Lehrstuhl für IT-Sicherheitsinfrastrukturen, Informatik 1
Friedrich-Alexander-Universität Erlangen-Nürnberg

Bachelor Thesis

Analysis of BitTorrent Trackers and Peers

Counting Confirmed Downloads in BitTorrent

Stefan Schindler[1]

Erlangen, June 30, 2015

Examiner: Prof. Dr.-Ing. Felix Freiling
Advisor: Philipp Klein, M. Sc.

[1]Email: stefan@kaloix.de, Student number: 21676746

Todo

- Describe code functionality or exact implementation or both?
- Program tool
- Program evaluation and diagram output
- Perform main analysis
- Write thesis
- Grafik, Listings, Tabellen-Verzeichnis
- pymdht thesis; http://people.kth.se/ rauljc/p2p11/

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Task . . . 3
1.3 Related Work . . . 3
1.4 Results . . . 3
1.5 Outline . . . 3
1.6 Acknowledgments . . . 4

2 Background **4**
2.1 BitTorrent Protocol . . . 4
2.1.1 Bencoding . . . 4
2.1.2 Metainfo File . . . 4
2.1.3 Tracker Server . . . 5
2.1.4 Peer Wire Protocol . . . 6
2.2 DHT Protocol . . . 6
2.3 BitTorrent and German Law . . . 6

3 Implementation **7**
3.1 Dependencies . . . 7
3.2 Functionality . . . 7
3.3 Architecture . . . 9
3.4 Justification of Configuration Values . . . 9
3.5 Restrictions . . . 10
3.6 Usage . . . 10

4 Evaluation **10**
4.1 Choosing Torrents . . . 10
4.2 Confirmed Downloads . . . 10
4.3 Further Results . . . 10

5 Conclusion and Future Work **11**

References **12**

1 Introduction

[Some general information on the context and setting]

- BitTorrent by Bram Cohen 2008, a decentral network
- BitTorrent traffic statistics
- Other file sharing technologies

1.1 Motivation

[Specific motivation for the problem at hand]

- General statistics about illegal usage of BitTorrent
- Unique peers vs observed downloads as lower bound
- Gather statistics without up- or downloading content

1.2 Task

[Concrete task to be solved]

- Tool: Evaluate one or multiple given torrents
- Tool: Count observed downloads per hour
- Tool: Determine download speeds of peers
- Tool: Group them after geographical territories using IP address geolocation
- Drawback: One can only derive lower bounds from observing peers using the standard BitTorrent protocol
- Drawback: There is no complete overview about all running torrents, we need to choose a subset
- Evaluation: Choose interesting torrents for analysis from different content categories
- Evaluation: Run analysis tool for several days or weeks

1.3 Related Work

[Other relevant academic work and how it differs from this work]

- [16]
- [4]
- ...

1.4 Results

[What has been achieved in this work?]
...

1.5 Outline

[How is the thesis structured and why?]
...

1.6 Acknowledgments

[A big thank you for the support to ...]

. . .

2 Background

This chapter explaines technologies and specifications utilized during this research project.

2.1 BitTorrent Protocol

Besides several extensions, BitTorrent still works as designed by Bram Cohen in January 10, 2008, in the *BitTorrent Enhancement Proposal* number 3, shortened BEP 3 [3]. It establishes a method to distribute a predefined set of files amoung an arbitrary number of recipients without overwhelming load on a central entity. This is achived by splitting the file set in pieces and let peers send them to each other. Three main parts are defined to enable the process: The BitTorrent file format containing identifying metadata about the file set, the communication procedure with a tracker server where peers can learn internet protocol addresses of other peers, and the peer wire protocol spoken between peers.

2.1.1 Bencoding

In order to store common data structures as well as transmit them over TCP, encoding is required to preserve the data's type and semantic. To realize BitTorrent, Cohen came up with *bencoding* to annotate data appropriately. Any integers and length information is encoded in base 10 ASCII format. All types but strings have specific beginning and ending delimiter characters. Basic supported datatypes are byte strings and integers, saved as `<length>:<string>` and `i<integer>e` respectively. Composite types include lists stored as `l<value1><value2>e` and dictionaries alike `d<key1><value1><key2><value2>e`. Note that only strings can be used as dictionary keys.

2.1.2 Metainfo File

Metadata about a downloadable file set is stored bencoded in a metainfo file, which uses the `.torrent` file name extension. The purposis is to locate the tracker server and describe all content pieces with cryptographic hashes. Data is arranged in dictionaries of the following structure:

announce Uniform resource locator of the tracker server, usually of the form `http://<host>:<port>/announce`.

info The info dictionary describing the torrent's pieces

- **name** Optional file or directory name
- **piece length** Number of bytes per piece
- **pieces** Concatenation of raw SHA-1 hash values for every individual piece
- **length** Only present if the torrent is a single file; file size in bytes; not used in this project at all
- **files** Only present if the torrent containes multiple files; list of dictionaries, one per file; not used in this project at all
  - **length** File size in bytes
  - **path** List of subdirectory names and filename representing a path

The number of pieces can be derived from the `pieces` key of the `info` dictionary by dividing it's length by 20, since a SHA-1 hash is 20 bytes. The torrent identifying *info hash* is calculated as the SHA-1 hash of the bencoded info dictionary.

2.1.3 Tracker Server

Traditional communication with the tracker server is done via the GET request method of the Hypertext Transfer Protocol. The request is sent with the following parameters, whereby keys and values must be quoted using percent-encoding [2, § 2.1]. TODO announce

info_hash SHA-1 hash of the bencoded info dictionary from the metainfo file

peer_id String of 20 bytes self choosen by each peer; contains client software information by convention

ip Optional parameter with the peer's own IP address

port Port number this peer is listening on for connections from other peers; recommended ports are 6881 to 6889

uploaded Amount of uploaded pieces so far

downloaded Amount of downloaded pieces so far

left Amount of pices left to download

event Optional key about the circumstances of this request; is `started`, `completed` or `stopped`

compact To save bandwidth, a compact list of peers can be requested according to BEP 23 [5]; is `0` or `1`

The tracker's response message should contain a bencoded dictionary in the message body. Following keys are defined:

failure reason In case of failure, human-readable error message explaing why the request could not be fulfilled

interval Suggested interval in seconds the client should wait between tracker requests

peers List of dictionaries, one per peer

> **peer id** Self-selected ID
>
> **ip** IP address
>
> **port** Port number
>
> In case of a compact peer list, this is a single byte string instead of a list of dictionaries. 6 bytes per peer are used containing 4 bytes for the IPv4 address and 2 byte representing the port number.

Tracker servers are the only centralized infrastructure required by traditional BitTorrent. Hence it is advisable to reduce bandwith during tracker requests as much as possible. As BEP 23 demonstrates, using the *UDP tracker protocol* instead of HTTP over TCP can reduce traffic by 50 % [14]. Since UDP datagrams may arrive out of order, the client sends a randomly choosen transaction ID with every request to identify the matching server response afterwards.

connect The first step in the UDP tracker protocol is a connect request. A connection ID is sent in return by the server, which must be included in following requests. As it is possible to spoof an UDP packet's scource IP address, the server could be abused for a denial-of-service amplification attack against a third party. The need for a connection ID on other requests, which trigger larger responses, renderes this impossible. The connection ID is valid within the next minute.

announce This announce request includes the same parameters as the HTTP commnication described above. Additional parameters are an unused `key` value and the `num_want` value, allowing to specify the amount of returned peers. The announce response again includes a desired request interval as well as a number of active leechers and seeders. Peer's IPv4 addresses and port numbers are included using six bytes each.

Finally a scrape request is defined, giving clients access to the numbers of leechers, seeders and completed downloads as reported by the server. There is no guarantee of validity for these values, since they may be manipulated or choosen by the server freely. An error response package containing a human-readable message may be sent by the server at any time.

2.1.4 Peer Wire Protocol

The peer protocol is spoken between peers and allows bidirectional communication with predefined messages. At first an initial handshake is exchanged, containing a protocol string, eight bytes reserved for alternate protocol behaviour and extensions, as well as the torrent's info hash and the peer's ID. The connecting client sends it's handshke message first. All following messages begin with an overall lenght prefix, followed by a type identifyer and the payload, if appropriate.

The so called `bitfield` message may be sent immediately after the handshake to indicate which pieces were already downloaded and verified by the peer. With same intentions `have` messages are sent to all connected clients, if a peer has successfully downloaded a new piece. These are the only two message types relevant in the scope of this work; for completeness, the meanings of further message types are as follows: `choke` and `unchoke` express the willingness and possibility to fulfill requests for pieces. Similarly `interested` and `not interested` indicate whether a peer would start downloading if unchoked. A bulk of missing pieces can be requested using a `request` message with begin and end indices, a single piece can be requested using `piece`. When requests were sent to multiple clients to increase download speed, a `cancel` message is used to revoke a pieces request.

2.2 DHT Protocol

Despite the complete file payload being transmitted from client to client, it is not possible to run BitTorrent without a central server keeping track of all peers. However, the mandatory tracker server contradicts the concept of a decentralized file distribution network and, in addition, has to be maintained financially. The *DHT Protocol* [9] solves this problem, as stores peer contact information in a distributed hash table. Participating peers run a seperate DHT *node*, which listes on a UDP port.

Nodes First, a node calculates a random 20 byte identifier, called node ID. The distance between two node IDs is defined as the bitwise exclusive disjunction interpreted as an unsigned integer. Each node maintaines a routing table, which maps node IDs to their corresponding node's IP address and UDP port number. The closer node IDs are to the node's own ID, the more nodes are stored in the routing table. Therefore, nodes only know about a limited number of other nodes, whith greater detail close to thier own identifier.

Lookup The process of extracting peers for a given info hash proceedes iteratively. The same distance metric as used between node IDs is used to identify nodes in the routing table with IDs close to the info hash in question. Due to the routing table's structure, contacted peers can return IDs and addresses of even closer nodes. Eventually, nodes will be able to return actual peer contact information, since peers known to download a torrent are stored in a second table, the hash table. Following the same scheme used on the routing table, peers with info hashes close to the own node ID are stored preferably.

Announcing When a peer downloads a torrent, it should announce the info hash and BitTorrent port to multiple other nodes in order to be included in the distributed hash table. Again, the problem of IP address spoofing exists, allowing malicious hosts to register third parties for a torrent. This is why a token system is used. On every successful request for peers, the response includes the SHA-1 hash of both the source IP address and a secret value. The recipient's IP address and the IP address included in the hash are now guaranteed to be equal. When announcing download participation, a node must include this token, allowing the contacted to verify the announce request's source IP address and updating it's hash table.

2.3 BitTorrent and German Law

...

Basic legal details of uploading and downloading via BitTorrent
Basic legal classification about the analysis tool of this thesis

3 Implementation

The software tool written for this thesis was developed using the version control system Git [15] in conjunction with a private online repository provided by GitHub. The source code was published under the GNU General Public License version 3 [13].

3.1 Dependencies

There are a few external dependencies, which are all free and open-source software. The *BencodePy* project by Eric Weast [17] provides an encoder and decoder for bencoding messages and values. The *Object Relational Mapper* of *SQLAlchemy* [1] is used to store evaluation results in the *SQLite* database format [7]. The *GeoIP2 API* [10] is used to perform IP geolocation lookups in the *GeoLite2 City Database* [11]. This database is provided by MaxMind, Inc. under the *Creative Commons Attribution-ShareAlike 3.0 Unported License*. In order to run a dedicated DHT node, the tool *pymdht* by Raul Jimenez [8] is used.

3.2 Functionality

To count confirmed downloads by peers of one or multiple given torrents over a time period, the *BitTorrent Download Analyzer* was written in Python 3. Torrents and all configuration parameters have to be provided at start, as they cannot be changed later. The program stores results in a SQLite database and runs until manual termination. A configuration file with several variables named `config.py` is provided. For simplification these variables will be referred to with the prefix “`config.`” in the following, so `config.x` translates to variable `x` in the configuration file.

The main task is to count confirmed downloads by peers of a given torrent. A download is considered as confirmed, when a peer crosses a threshold of downloaded pieces as defined in `config.torrent_complete_threshold`. Thus there must be contact with a peer at least twice – with the amount of downloaded pieces once below and once equal or above the threshold – in order to be counted. To determine the download progress of as many peers as possible, two tasks have to be done: First, establish contact to peers from every possible source. Second, receive continuous and reliable information about the download progress of every peer.

Import Torrents Beforehand, the torrents to be analyzed must be imported. Since both magnet links and torrent files are supported, there are two major ways to do this: All torrent files must be placed in a common directory, specified by `config.input_path`. They are detected by their `.torrent` file name extension. Following the specification of BitTorrent files [3], the announce URL, infohash, pieces count and pieces size are extracted.

All magnet links [6] to be considered must be placed in a file defined by `config.magnet_file`, one per line. Since magnet links do not contain the amount and size of the torrent’s pieces, but only their infohash, this information must be retrieved from the swarm of other peers. A few peer adresses are gathered with a DHT lookup, then peers are contacted sequentially until the info dictionary could be received using the Extension Protocol [12] and the *ut_metadata* extension [6]. The metadata and source of each imported torrent is stored the database for later reference.

Contact Peers Sources for peer’s IP addresses and port numbers include announce requests to the tracker server as well as lookups in the BitTorrent DHT network. Both are done in dedicated threads and periodically as defined by `config.tracker_request_interval` and `config.dht_request_interval`. The received peer addresses are filtered for duplicates and placed in a common queue. Each request procedure is recorded in the database with the number of received peers, duplicate peers and duration for further analysis.

Every peer in the queue has a timestamp assigned and must not be contacted prior to this time. When placed in the queue first, the timestamp is set to the current time. Peers in the queue are visited in parrallel, whereby the number of threads can be set in `config.peer_evaluation_threads`. The queue is sorted ascending, so if timestaps lie in the past the peer which is due the longest time is choosen, if timestamps lie in the future the thread will wait until the attached time is reached. For threads to be able to react to new peers, waiting is capped to `config.evaluator_reaction`. When the limit is reached, the peer is put back in queue and another one will be choosen.

When a peer with permitted timestamp is picked, a TCP connection is established and the download progress evaluation initiated. Additionally, incoming connections from peers trying to download pieces are used to gather their download status. Therefore a TCP server is listening on the the port defined in `config.bittorrent_listen_port`. On successful download progress extraction results are written to the database and IP address port tuples are linked to their database ID in internal memory for later database updates. Failed contacts or peers with unknown infohashes are ignored.

Evaluate Download Progress Once a connection is established with a peer, it's download progress must be determined only using peer messages as defined by the BitTorrent Protocol [3]. Since there is no dedicated request command for the number of available pieces, we depend on peer messages sent voluntary by the remote peer. Fortunately it is common to advertise available pieces right after the BitTorrent Protocol handshake with `bitfield` and `have` messages. These are stored for every peer contact in a seperate queue and processed by another thread. Messages are received until a timeout defined by `config.network_timeout` hits, which is restarted after every message. Additionally there is a limit on the number of messages named `config.receive_message_max` to prevent infinite sessions.

Now the only possible approach to aquire the download progress is to compile a combined bitfield from these messages and count the present pieces. The number of total torrent pices from in the info dictionary helps validating the results.

Peer Database With reference to the peer's database identifier, the IP address and port number are saved in the internal memory in order to update an already stored peer entry in the database at later contact. Time and pieces count only of the first and the last peer contact are saved in the database, since this is enough to assess the transition of the confirmed downlaod's threshold. Additionally the download speed is calculated between each two consecutive contacts, whereby only an overall maximum is kept.

Other information about peers stored in the database include an anonymized IP address, the BitTorrent Protocol peer ID, top and second level domain of the hostname, IP geolocation with city, country and continent as determined by the *GeoIP2 API* [10], number of contacts and the original source of the peer's IP address. Afterwards the peer's queue timestamp is updated with the current time plus `config.peer_revisit_delay` and it is returned to the queue if `config.torrent_complete_threshold` is not reached.

Secundary Statistics In order to enable and proof validity of test results some statistics are logged at a certain interval defined in `config.statistic_interval`. These are the length of the peer queue, length of the queue of visited peers, average workload of peer contact threads, mean time for receiving all messages before timeout. Cumulative values are unique incoming peers, successful initiated contacts, failed initiated contacts at first try, failed initiated contactsa at later try and successful evaluated incoming peers.

...

Explain implementation of the features:

- Import torrents from `.torrent` files (BEP 3)
- Import torrents form magnet links by fetching metadata via the *ut_metadata* extension (BEP 9) using the Extension Protocol (BEP 10)
- Continuously get IPv4 peers from the tracker using HTTP (BEP 3) and UDP announce requests (BEP 15)
- Communicate with peers using a subset of the Peer Wire Protocol (BEP 3)
- Continuously get IPv4 peers by integrating a running DHT node (BEP 5) from the *pymdht* project using local telnet
- Actively contact collected peers and calculate minimum number of downloaded pieces by receiving all *have* and *bitfield* messages until a timeout

- Passively listen for incoming peer connections and calculate minimum number of downloaded pieces analog
- Save number of downloaded pieces from first and last visit and maximum download speed per peer in a SQLite database
- Save city, country and continent via IP address geolocation
- Save ISP by hostname and anonymized IP address
- Analyze multiple torrents at once
- Synchronized analysis shutdown process
- Produce extensive log output
- Save duplicate and timing statistics about peers received via DHT and tracker
- Save statistics about failed and succeeded peer connections
- Save workload statistics of active peer evaluation threads

3.3 Architecture

It is structured in the main script, an application module, five helper modules and an utility module, which will be described in detail.

They have the following roles:

`main.py` This is the main script to be invoked when performing the analysis.

`analyzer.py`

`torrent.py`

`tracker.py`

`dht.py`

`protocol.py`

`storage.py`

`util.py`

`config.py`

3.4 Justification of Configuration Values

`config.network_timeout` The timeout for network operations is five seconds. It is used when asking the BitTorrent tracker or DHT node for peers and when asking other peers for metadata. These cases are uncritical as a failure is visible in the log files and did not happen during this research. The important spot of application is during the peer evaluation process. While all messages are received from a peer, the timeout resets after every message. The message collection is considered complete after the timeout finished without receiving a message.

To assess a reasonable timeout, the maximum used time between messages was recorded for every peer session. An average of these values was calculated and stored every five minutes, according to `config.statistic_interval`. Only for this timeout estimation run `config.network_timeout` was set to 30 seconds to achive most unbiased results. As can be seen in [**timeout-test**], five seconds are _ times above the total average of _ seconds.

INSERT MAXIMUM MESSAGE TIMEOUT AVERAGES HERE

`config.torrent_complete_threshold` ...

`config.peer_evaluation_threads` ...

3.5 Restrictions

Restrictions and why this does not invalidate the results (hopefully, TBD):

- No support for IPv6 on HTTP, UDP or DHT requests
- No support for the Micro Transport Protocol (μTP)
- No support for Peer exchange (PeX)
- No support for the Tracker exchange extension (BEP 28)
- No support for the BitTorrent Local Tracker Discovery Protocol (BEP 22)
- No BitTrorrent Protocol support for getting download progress info

3.6 Usage

pymdht: It must me started seperately and is controlled automatically using a localhost Telnet connection. It could not be integrated directly, since it is written in Python version 2. The desired UDP node port and the Telnet control port must be given as arguments and should reflect the values written in the *BitTorrent Download Analyzer's* configuration file. The typical command used here is `run_pymdht_node.py --port=17000 --telnet-port=17001`. It is sensible to check whether *pymdht* has crashed before and after each analysis run to ensure complete results.

The main script, named `btda.py` can be controlled by using the following command line options.

`--active <threads>` Actively contact and evaluate peers using the specified number of threads.

`--passive` Listen on the port specified in the configuration file for incoming connections and evaluate these peers.

`--dht` Integrate and control an already running *pymdht* [8] DHT node using Telnet. The UDP port on which the node is running and the localhost Telnet port where *pymdht* can be controlled are given via `config.dht_node_port` and `config.dht_control_port` respectively.

`--debug` Write log messages to the console instead of a file and include debug messages.

`--help` Show a help message and exit.

4 Evaluation

4.1 Choosing Torrents

4.2 Confirmed Downloads

Diagram: Timeline of confirmed downloads in 1 hour steps

Map: Distribution of downloads (TODO)

Map: Distribution of download speeds (TODO)

4.3 Further Results

Diagram: Timeline of new vs. duplicate received peers in 1 hour steps (TODO)

Table: Examine hostnames by ISP or seedbox provicers (TODO)

5 Conclusion and Future Work

...

References

[1] Michael Bayer. *SQLAlchemy*. Version 1.0.5. 2006. URL: `http://www.sqlalchemy.org/`.

[2] Tim Berners-Lee, R Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic syntax*. RFC 3986. 2005. URL: `https://tools.ietf.org/html/rfc3986`.

[3] Bram Cohen. *The BitTorrent Protocol Specification*. BEP 3. 2008. URL: `http://www.bittorrent.org/beps/bep_0003.html`.

[4] Anders Drachen, Kevin Bauer, and Robert WD Veitch. "Distribution of digital games via BitTorrent". In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. ACM. 2011, pp. 233–240.

[5] David Harrison. *Tracker Returns Compact Peer Lists*. BEP 23. 2008. URL: `http://www.bittorrent.org/beps/bep_0023.html`.

[6] Greg Hazel and Arvid Norberg. *Extension for Peers to Send Metadata Files*. BEP 9. 2008. URL: `http://www.bittorrent.org/beps/bep_0009.html`.

[7] D. Richard Hipp. *SQLite 3*. 2000. URL: `https://www.sqlite.org/`.

[8] Raul Jimenez. *pymdht*. Version 12.11.1. 2009. URL: `https://github.com/rauljim/pymdht`.

[9] Andrew Loewenstern and Arvid Norberg. *DHT Protocol*. BEP 5. 2008. URL: `http://www.bittorrent.org/beps/bep_0005.html`.

[10] MaxMind. *GeoIP2 Precision Web Services. MaxMind APIs*. Version 2.1.0. 2014. URL: `http://dev.maxmind.com/geoip/geoip2/web-services/#MaxMind_APIs`.

[11] MaxMind. *GeoLite2 Free Downloadable Databases*. 2015. URL: `http://dev.maxmind.com/geoip/geoip2/geolite2/`.

[12] Arvid Norberg, Ludvig Strigeus, and Greg Hazel. *Extension Protocol*. BEP 10. 2008. URL: `http://www.bittorrent.org/beps/bep_0010.html`.

[13] Stefan Schindler. *BitTorrent Download Analyzer*. 2015. URL: `https://github.com/st3f4n/bittorrent-analyzer/tree/master/btda`.

[14] Olaf van der Spek. *UDP Tracker Protocol for BitTorrent*. BEP 15. 2008. URL: `http://www.bittorrent.org/beps/bep_0015.html`.

[15] Linus Torvalds. *Git*. Version 2. 2005. URL: `https://git-scm.com/`.

[16] Paul A Watters, Robert Layton, and Richard Dazeley. "How much material on BitTorrent is infringing content? A case study". In: *Information Security Technical Report* 16.2 (2011), pp. 79–87.

[17] Eric Weast. *BencodePy*. Version 0.9.4. 2014. URL: `https://github.com/eweast/BencodePy`.