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, August 19, 2015

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

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

Eidesstattliche Erklärung / Statutory Declaration

Hiermit versichere ich eidesstattlich, dass die vorliegende Arbeit von mir selbständig, ohne Hilfe Dritter und ausschließlich unter Verwendung der angegebenen Quellen angefertigt wurde. Alle Stellen, die wörtlich oder sinngemäß aus den Quellen entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

I hereby declare formally that I have developed and written the enclosed thesis entirely by myself and have not used sources or means without declaration in the text. Any thoughts or quotations which were inferred from the sources are marked as such. This thesis was not submitted in the same or a substantially similar version to any other authority to achieve an academic grading.

Der Friedrich-Alexander-Universität, vertreten durch den Lehrstuhl für Informatik 1, wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Arbeit einschließlich etwaiger Schutz- und Urheberrechte eingeräumt.

Erlangen, August 19, 2015

Stefan Schindler

Zusammenfassung

Zusammenfassung auf Deutsch

Abstract

Zusammenfassung auf Englisch

Contents

List of Figures

List of Tables

1. Introduction

In 2008, Bram Cohen published the specification for a decentralized file sharing protocol called *BitTorrent*. It soon became the most used file sharing technology, since it enables users to publish and distribute collections of large files easily. The main advantage is the peer-to-peer technology used for data transfer, eliminating the need for central file servers with heavy load or even costly content distribution networks. On top of that, the integrated file validation using the cryptographic hash function SHA-1 enable software clients to verify received data. This makes the protocol robust against transmission errors and malicious peers trying to manipulate the content. Finally, BitTorrent can operate over slow and unreliable connections exceptionally well, because the payload is split in small pieces of data which can be sent in arbitrary order and received from different participants.

To date BitTorrent still has a remarkable share in private Internet traffic. According to a study [15] by networking equipment company Sandvine Inc., BitTorrent has a downstream traffic share in fixed-line Internet accesses between 3 % in North America and 23 % in the Asia-Pacific region, with Europe at 10 %. In downloaded data per month, this translates to 1.4 GB, 7.2 GB and 2.4 GB, respectively. Even more bandwidth is used for upstream with values ranging from 24 % in Latin America to 56 % in the Asia-Pacific region. Detailed numbers are cited in table 1.1. Other file sharing technologies than BitTorrent are hardly used.

File sharing such as BitTorrent is reported by music as well as film industry to cause billions of losses: An industry friendly institute reported "3.7 billion USD Estimated Download Piracy Losses to U.S. Integrated Firms" [18, table 1] in 2006 regarding music sales only. However, the harm of illegal downloads is unclear [5]. Other studies suggest delayed digital releases promote piracy [3] or find no [14] or even positive [19] correlation between illegal downloads and legal sales. Undoubtedly the amount of copyright infringing content which is downloaded via BitTorrent is quite high. A case study [22] from the University of Ballarat, Australia, finds numbers between 90 % and 97 %.

1.1. Motivation

When assessing popularity or peer numbers, previous studies [4, 22, 24] often relied on numbers reported by tracker servers using so called *scrape requests*. These requests allow clients to ask for statistics about one, multiple or even all torrents a tracker is managing. When successful, servers answer with the number of current downloaders, current uploaders and finished downloads since the torrent was registered with the server. These values may be either to their best knowledge or flawed and misleading, it can even be manipulated by peers by sending false download confirmations to the tracker. There is no possibility for a client to verify these numbers.

This thesis will make an attempt to collect confirmed download numbers by contacting every peer of the BitTorrent swarm for a given set of torrents and learning his download progress form first-hand. The download progress is extracted using the standard BitTorrent protocol with various common extensions thereof. This is done repeatedly for all peers over a time period, while a confirmed download is recorded when the peer crosses a certain threshold. This method has flaws which will be discussed later*, but it gives a fix lower bound for download numbers.

To observe the law in every way, it was important to neither download nor upload any actual content. Luckily this is not necessary for this task, as it is practice for every peer to inform the opposing peer about its exact presence of downloaded pieces upon connection establishment. This behavior will be exploited by recording this progress in a database.

		Upstream		Downstream	
Region	Access Type	Share	Volume	Share	Volume
North America	fixed	25.49 %	2,167 MB	2.80 %	1,369 MB
	mobile	1.88 %	1 MB	n/a	n/a
Europe	fixed	36.56 %	1,865 MB	10.39 %	2,400 MB
	mobile	8.99 %	6 MB	n/a	11 MB
Latin America	fixed	23.87 %	454 MB	7.42 %	913 MB
	mobile	n/a	n/a	n/a	n/a
Asia-Pacific	fixed	55.91 %	7,492 MB	22.78 %	7,221 MB
	mobile	3.43 %	5 MB	n/a	n/a
Africa	fixed	28.21 %	n/a	13.29 %	n/a
	mobile	3.59 %	n/a	4.88 %	n/a

Table 1.1.: Share and volume per month of BitTorrent traffic per household, from SANDVINE study *Global Internet Phenomena Report 2H 2014* [15]. *Share* percentages were determined by SANDVINE and reportedly measured during "peak period traffic", see "Top 10 Peak Period Applications". *Volume* values given above are an estimation based on BitTorrent's share in peak traffic and SANDVINE's mean value for overall "Monthly Consumption" per household. *n/a* indicates BitTorrent is not among the top ten applications. No traffic volume is stated for Africa.

1.2. Task

Since there is no client or framework for peer communication on BitTorrent protocol level, and every other task is very specific to the requirements of this project, the whole code base used here was written from **scratch**. The need for communicating with peers without downloading any torrent payload disqualifies other related projects like *libtorrent*. Only exception is the mainline DHT node which is a foreign project.

The process of analyzing a torrent should be completely automated. As the most convenient method, **input** of torrents via `.torrent` files and magnet links is supported. They must be parsed beforehand and are stored in a SQL database for later reference. Metadata for magnet links is retrieved from other peers using the *Extension for Peers to Send Metadata Files*.

Secondly, addresses of **peers** participating in the relevant torrents are needed. They are collected by sending appropriate requests to the torrent's tracker servers using either the TCP or the UDP tracker protocol. Likewise requests for peers of the given torrent are issued on the DHT network. The collection of peers is performed continuously during the analysis to include newly participating clients. Additionally a TCP server is listening for incoming connections in order to include peers behind a NAT and hence are not reachable otherwise. Incoming peer will be evaluated equally, but only can be counted when connecting once with progress below and once with progress above the threshold. Statistics about duplicate versus unique received address-port tuples are recorded.

Reading a peer's download **progress** is the core part. After exchanging BitTorrent protocol handshakes, all further messages from the remote peer are received and recorded until no more message is received for certain period. These messages specify which pieces are available for download, and analog which pieces the peer has downloaded. There is research whether or not peers can gain an advantage for misreporting this data [8, 9], but until now it has not surfaced as a problem at large. Before closing the connection, a message announcing the port of my own DHT node is sent to the peer in order to popularize the node's routing table.

The download progress is stored in a **database**, together with a timestamp of the contact. When contacting a peer later another time, a decision can be made if the peer has crossed the threshold. This threshold is below 100 % to compensate for peers disconnecting immediately after they finished the download. For additional analysis, the peer's download speed is derivated. Also, a lookup in a IP geolocation database is performed, and the peer's location, host name and client program is recorded.

In the database there is only one record per peer, with two pieces values: one from the first contact with that peer and one from the latest contact. The number of confirmed downloads can now be obtained by filtering for peers with the first value below the threshold and the second one equal or above the threshold. Remaining rows can be aggregated by their latest timestamp to get download numbers per hour.

1.3. Related Work

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

- [22]
- [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 ...]
* Philipp Klein for discussing methods and implementation
* RRZE for providing the VM and probably receiving copyright warning letters
* Michael Gruhn for discussing ideas
* Felix Freiling for extending the deadline
...

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 [BEP 3]. It establishes a method to distribute a predefined set of files among an arbitrary number of recipients without overwhelming load on a central entity. This is achieved 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 data types 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

A torrent's payload may be either a single file or a directory with subdirectories and multiple files. The metadata of such a downloadable file set is stored in a bencoded file, called *metainfo file*, which is using the `.torrent` file name extension. For easy reference, values are grouped in dictionaries as listed in table 2.1. These files allow users to identify torrents, since they contain a human readable description as well as cryptographic hash values on the torrent's pieces. Additionally, the URL of a tracker server is stored, allowing BitTorrent clients to gain information about other peer's addresses and participate in the network. The *info hash* used to identify a torrent as a whole is calculated as the SHA-1 hash of the bencoded `info` dictionary, which is part of the metainfo file.

2.1.3. Tracker Server

The biggest problem of BitTorrent is to learn about the contact information of fellow peers. The traditional solution is a tracker server, where peers announce their participation in the torrent swarm and receive a list of other peer's IP addresses and port numbers in one step. 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 [RFC 3986, § 2.1].

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

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

Key	Explanation
`announce`	This is the URL of the tracker server, which usually has the format `http://<host>:<port>/announce`.
`info`	This dictionary describes the torrent's contents, its keys are explained below.
`info/name`	In case of of a single file, this is the file name the data is stored with when downloaded, otherwise the directory name. This key is optional.
`info/piece length`	The number of bytes of each piece.
`info/pieces`	For each piece a SHA-1 hash value calculated. Their raw bytes are concatenated and stored here. The total number of pieces can be derived from this value by dividing its length by 20, since a SHA-1 hash is 20 bytes.
`info/length`	In single file mode, this is the total file size in bytes, otherwise it's not present. The value is not used in this research.
`info/files`	In multi file mode, this is a list of dictionaries with information about every file, otherwise it's not present. The keys of these dictionaries are described below, but are not used in this research.
`info/files/length`	The size of this file in bytes.
`info/files/path`	This is the file's path and name, represented as a list of strings. All but the last item are directory names, the last item is the file name.

Table 2.1.: Structure of nested dictionaries in the bencoded metainfo file format

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 pieces 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 [BEP 23]; 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 explaining 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.

2.1.4. UDP Tracker Protocol

Tracker servers are the only centralized infrastructure required by traditional BitTorrent. Hence it is advisable to reduce bandwidth 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 % [BEP 15]. Since UDP datagrams may arrive out of order, the client sends a randomly chosen 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 source 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, renders this impossible. The connection ID is valid within the next minute.

announce This announce request includes the same parameters as the HTTP communication 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.

scrape 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 chosen by the server freely. An error response package containing a human-readable message may be sent by the server at any time.

2.1.5. 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 behavior and extensions, as well as the torrent's info hash and the peer's ID. The connecting client sends its handshake message first. All following messages begin with an overall length prefix, followed by a type identifier 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, still a central server keeping track of all peers is needed. 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* [BEP 5] solves this issue, as it stores peer contact information in a distributed hash table. Participating peers run a separate DHT *node*, which communicate sending bencoded messages over the User Datagram Protocol. The distributed hash table follows the Kademlia design as described by Maymounkov and Mazières in 2002 [13].

Nodes First, a node generates his own 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 maintains 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, with greater detail close to their own identifier.

Lookup The process of extracting peers for a given info hash proceeds 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 node to verify the announce request's source IP address and updating its hash table.

Integration The presence of DHT support is advertised in the standard BitTorrent handshake of the Peer Wire Protocol using the last bit of the eight reserved bytes. Peers receiving this indicator should send a `port` message, containing their own UDP node port number. This technique helps populate routing tables, especially on newly installed systems with an empty table.

2.3. Magnet Link

The concept of a *magnet link* described in BEP 9 [BEP 9] is used to create a uniform resource identifier for torrents of minimal size, in comparison to the metainfo file format. Its only mandatory component is the info hash, the SHA-1 value of the info dictionary. The link uses the `magnet:` URI scheme and stores info hash, a display name and tracker announce URLs in the query string. The emerging problem is the loss of the info dictionary's content. Since a tracker URL is optional, the metadata must be obtained from other peers.

2.3.1. Extension Protocol

To expand the functionality of the BitTorrent Protocol, the *Extension Protocol* was defined [BEP 10]. It introduces the generic `extended` message to the Peer Wire Protocol, which itself can have various subtypes depending on which extensions are actually used. Extension Protocol support is indicated in the standard Peer Wire Protocol handshake by setting the 20th bit from the right of the eight reserved bytes.

When both peers ascertain support for the Extension Protocol, extended messages containing a second handshake are exchanged. The handshakes contain a dictionary with one or multiple keys: First, a bencoded dictionary `m` with names of supported extensions together with randomly assigned extension IDs and secondly, additional keys depending on the used extensions. This setup allows for an arbitrary number of extensions with dynamic IDs, without the need for a global registry of extensions. Further extended messages contain the extension ID and payload as defined by the respective extension.

2.3.2. Extension for Peers to Send Metadata Files

The *Extension for Peers to Send Metadata Files* [BEP 9] with identifier `ut_metadata` is the first and only extension described here to make use of the Extension Protocol. It places one additional item in the handshake dictionary, namely `metadata_size`, containing the size of the bencoded info dictionary in bytes. For transmission, the bencoded info dictionary is divided in pieces of 16 kibibytes. The number of metadata pieces follows from the `metadata_size` parameter.

To gather initial contact information of peers to be asked for metadata, the DHT protocol may be used. Then every piece must be requested separately from these peers, using a `request` message. These are answered by the same number of `data` or `reject` messages, depending on whether the opposing client is able to deliver the piece. When all pieces are present, they can be composed and checked against the info hash value.

2.4. BitTorrent and German Law

2.4.1. Illegal Content

While there are no legal restrictions on using BitTorrent in general, the download of content without permission of the author or right holder is considered an illegitimate reproduction according to the German Copyright Act [UrhG, art. 15 (1), 16]. The common exception of private copying [UrhG, art. 53] is not applicable here, since the source is "obviously unlawfully-produced".

Even more serious is the upload process always involved in BitTorrent. Illegitimate distribution of proprietary content may be sentenced with imprisonment or a fine [UrhG, art. 106]. More common and often abused [17] is the system of special notifications [UrhG, art. 97a], sent from right holders to assumed copyright infringers. These warning letters are supposed to settle the controversy extrajudicial in exchange of a fee. Entitlement of right holders to indemnity and expense allowance exists [UrhG, art. 97].

2.4.2. Collecting IP addresses

Privacy is regulated by the Federal Data Protection Act [BDSG] in Germany. It introduces a concept of personal data which includes "any information [...] of an [...] identifiable individual" [BDSG, sec. 3 (1)]. The collection of personal data is inadmissible without consent of the concerned person [BDSG, sec. 4]; other exceptions permitted by this Act do not apply. It is disputed whether IP addresses are within the definition of personal data [10], so to comply with the law by all means the anonymization of IP addresses is preferable.

3. Implementation

The software tool written for this thesis was developed using the version control system *Git* [21] in conjunction with a private online repository provided by GitHub. The source code was published [16] under the *GNU General Public License* in version 3.

3.1. Dependencies

There are a few external dependencies, which are all free and open-source software. The *BencodePy* project by Eric Weast [23] provides an encoder and decoder for bencoding messages and values. The *Object Relational Mapper* of *SQLAlchemy* [2] is used to store evaluation results in the *SQLite* database format [6]. The *GeoIP2 API* [11] is used to perform IP geolocation lookups in the *GeoLite2 City Database* [12]. 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 [7] 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.

For reference, a scrape request is sent to the tracker server every few minutes in order to compare our own counting of peer downloads with numbers as reported by the tracker server.

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 [BEP 3], the announce URL, info hash, pieces count and pieces size are extracted.

All magnet links [BEP 9] 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 info hash, this information must be retrieved from the swarm of other peers. A few peer addresses are gathered with a DHT lookup, then peers are contacted sequentially until the info dictionary could be received using the Extension Protocol [BEP 10] and the *ut_metadata* extension [BEP 9]. 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 parallel, whereby the number of threads can be set in `config.peer_evaluation_threads`. The queue is sorted ascending, so if timestamps lie in the past the peer which is due the longest time is chosen, 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 chosen.

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 info hashes are ignored.

Evaluate Download Progress Once a connection is established with a peer, its download progress must be determined only using peer messages as defined by the BitTorrent Protocol [BEP 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 separate 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 acquire the download progress is to compile a combined bitfield from these messages and count the present pieces. The number of total torrent pieces 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 download'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* [11], 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.

Secondary Statistics TODO probably out of date

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 contacts at later try and successful evaluated incoming peers.

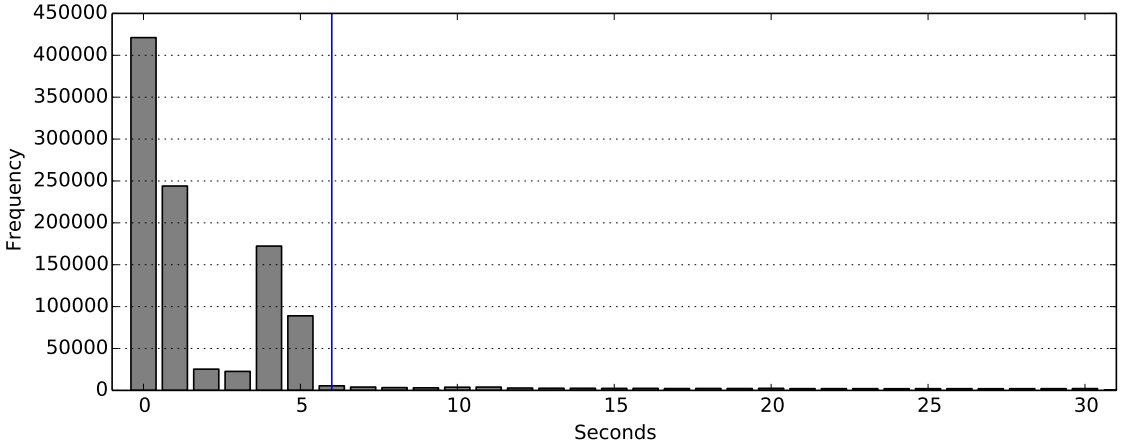


Figure 3.1.: Duration of receiving one peer message

...

Explain implementation of the features:
See btda/README.md/Features

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 six 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. Message collection is considered to be complete after the timeout finished without receiving a message.

To assess a minimum timeout, an analysis with special configuration parameters was performed[1]. Here the maximum time used for receiving one message was recorded for every peer contact. These durations were rounded and the number of occurrences plotted in figure 3.1. For this task `config.network_timeout` was set to 30 seconds to achieve most unbiased results. During this test at 977,301 out of 1,040,817 peer contacts the maximum duration for receiving one message was below six seconds, which equals 93.9 %.

`config.torrent_complete_threshold` ...

`config.peer_evaluation_threads` ...
TODO: Number of threads: workload plot

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
- Drawback: One can only derive lower bounds from observing peers using the standard BitTorrent protocol

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* [7] 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.

[1] Files: `2015-08-14_17-46-44_faui1-246.sqlite`, `2015-08-14_17-46-44_faui1-246_timeout.txt`

4. Evaluation

Data collection with the BitTorrent Download Analyzer tool was performed on a virtual machine running Ubuntu 14.04 LTS with 1.0 GB of RAM, a 3.4 GHz processor and an own IPv4 address without NAT. The 15 chosen torrents were analyzed at once concurrently using **512 threads**. The analysis was performed from 11:11 am, August 15* to 11:11 am, August 17*, 2015. A time period of **48 hours** was chosen in order to detect patters during the day-night cycle. Necessary measures to ensure valid results were taken: The DHT node provided by *pymdht* did not crash before or during the analysis. The BitTorrent Download Analyzer or any of its threads did not crash during the analysis and gave no relevant error message in the logfile.

4.1. Choosing Torrents

Due to the distributed nature of BitTorrent, there unfortunately is no complete list or for all torrents currently active. Thus external data about popularity of torrents is needed, even if there is no guarantee of correctness. So to determine most popular torrent directory sites the global traffic rankings by Alexa Internet, Inc. [1] were consulted. The websites where Alexa's ranking was looked up were collected through manual investigation using web search engines, relevant news sites and cross references between torrent sites. Table 4.1 shows the 17 sites found having an rank below 5,000.

Popular torrents were often found to be registered on multiple tracker sites, which leads to mostly identical top torrents across various torrent sites. For the definite selection of torrents, the meta-search engine Torrentz [20] was used: It monitors torrents from all other major torrent sites and provides sort and filter options by peer count, torrent age and size. Torrents in three size groups were chosen: The nine most popular torrents below 1 GB, the nine most popular torrents above 1 GB and the nine most popular torrents above 10 GB. These **27 torrents** are listed in table 4.2.

4.2. Getting Addresses of Peers

Peers from all sources were considered, namely from the tracker server, the DHT network and incoming connections. The number collected peer addresses in regards to their source is shown in table 4.3. Since the *incoming peers* data point only includes peers when their download progress was successfully evaluated, the ratio of 95 %* new incoming peers implies that $1 - 95\,\% = 5\,\%$* of unique incoming peers contacted us a second time, despite not receiving any torrent data.

Summary plot

The progression during the analysis period of all peer requests is shown in appendix A.1. The ratio between new and duplicate received peer address information settles fairly quickly*. *incoming-duplicate* is actually good.

4.3. Evaluating Peer's Download Progress

Define failure

Failure reasons: Unsuppored client (mTP), probably not, see Client Analysis, probably bad/out of date peer address data

Failure rate per source out of scope

Total unique active success from peer database

Rank	Site name	Domain name	Alexa Rank
1	Kickass Torrents	kat.cr	116
2	ExtraTorrent.cc	extratorrent.cc	335
3	Nyaa Torrents	www.nyaa.se	399
4	Torrentz Search Engine	torrentz.eu	464
5	The Pirate Bay	thepiratebay.se	507
6	YTS	yts.to	669
7	Rarbg	rarbg.to	1,150
8	1337x	1337x.to	1,661
9	EZTV	eztv.ch	1,831
10	torrentHound.com	www.torrenthound.com	2,188
11	IPTorrents	iptorrents.com	3,256
12	isoHunt	isohunt.to	3,816
13	Bitsnoop P2P Search	bitsnoop.com	4,293
14	Torrent Downloads	www.torrentdownloads.me	4,315
15	LimeTorrents.cc	www.limetorrents.cc	4,552
16	TamilRockers.net	tamilrockers.com	4,586
17	Monova Torrent Search	www.monova.org	4,843

Table 4.1.: Popularity of torrent directory sites according to Alexa's [1] global traffic ranking. Only sites with a rank below 5,000 are listed. Data is accurate as of July 16, 2015.

Peer addresses received via DHT and tracker requests were contacted continuously. 0 % of contacted peers failed when contacted for the first time. Of the peers contacted successfully once, 0 % failed later.

4.4. Counting Confirmed Downloads

The timeline of confirmed downloads and downloads reported by the tracker server in scrape requests is shown in one hour steps in figure A.2.

Map: Distribution of downloads (TODO)

4.5. Peer's Analysis

4.5.1. Download Speed

TODO: Graph of speed per country with standard derivation indicators, n>=10 per country

todo: Map of Distribution of download speeds

4.5.2. BitTorrent Clients

TODO: Table of frequency of BT clients

4.5.3. Peer's Host Names

TODO: Table: Examine hostnames by ISP or seedbox provicers

ID	Site name	Torrent name	Size	Leechers	Seeders
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
15	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23
1	Kickass Torrents	torrent_name	3.0 GB	42	23

Table 4.2.: * List of the 20 most popular torrents according to meta-search engine TORRENTZ [20]. At the end are five torrents above 5 GB and five recently published torrents . Selection was made on August 42, 2015, 11:00. The *ID* was chosen by the SQLite database used for storage during the analysis.

Source	Total	Unique	New	Unique per Hour
Tracker server	5	2	3 %	
DHT network	10	2	3 %	
Incoming peers	1	2	3 %	

Table 4.3.: Total and unique received peer addresses per source. *Tracker server* and the *DHT network* are actively contacted for new peers. Values of *incoming peers* only include peers whose download progress was evaluated successfully. The duration to calculate the *unique per hour* value is 48 h*, as mentioned earlier.

Event	Reason	Quantity	Share per Event
First contact	timed out	1653	
	[Errno 111] Connection refused	420	
	Socket connection broken	317	
	[Errno 104] Connection reset by peer	47	
	[Errno 113] No route to host	26	
	Total	4321	
Later contact	Socket connection broken	265	
	timed out	27	
	[Errno 111] Connection refused	6	
	[Errno 104] Connection reset by peer	2	
	Total	4321	
Incoming peer	Peer speaks unknown protocol	891	
	timed out	353	
	[Errno 104] Connection reset by peer	9	
	Socket connection broken	7	
	Total	4321	

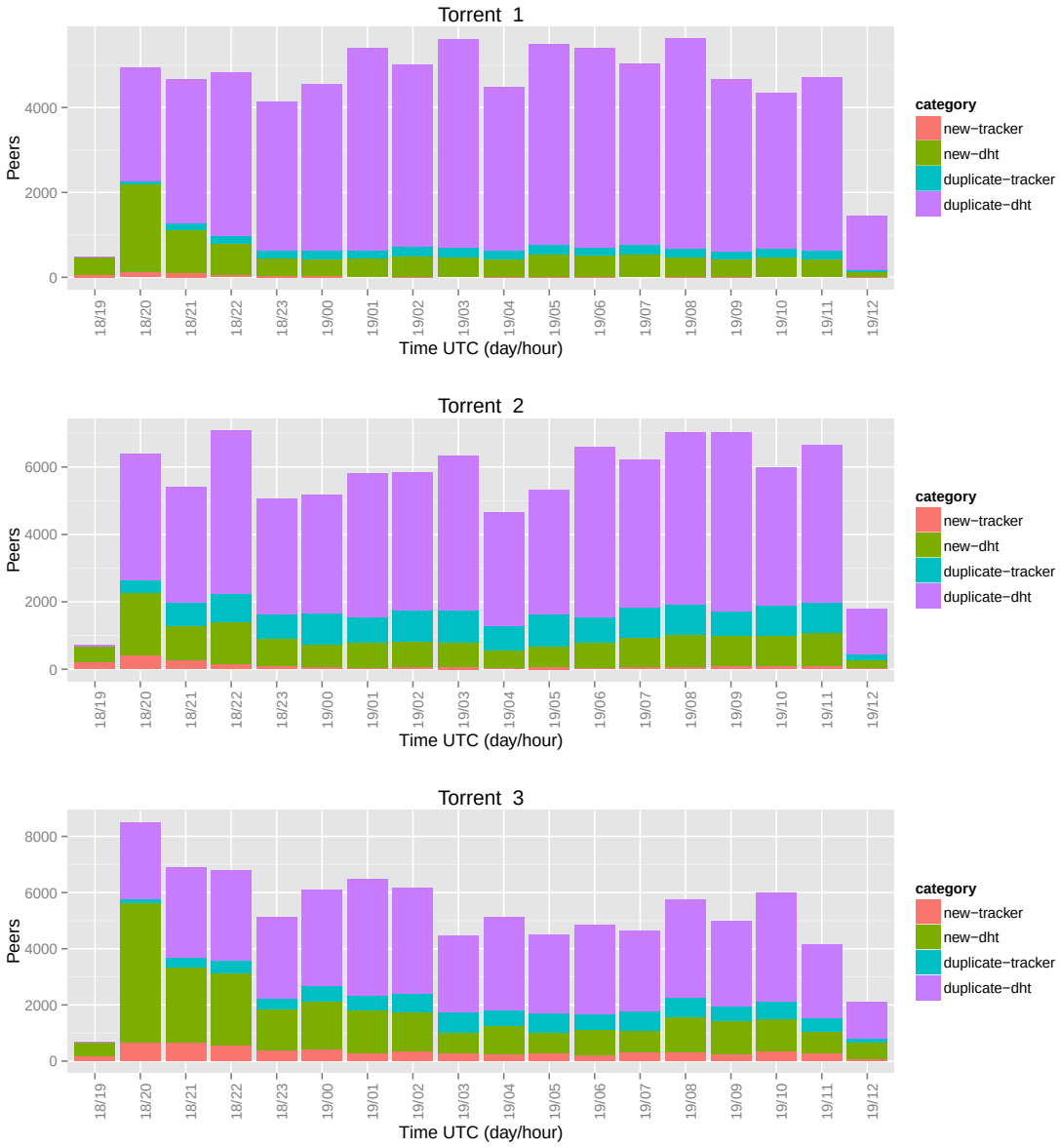
Table 4.4.: Reasons for failed peer valuations during the analysis process. *Event* describes the point in the evaluation process, when the problem occurred. *First* and *later contact* are performed when actively contacting collected peer addresses, while a peer must be contacted once successfully to be contacted another time. An *incoming peer* represents an incoming connection on the BitTorrent listening port.

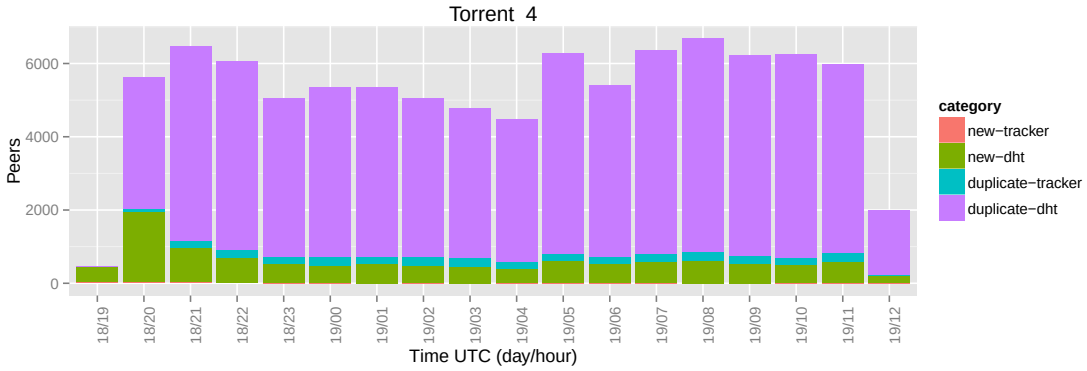
5. Conclusion and Future Work

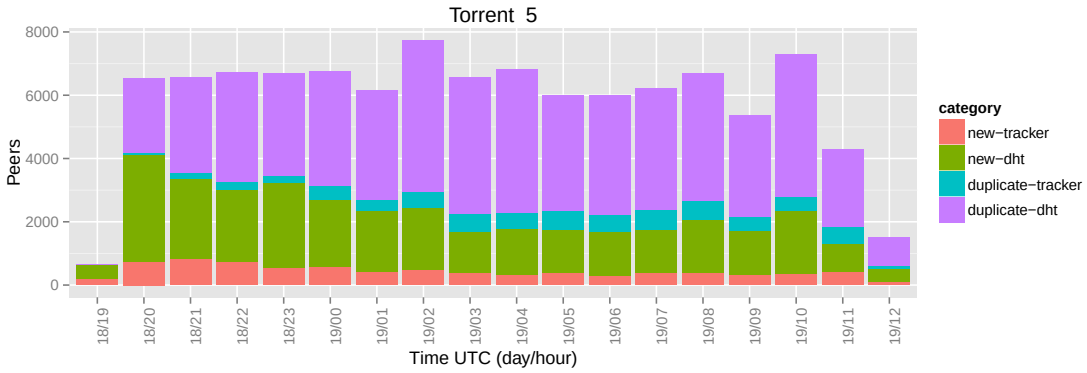
...

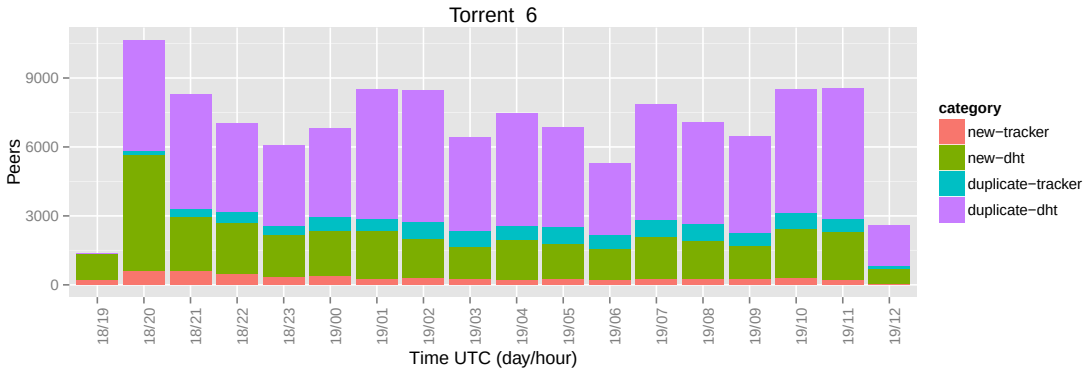
A. Collected Data

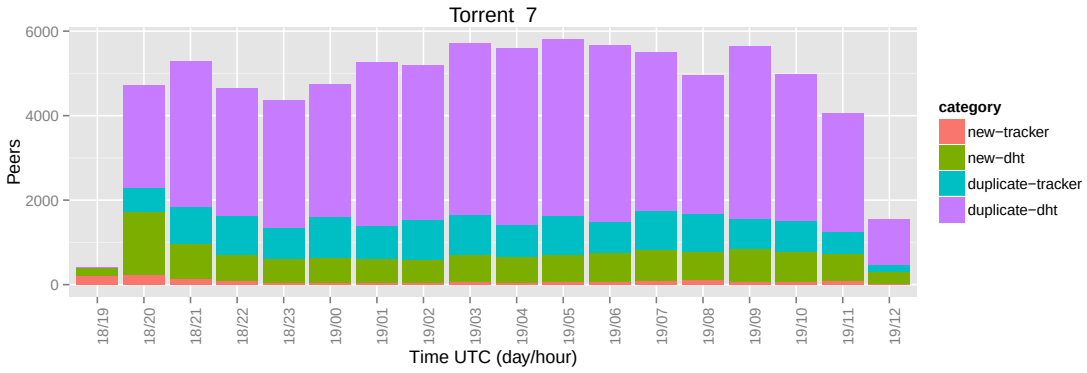
A.1. Development of received peer addresses per source

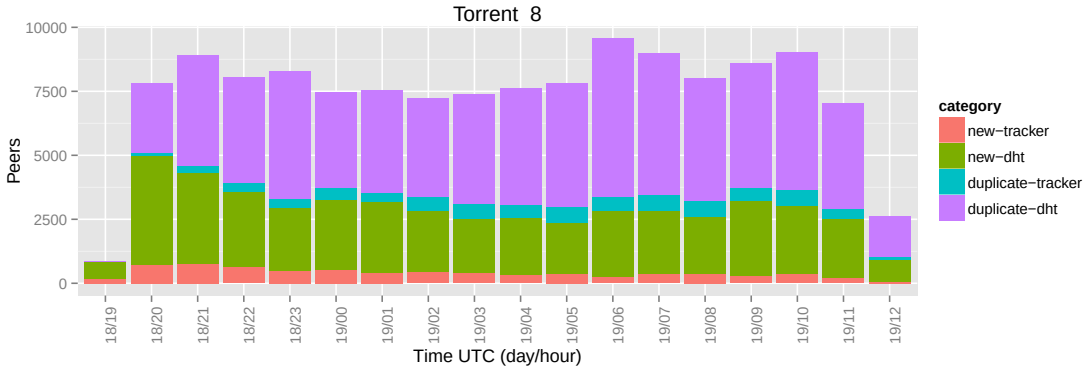



Torrent 4
Peers
0
2000
4000
6000
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
new−tracker
new−dht
duplicate−tracker
duplicate−dht

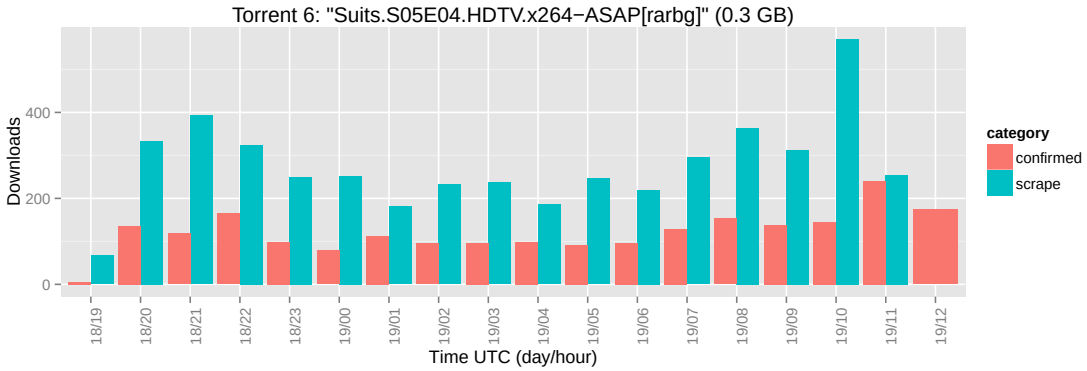

Torrent 5
Peers
0
2000
4000
6000
8000
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
new−tracker
new−dht
duplicate−tracker
duplicate−dht

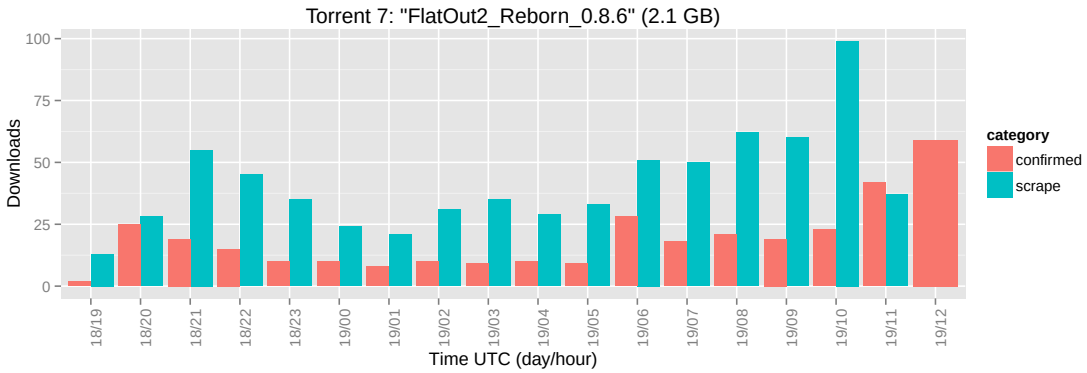

Torrent 6
Peers
0
3000
6000
9000
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
new−tracker
new−dht
duplicate−tracker
duplicate−dht

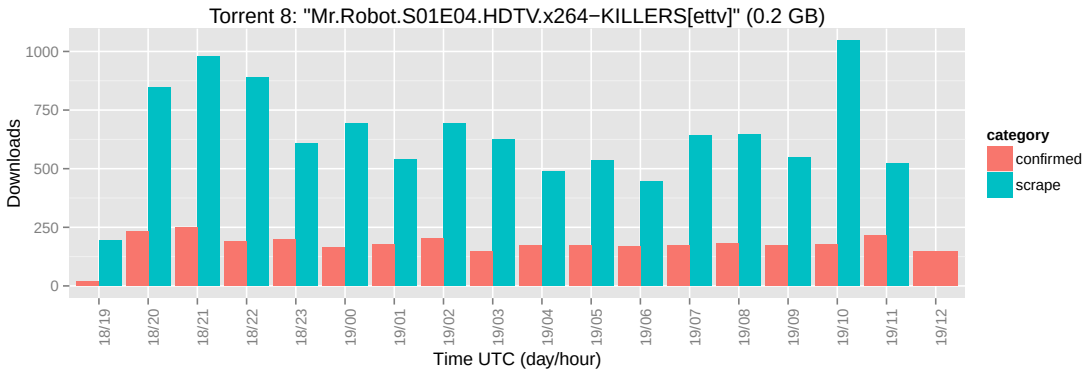

Torrent 7
Peers
0
2000
4000
6000
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
new−tracker
new−dht
duplicate−tracker
duplicate−dht

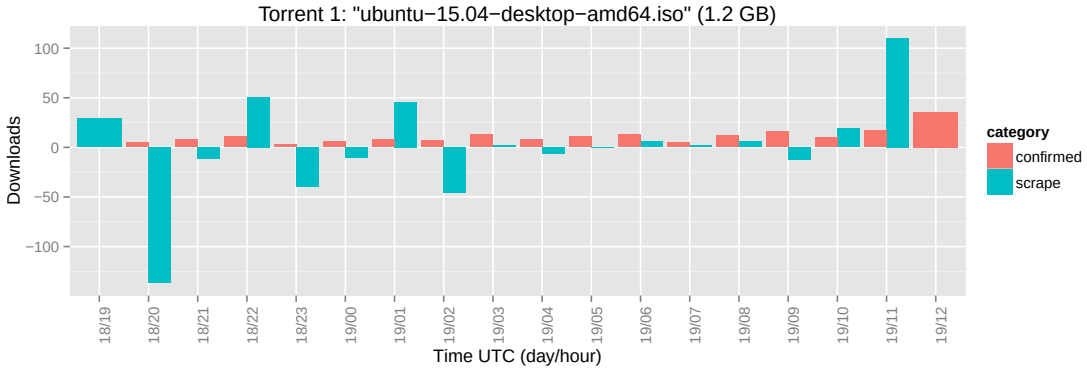


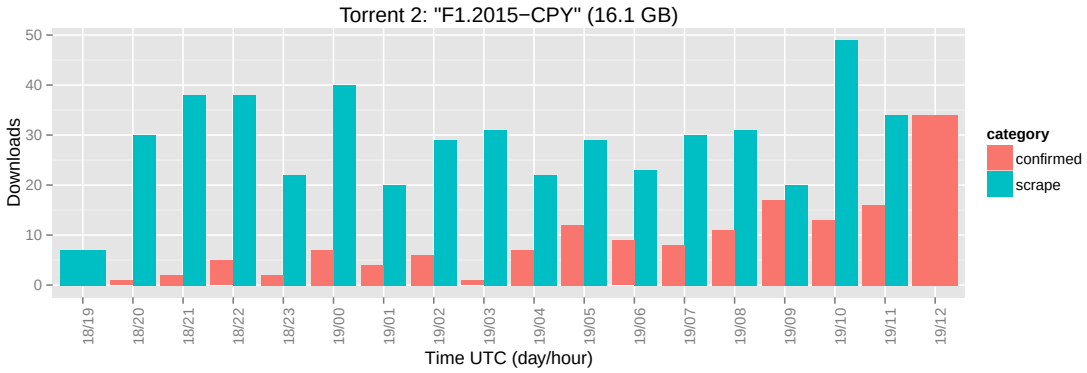
A.2. Confirmed and server reported download numbers

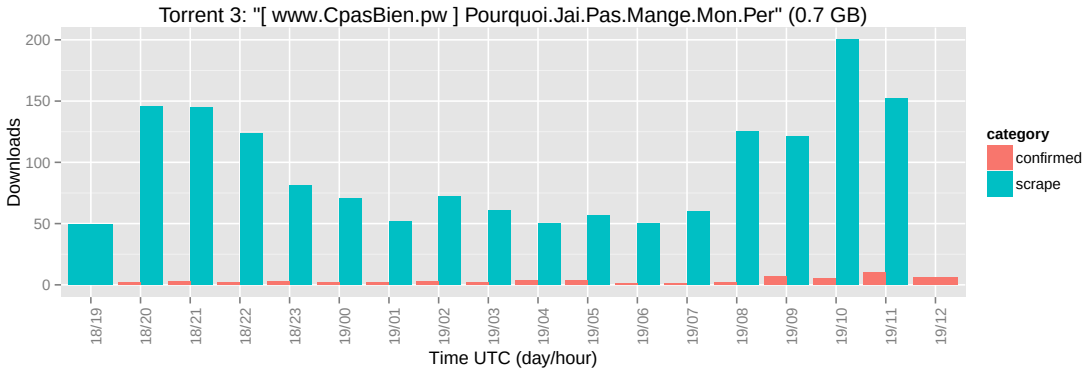


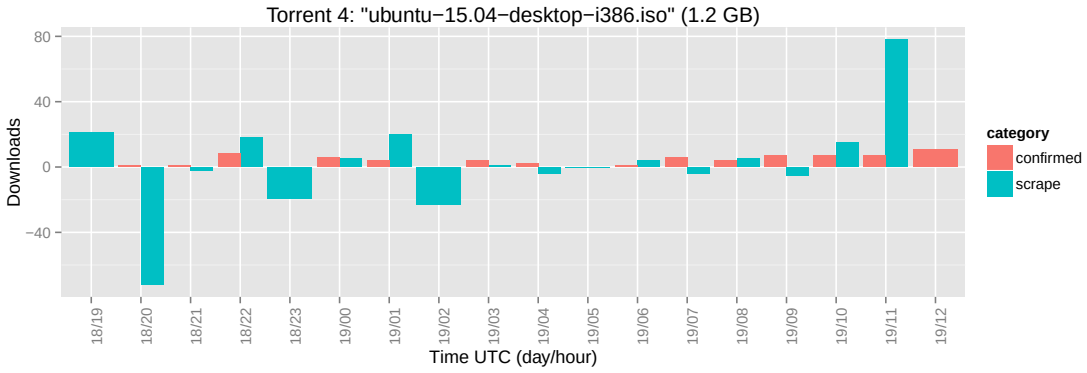


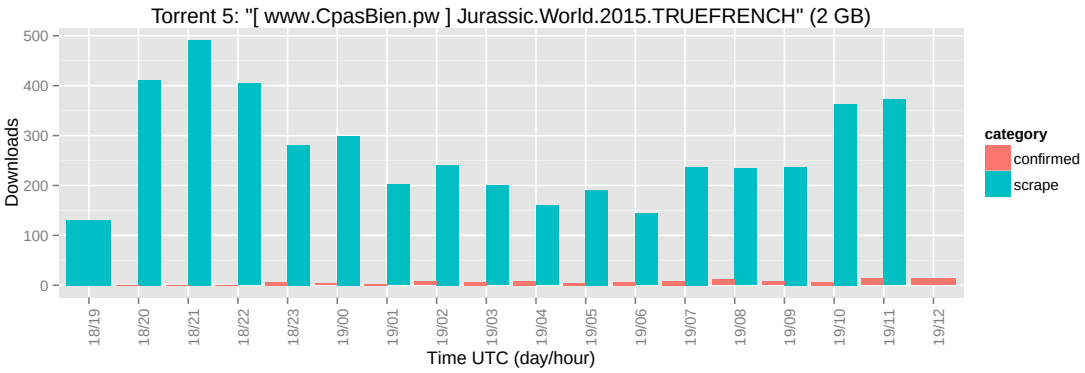



Torrent 1: "ubuntu−15.04−desktop−amd64.iso" (1.2 GB)
Downloads
100
50
0
−50
−100
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
confirmed
scrape


Torrent 2: "F1.2015−CPY" (16.1 GB)
Downloads
50
40
30
20
10
0
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
confirmed
scrape


Torrent 3: "[www.CpasBien.pw] Pourquoi.Jai.Pas.Mange.Mon.Per" (0.7 GB)
Downloads
200
150
100
50
0
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
confirmed
scrape


Torrent 4: "ubuntu−15.04−desktop−i386.iso" (1.2 GB)
Downloads
80
40
0
−40
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
confirmed
scrape


Torrent 5: "[www.CpasBien.pw] Jurassic.World.2015.TRUEFRENCH" (2 GB)
Downloads
500
400
300
200
100
0
18/19
18/20
18/21
18/22
18/23
19/00
19/01
19/02
19/03
19/04
19/05
19/06
19/07
19/08
19/09
19/10
19/11
19/12
Time UTC (day/hour)
category
confirmed
scrape

Bibliography

[1] Inc. Alexa Internet. *Alexa Site Overview*. 1996. URL: http://www.alexa.com/siteinfo.

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

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

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

[3] Brett Danaher and Joel Waldfogel. "Reel piracy: The effect of online film piracy on international box office sales". In: *Available at SSRN 1986299* (2012).

[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.

[UrhG] juris GmbH. *Act on Copyright and Related Rights (Copyright Act)*. Ed. by Ute Reusch. 2014. URL: http://www.gesetze-im-internet.de/englisch_urhg/englisch_urhg.html.

[BDSG] juris GmbH. *Federal Data Protection Act*. Ed. by Language Service of the Federal Ministry of the Interior. 2014. URL: http://www.gesetze-im-internet.de/englisch_bdsg/englisch_bdsg.html.

[5] Robert G Hammond. "Profit Leak? Pre-Release File Sharing and the Music Industry". In: *Southern Economic Journal* 81.2 (2014), pp. 387–408.

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

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

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

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

[8] Dave Levin et al. "Bittorrent is an auction: analyzing and improving bittorrent's incentives". In: *ACM SIGCOMM Computer Communication Review*. Vol. 38. 4. ACM. 2008, pp. 243–254.

[9] Thomas Locher et al. "Free riding in BitTorrent is cheap". In: *Proc. Workshop on Hot Topics in Networks (HotNets)*. Citeseer. 2006, pp. 85–90.

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

[10] Patrick Lundevall-Unger and Tommy Tranvik. "IP addresses–Just a Number?" In: *International Journal of Law and Information Technology* (2010), eaq013.

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

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

[13] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Peer-to-Peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5. URL: http://dx.doi.org/10.1007/3-540-45748-8_5.

[14] Jordi McKenzie. "Illegal music downloading and its impact on legitimate sales: Australian empirical evidence". In: *Australian Economic Papers* 48.4 (2009), pp. 296–307.

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

[15] Sandvine. *Global Internet Phenomena Report 2H 2014*. Study. Sandvine, 2014. URL: https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf.

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

[17] Sandra Schmitz and Thorsten Ries. "Three songs and you are disconnected from cyberspace??? Not in Germany where the industry may 'turn piracy into profit'". In: *European Journal of Law and Technology* 3.1 (2012). ISSN: 2042-115X. URL: http://ejlt.org/article/view/116.

[18] Stephen E Siwek. *The true cost of sound recording piracy to the US economy*. Institute for Policy Innovation, 2007.

[19] Michael D Smith and Rahul Telang. "Piracy or promotion? The impact of broadband Internet penetration on DVD sales". In: *Information Economics and Policy* 22.4 (2010), pp. 289–298.

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

[20] *Torrentz Search Engine*. 2003. URL: https://torrentz.eu/.

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

[22] 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.

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

[24] Chao Zhang et al. "Unraveling the bittorrent ecosystem". In: *Parallel and Distributed Systems, IEEE Transactions on* 22.7 (2011), pp. 1164–1177.