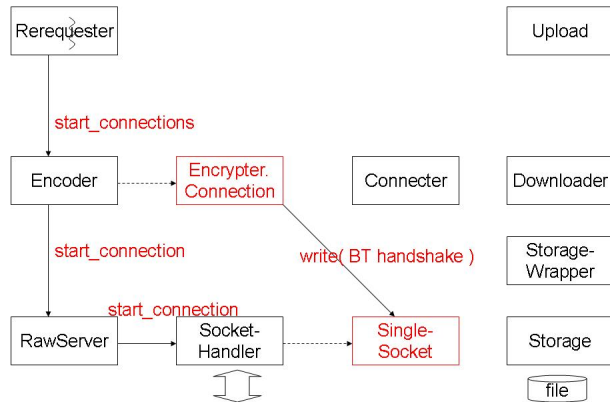


Figure 3:

BT Handshake

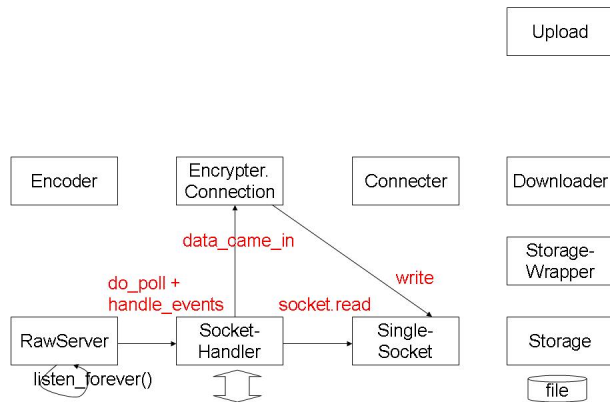


Figure 4:

BT Handshake Finished

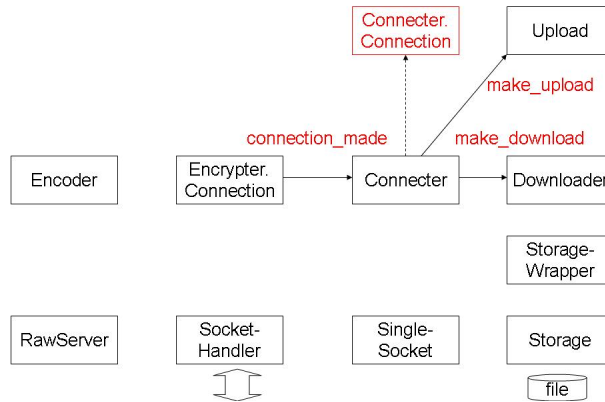


Figure 5:

Downloader, the Upload and the Choker. (Note that there are 2 classes called Connection, in the BT1/Encrypter.py module and in the BT1/Connector.py module.). See Figure 5.

27. When messages come in, the Connector will call the appropriate message handlers in these latter three objects and the Connector.Connection. For example, when a PIECE message comes in, it calls the Downloader, which, in turn, calls the StorageWrapper. See Figure 6.
28. To send a message, Connector.Connection calls the Encrypter.Connection, which, in turn, calls the SingleSocket object, which, in turn, writes the message to the Python socket.
29. When a new connection arrives on a listening socket, the SocketHandler calls the Encoder which creates a BT1/Encrypter.py/Connection object for it, as with outgoing connections.

3 File Descriptions

3.1 root directory

Contains the main Python scripts for

- Creating a torrent, via a terminal: btmakemetofile.py or GUI: btmaketorrentgui.py
- Starting the tracker: btrack.py
- Seeding and downloading a file, via a terminal: btdownloadheadless.py or a GUI: btdownload-gui.py

3.2 BitTornado directory

ConfigDir.py Creates directories for BitTornado caches. Defines functions for reading/ writing config files, BitTornado's state and current torrents.

'PIECE' Message

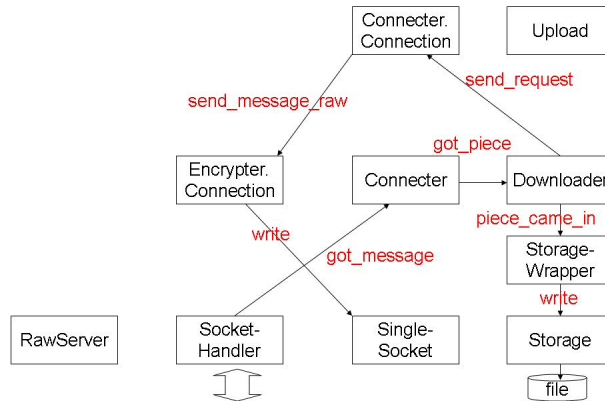


Figure 6:

ConfigReader.py Handles configuration via the GUI.

ConnChoice.py Input to a dropdown box giving a choice of network connections (DSL 512 kb, 1024 kb, dial-up, etc.) for configuration via the GUI.

CreateIcons.py Some icons in source-code encoding.

CurrentRateMeasure.py Defines class for measuring data rates, used in `btdownload.bt1.py` as argument to `BT1.Connector` and `BT1.Upload`, and by `BT1.Rerequester` for progress reports to the tracker.

HTTPHandler.py A HTTP server skeleton, used by the tracker `track.py`.

Psyco.py Enabled/disable Psyco, a JIT compiler for Python.

RateLimiter Defines class offering some rate limit calculations, used in various classes, see `download.bt1.py`.

RateMeasure.py Defines another rate-measure class?, used by `BT1.DownloaderFeedback`.

RawServer.py This module controls the main network I/O handler, `SocketHandler`, and executes tasks (i.e., a method executed at a particular point in time) on behalf of the BT client.

ServerPortHandler.py Variant of `RawServer.py` used by `launchmanycore.py` which downloads a bunch of torrents in parallel.

SocketHandler.py Main network I/O handler and dispatcher, handles the Python sockets.

__init__.py Initialization code for the client, contains code for creating a peer id.

bencode.py Encoding/decoding functions for BT's on-the-wire format, see BT protocol spec.

bitfield.py Defines BitField class used in BT protocol to report the available pieces to a peer at connection time.

clock.py Defines RelativeTime class (not used).

download_bt1.py Main module for downloads via BitTorrent, defines BT1Download class.

inifile.py Writes a Windows style INI file, used by ConfigDir.py.

iprangeparse.py Used by the tracker track.py for banned IP addresses.

launchmanycore.py Main module of parallel downloader, used by btlaunchmany.py.

natpunch.py Module to open ports on NAT box/firewall via UPnP.

parseargs.py Parser for command-line arguments.

parsedir.py Finds unique torrents in a directory tree, used by parallel downloader launchmanycore.py and the tracker track.py (to see for which torrents the tracker is allowed to track peers).

piecebuffer.py Defines PieceBuffer class used by BT1.Storage for reading from files.

selectpoll.py Defines class poll which keeps track of registered pollers (normal or listen sockets) used by SocketHandler if no polling is available from Python.

subnetparse.py Used by tracker track.py for access control.

torrentlistparse.py Used by tracker track.py

zurllib.py HTTP client used for communication with a tracker, e.g., by Rerequester class.

3.3 BitTornado/BT1 directory

Choker.py Defines Choker class.

Connector.py Handles incoming non-handshake messages via Connector and Connection classes.

Downloader.py Keeps track of download-side of a BT download.

DownloaderFeedback.py Keeps track of statistics and peer info (i.e., the so-called “spew” data) used by higher layers to display information about current peers.

Encrypter.py Defines the other Connection class that handles BT handshake messages. Defines the Encoder class that does connection setup of outgoing and handling of incoming connections and sends BT keepalives on existing connections. See Connector.py.

FileSelector.py Allows for assigning priorities to specific files in a multi-file torrent download.

Filter.py Used by the tracker track.py

HTTPDownloader.py Defines the HTTPDownloader class that supports HTTP downloads as part of the ‘httpseeds’ BitTorrent extension.

NatCheck.py Used by tracker track.py to check if it can connect to itself.

PiecePicker.py Defines PiecePicker class that selects the next piece to download.

Rerequester.py Defines Rerequester class, the tracker client.

Statistics.py Defines Statistics class that allows all statistics kept in the BT client to be accessed in one place (not used).

Storage.py Defines Storage class.

StorageWrapper.py Defines StorageWrapper class.

StreamCheck.py Some network test, commented out of SocketHandler.py

T2T.py Multitracker protocol module.

Uploader.py Defines Upload class, cf. Downloader.py.

__init__.py Practically empty.

btformats.py Functions for checking some BT data structures.

fakeopen.py Some file I/O stuff (not used).

makemetafile.py Functions to build a .torrent from a (set of) file(s).

track.py Main tracker module.