Lehrstuhl für Informatik 1
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Bachelor Thesis

Analysis of BitTorrent Trackers and Peers

Counting Confirmed Downloads in BitTorrent

Stefan Schindler

Erlangen, August 28, 2015

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

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 28, 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 enables software clients to verify received data. This makes the protocol robust against transmission errors and malicious peers trying to distribute manipulated 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 [11] 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 and household, 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 barely used.

File sharing 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" [13, table 1] in 2006 regarding music sales only. However, the harm of illegal downloads is unclear [4]. Other studies suggest delayed digital releases promote piracy [2] or find no [9] or even positive [14] 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 [16] from the University of Ballarat, Australia, finds numbers between 90 % and 97 %.

1.1 Motivation

When assessing popularity or peer numbers, previous studies relied on information reported by tracker servers. So called *scrape requests* allow to ask for statistics about a specific or even all torrents a tracker is managing. When successful, servers answer with the torrent-identifying info hashes together with the number of current downloaders, current uploaders and finished downloads since the torrent was registered with the server. In a second step, one can use the info hashes to crawl peer addresses from tracker servers for further analysis. This data may be to their best knowledge, but there is no possibility for verification.

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 at 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's download 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 the task at hand, 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* [11]. *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* [BEP 9].

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 the appropriate protocol, either TCP or UDP. Likewise requests for peers of the given torrent are issued in 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 peers will be evaluated equally, but can only be counted when connecting at least twice – once with progress below and once above the threshold. Statistics about received duplicate versus unique 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 [5], but until now this has not surfaced as a problem at large. Before closing the connection, a message announcing the port of the own DHT node is sent to the peer in order to popularize the own node and fill its 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

There is numerous research about the scope of BitTorrent. Like mentioned before, analysis relies on information crawled from tracker servers. Watters, Layton, and Dazeley [16], emphasizing the copyright infringing use of BitTorrent, relied solely on information from scrape requests, examining the number of seeders. More detailed results can be obtained by assembling a dataset of real peers, by looking up peer addresses on tracker servers. This approach was taken by Drachen, Bauer, and Veitch [3] in 2011. While focusing on a sample of 173 video games, they found an average of 537 thousand unique peers per game among the top ten games over a three month period. These top 10 games occupied 41.8 % of all peers observed.

The same approach was taken by Zhang et al. [17], also in 2011. The study *Unraveling the BitTorrent Ecosystem* published by the IEEE associated research group claims to include "the large majority of torrents in the public (English-language) ecosystem". With detailed description of the used methodology, in a 12 hour window they counted 5.1 million unique peers in 1.2 million active torrents and 728 active trackers. Only 1 % of these torrents had over 100 peers and 44 % of peers were found to be active in multiple torrents. Coverage achieved in this thesis is not comparable by far. Instead, the concept of counting confirmed downloads will be demonstrated and examined based on a small set of popular torrents.

Further notable related research areas concentrate on extent [6] and punishment [5, 1] of free-riding peers, who do not upload any data after downloading from other peers, or the special implications of private BitTorrent communities [10], which promise higher download speeds by enforcing upload to download ratios.

1.4 Results

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

1.5 Outline

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

1.6 Acknowledgments

I want to thank Philipp Klein for discussing methods and implementation as well as troubleshooting the virtual machine used for data collection, Michael Gruhn for discussing ideas, and the RRZE for providing the virtual machine and handling any unjustified copyright warning letters.

2 Background

This chapter explaines technologies and specifications utilized during this research project. Section 2.1 explains the basic application of BitTorrent principles, from the `.torrent` file to downloading content. Sections 2.2 and 2.3 go into detail about the trackerless operation of BitTorrent with the DHT network and magnet links. Eventually, file sharing will be discussed concerning relevant German copyright and privacy laws in section 2.4.

2.1 BitTorrent Protocol

BitTorrent is specified in currently 42 [BEP 0] BitTorrent Enhancement Proposals (BEP), most of them being extensions for special use cases. A comprehensive overview of the basics is also given in a wiki provided by Theory.org [26]. The following sections describe the essential parts based on the definitions of [BEP 3]. Now, the goal of BitTorrent is the distribution of a predefined set of files among an arbitrary number of recipients. Overwhelming load on a central entity is avoided by splitting the file set in pieces and let peers send them to each other. Three main parts are necessary to enable the process:

1. The BitTorrent file, which contains identifying metadata about the file set. It is usually distributed via torrent indexing websites or between users directly.
2. The tracker server, where peers can learn IP addresses and port numbers of other peers.
3. The Peer Wire Protocol, which is spoken between peers.

2.1.1 Bencoding

In order to store and transmit common data structures, an encoding is required to preserve the data's type and semantic. To realize BitTorrent, Cohen came up with *bencoding* to annotate data appropriately. When bencoded, a value's length is detectable by specific beginning and ending delimiter characters: Integers, lists and dictionary are prefixed with small letters `i`, `l` and `d` respectively and closed with an `e`. Strings have a length prefix. Details and examples are provided in table 2.1.

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*

Type	Encoding	Example
String	`<length>:<string>`	`3:abc` = `"abc"`
Integer	`i<integer>e`	`i23e` = `23`
List	`l<val1><val2>e`	`l3:abci23ee` = `["abc", 23]`
Dictionary	`d<key1><val1><key2><val2>e`	`d3:abci23ee` = `{"abc": 23}`

Table 2.1: Data types of Bencoding with examples. Any integers and length information is encoded in base 10 ASCII format. Lists and dictionaries are composite data types, so they can contain any other bencoded values. This allows nested dictionaries or lists. Note that only strings can be used as dictionary keys.

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

file, which is using the `.torrent` file name extension. For easy reference, values are grouped in dictionaries as listed in table 2.2. These files can easily shared between users and allow them 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. An authentic torrent file of a Debian image is printed below. It contains some additional keys like a comment and a creation date. The hashes of the `info/pieces` key were removed and dictionary keys were highlighted:

```
1 d8:announce41:http://bttracker.debian.org:6969/announce7:comment35:"Debian CD
from cdimage.debian.org"13:creation datei1429970901e9:httpseedsl84:http://cdimage
.debian.org/cdimage/release/8.0.0/iso-dvd/debian-8.0.0-amd64-DVD-1.iso84:http://
cdimage.debian.org/cdimage/archive/8.0.0/iso-dvd/debian-8.0.0-amd64-DVD-1.isoe4:
infod6:lengthi3976200192e4:name28:debian-8.0.0-amd64-DVD-1.iso12:piece length
i1048576e6:pieces75840:<hashes>ee
```

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 standard HTTP. The request is sent with the parameters shown in table 2.3, whereby keys and values must be quoted using percent-encoding [RFC 3986, § 2.1]. An exemplary request would be:

```
1 http://bttracker.debian.org:6969/announce?port=6881&uploaded=758&info_hash=W%E1Y%
A5%82a%C8%D2%F4%2Ad%98%0D%2B%80%8E9%01%FC%F6&peer_id=hNsfr5PYlFtWO73yvSGX&compact
=1&event=started&left=1896&downloaded=1896
```

Key	Explanation
`info_hash`	This is the SHA-1 hash of the bencoded info dictionary from the metainfo file.
`peer_id`	A string of 20 bytes is chosen by each peer. It contains client software information by convention, see section 4.4.2.
`ip`	In case the client uses a proxy, the peer's original routable IP address can be submitted in this optional parameter.
`port`	This is the port number the peer is listening on for connections from other peers. [BEP 3] recommends a port between 6881 and 6889.
`uploaded`	Amount of pieces this peer has uploaded so far.
`downloaded`	Amount of pieces this peer has downloaded so far.
`left`	Amount of pieces this peer has left to download.
`event`	The current download status can be communicated in this optional key. Valid values are `started`, `completed` or `stopped`.
`compact`	Indicates weather or not the tracker should respond with a normal or compact peer list to save bandwidth. Allowed are `0` or `1`.

Table 2.3: Structure of a HTTP announce request from a peer to a tracker server. A compact peer list is defined in [BEP 23]: Addresses are all concatenated, while six bytes per peer are used, four for the IPv4 address and a two for the port. Some trackers dismiss the `compact` key and always send compact peer lists.

Key	Explanation
`failure reason`	In case of failure, this human-readable error message explains why the request could not be fulfilled.
`interval`	A suggested interval in seconds the client should wait between tracker requests.
`peers`	Normally this is a list of dictionaries, one per peer. Its keys are described below. In case of a compact peer list as described in table 2.3, this is a single byte string instead, and further keys are not used.
`peers/peer id`	The self-selected peer ID, as described in table 2.3.
`peers/ip`	The peer's IP address.
`peers/port`	The peer's port number.

Table 2.4: Structure of a bencoded response from a tracker to a peer's announce request. The values are structured in a dictionary.

connect		announce	
Request	Response	Request	Response
`connection_id`	`action`	`connection_id`	`action`
`action`	`transaction_id`	`action`	`transaction_id`
`transaction_id`	`connection_id`	`transaction_id`	`interval`
		`info_hash`	`leechers`
		`peer_id`	`seeders`
		`downloaded`	`IP address`
		`left`	`TCP port`
		`uploaded`	`IP address`
		`event`	`TCP port`
		`IP address`	`...`
		`key`	
		`num_want`	
		`port`	

Table 2.5: Request and response packages for the actions *connect* and *announce* in the UDP tracker protocol as specified in [BEP 15]. On the initial *connect request*, the constant 41,727,101,980 is used as a `connection_id`. The `connection_id` returned by the server in the *connect response* is used for later request like the *announce request*. Numerical values for the `action` parameter are 0 during *connect* and 1 during *announce*. IP addresses and ports in the *announce response* are all concatenated and use six bytes per tuple.

In the HTTP message body of the tracker's response, the list of peers is returned in a bencoded dictionary with keys as explained in table 2.4.

2.1.4 UDP Tracker Protocol

Tracker servers are the only centralized infrastructure required in the traditional implementation of BitTorrent. Hence it is advisable to reduce bandwidth during tracker requests as much as possible. As [BEP 15] demonstrates, using the *UDP tracker protocol* instead of HTTP over TCP can reduce traffic by 50 %. The protocol defines three different types of requests a client can send to the server: connect, announce and scrape. When communicating with a tracker server, first of all a connect request must be sent. Each of these requests is answered by the server with a specific response. Transmitted values of the connect request and response, as well as the announce request and response are shown in table 2.5. 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. The three types of requests work as follows.

connect The first step in the UDP tracker protocol is a connect request. It is used to obtain a connection ID from 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 The announce request includes the same parameters as the HTTP announce request described in section 2.1.3. Additional parameters are an unused `key` value and the `num_want` value, allowing to specify the amount of returned peers. The announce response now includes the number of active leechers and seeders in addition to the peer data.

scrape Finally a scrape request is defined. Its setup is similar to the announce request, but now shown in table 2.5. The scrape response gives clients access to the numbers of leechers, seeders and

Type	ID	Contents
handshake	—	length prefix, length of protocol string, protocol string, eight reserved bytes, info hash, peer id
choke	0	—
unchoke	1	—
interested	2	—
not interested	3	—
have	4	piece index
bitfield	5	bitfield of present pieces
request	6	piece index, begin offset within the piece, length offset
piece	7	piece index, begin offset within the piece, block of piece data
cancel	8	piece index, begin offset within the piece, length offset
port	9	UDP DHT port
extended	20	extension id, payload

Table 2.6: Messages of the Peer Wire Protocol as defined in [BEP 3]. Port messages are part of the DHT protocol, see section 2.2. Extended messages are described later in section 2.3.1.

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. For all requests, 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 Wire Protocol is spoken between peers and allows bidirectional communication with predefined messages. At first, an initial handshake is exchanged, containing a protocol description string, eight reserved bytes for alternate protocol behavior and extensions, as well as the torrent's info hash and the sending peer's ID. The connecting client sends its handshake message first. All messages but the handshake are sent with an overall length prefix, followed by a numerical type identifier and the payload, if appropriate. An overview about defined messages is given in table 2.6.

Following the Tit-for-Tat principle of BitTorrent, a peer should try to appear interesting to the remote peer, to encourage him not to close the connection but to deliver pieces of content. This can be done by offering pieces himself, with the *bitfield* message. It is sent immediately after the handshake to indicate which pieces a peer has already downloaded and verified. When a peer has downloaded additional pieces while the connection was alive, *have* messages are sent to all connected clients to update the catalog of available pieces.

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 of a peer to fulfill requests for pieces of a remote peer, for instance for bandwidth management. Similarly *interested* and *not interested* indicate whether a peer would start downloading if unchoked, to allow the remote peer to unchoke the right peers. To demand a piece which is present at the remote peer, the *request* message is used and answered with the requested data within a *piece* message. When requests were sent to multiple clients beforehand to increase download speed, a *cancel* message is used to revoke a request.

2.2 DHT Protocol

Despite the complete file payload being transmitted from client to client, a central server keeping track of all peers is still needed in the setup described until now. 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 (DHT). Participating peers run a separate DHT *node*, which

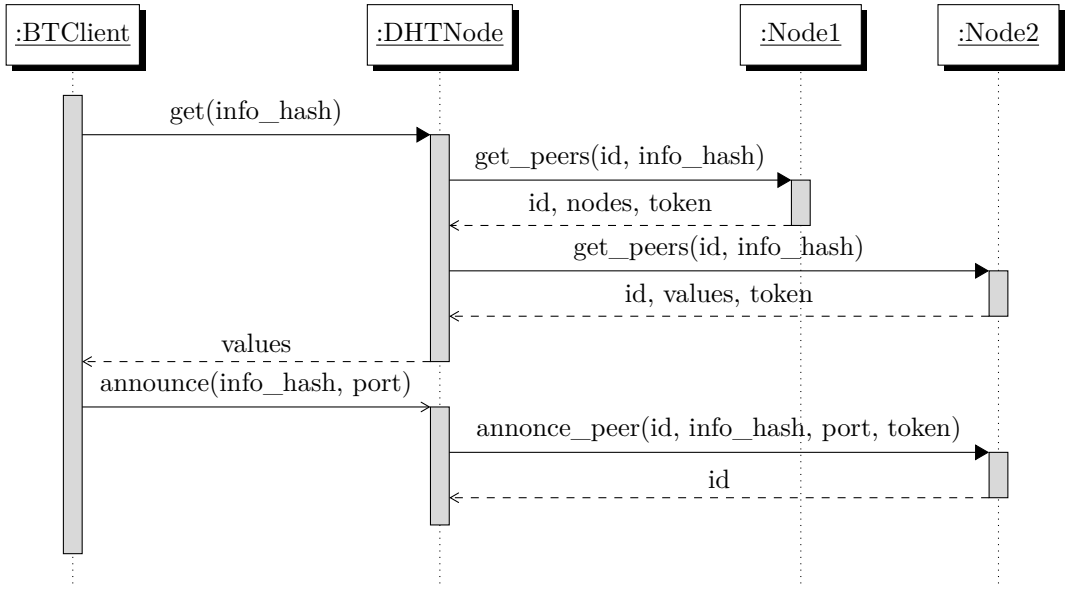

Figure 2.1: A sequence diagram of a request for peers in the DHT network as described in [BEP 5]. *BTClient* asks its own *DHTNode* for peers for a specific info hash. The DHT node issues a request to remote *Node1*, wich does not know any peers and answers with information about other nodes, including *Node2*, instead. *Node2* knows about peers and delivers the desired information to *DHTNode*. Finally, the participation in downloading the torrent is announced to *Node2*. The variable *values* refers to a list of IP addresses and TCP ports of peers.

communicates by sending bencoded messages over UDP. The DHT used in BitTorrent follows the Kademlia design as described by Maymounkov and Mazières [8] in 2002.

Nodes First, a node generates his own random 20 byte identifier, called node ID. Each node maintains a *routing table*, which maps IDs of other nodes to their corresponding IP address and UDP port number. The closer these node IDs are to the node's own ID, the more nodes are stored in the routing table. For this measurement, the distance between two node IDs is defined as the bitwise exclusive disjunction interpreted as an unsigned integer. Therefore, most entries in the routing table have close proximity to the node's own ID. Similarly, a separate table is maintained, where torrent info hashes with close proximity to the own node ID are mapped to IP addresses and TCP ports of peers known to download this torrent. This second table is part of the *distributed hash table*.

Lookup The process of extracting peers from the DHT network for a given info hash proceeds iteratively. The same distance metric as described above, is now used to identify nodes in the routing table with IDs close to the info hash in question. These chosen nodes are asked for peers for the info hash, and can either return the desired peers, or, due to the routing table's structure, contact information of nodes with even closer IDs. Eventually, nodes will be able to return contact information from peers participating in this torrent. An example of an request for peers is given in figure 2.1, where the querying node finds peer information on the second iteration.

Announcing When a peer downloads a torrent, it should announce this fact to multiple other nodes with ID close to the info hash, in order to be included in the distributed hash table. This is also shown in figure 2.1. Again, the problem of IP address spoofing exists here, allowing malicious hosts to register third parties for a torrent. This is why a token system is used. On every request for peers, the response includes the SHA-1 hash of both the querying node's IP address and a secret value as chosen by the queried node. 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 way the remote peer can include the DHT node in its routing table.

2.3 Magnet Link

The concept of a *magnet link* described in [BEP 9] is used to create a uniform resource identifier (URI) for torrents of minimal size, in comparison to the metainfo file format. Its only mandatory component is the info hash, which is 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. It can look like this:

```
1 magnet:?xt=urn:btih:57e159a58261c8d2f42a64980d2b808e3901fcf6&dn=debian-8.0.0-
amd64-DVD-1.iso&tr=http%3A%2F%2Fbttracker.debian.org%3A6969%2Fannounce
```

The emerging problem of a magnet link substituting a torrent file, is the loss of the info dictionary's content. Since a tracker URL is optional, the metadata must be obtained from other peers. This is possible thanks to the *Extension for Peers to Send Metadata Files* as described below in section 2.3.2.

2.3.1 Extension Protocol

To expand the functionality of the BitTorrent protocol, the *Extension Protocol* [BEP 10] was defined. It introduces an additional message type as previously indicated in table 2.6. The generic `extended` message 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. These handshakes include a bencoded dictionary with information about the actually used extensions and assign IDs for every extension dynamically. Additional values as defined by the used extensions are also included. This setup allows for an arbitrary number of extensions with dynamic IDs, without the need for a global registry of extensions. Further extended messages have three parts:

1. The type ID 20, indicating that it is an extended message,
2. The extension ID, indicating the corresponding extension for this message as defined in the handshake, and
3. the payload of the message.

2.3.2 Extension for Peers to Send Metadata Files

The *Extension for Peers to Send Metadata Files* [BEP 9] is the first and only extension to make use of the Extension Protocol used in this work. It enables peers to exchange metadata about torrents in the form of the info dictionary of a torrent. 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.

In order for a peer to ask another peer for metadata about a torrent, first the address of another peer is needed. It can be obtained using the DHT network. After establishing a peer connection with both the normal and extended handshakes, every piece must be requested separately from

this peer. These *request* messages 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 combined and checked against the info hash.

2.4 BitTorrent and German Law

In the following sections, a short overview is given about the legal situation in Germany regarding topics relevant to this thesis.

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 [12] 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 [7], so to comply with the law by all means, the IP addresses of peers were not collected during this work.

3 Implementation

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. The source code will be published [22] on GitHub under the GNU GPLv3 license at a later date.

3.1 Dependencies

There are a few external dependencies, which are all free and open-source software. The Python module *BencodePy* by Weast [23] provides an encoder and decoder for bencoded messages and values. The *Object Relational Mapper* of *SQLAlchemy* [18] is used to store evaluation results in the *SQLite* database format [19]. The *GeoIP2 API* [21] is used to perform IP geolocation lookups in the *GeoLite2 City Database* [25]. 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 Jimenez [20] is used.

3.2 Architecture

The torrent files and magnet links which should be analyzed, as well as all configuration parameters have to be provided at start, since 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 input and output directories are defined in `config.input_path` and `config.output_path` respectively.

The BitTorrent Download Analyzer is structured in a main script, an application module, five helper modules and an utility module. An overview about core tasks is given in the sequence diagrams of figures 3.1 and 3.2. To give an overview, first the roles of the modules will be explained. A detailed look on the functionality is given in section 3.3.

main.py This is the main script to be invoked when performing the analysis. It starts the worker threads of the analyzer module. Three main analysis components can be enabled separately with command line options. It uses the following syntax.

```
1 ./main.py <options>
```

`--active` Actively contact and evaluate peers using the number of threads specified in `config.peer_evaluation_threads`.

`--passive` Listen on the port specified in `config.bittorrent_listen_port` for incoming peer connections and evaluate these peers.

`--dht` Integrate and control an already running *pymdht* DHT node using Telnet. The UDP port on which the node is running and the localhost Telnet port where *pymdht* can be controlled are set in `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. When using this flag, it is advised to decrease `config.peer_evaluation_threads` to reduce the amount of log output.

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

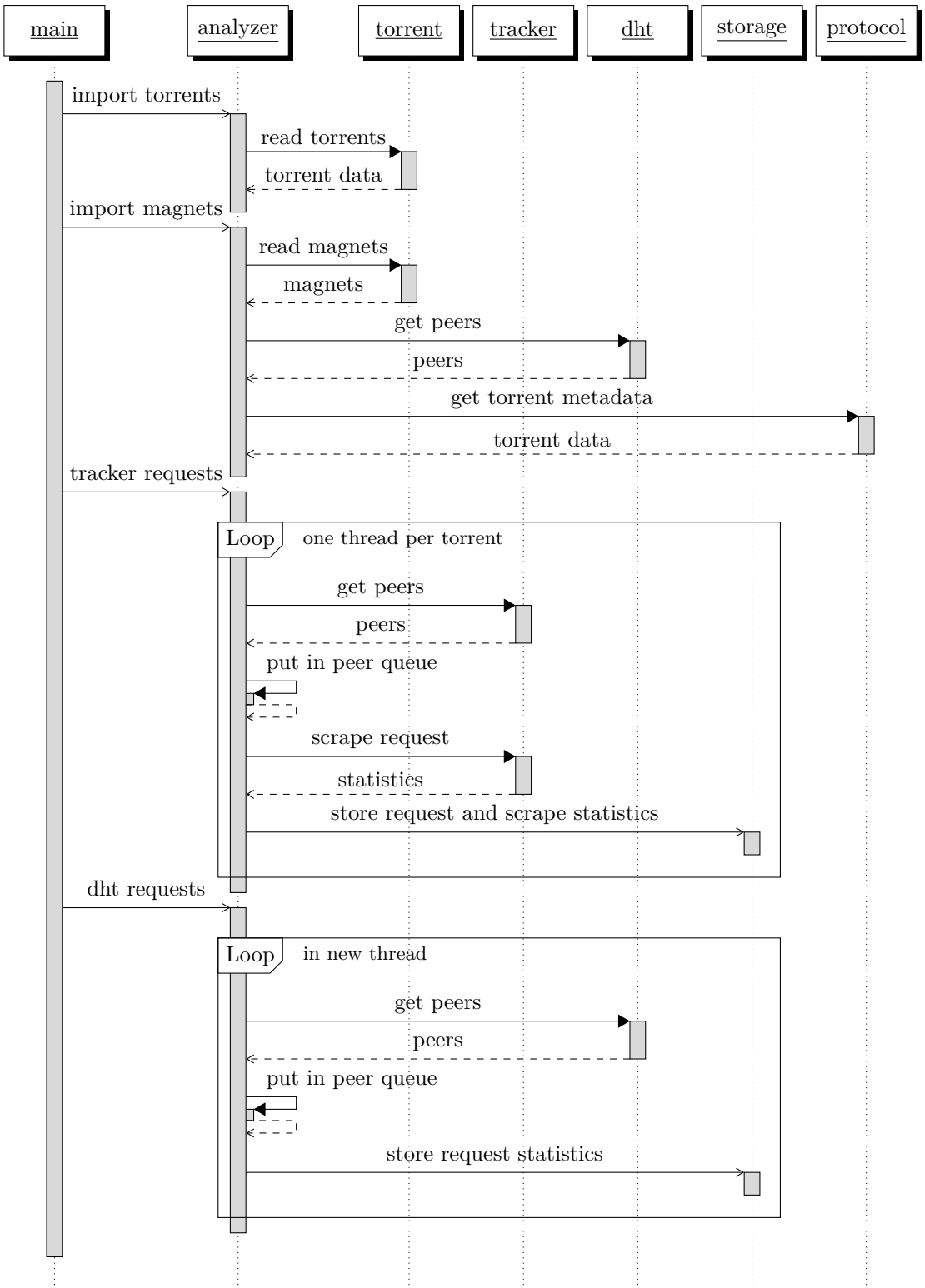

Figure 3.1: First part of the sequence diagram of the BitTorrent Download Analyzer. All loops are actually running in parallel. One additional thread, which is not shown here, writes monitoring statistics such as system load, memory consumption and queue lengths to the database. Only new unique peers are placed in the *peer queue*, others are discarded.

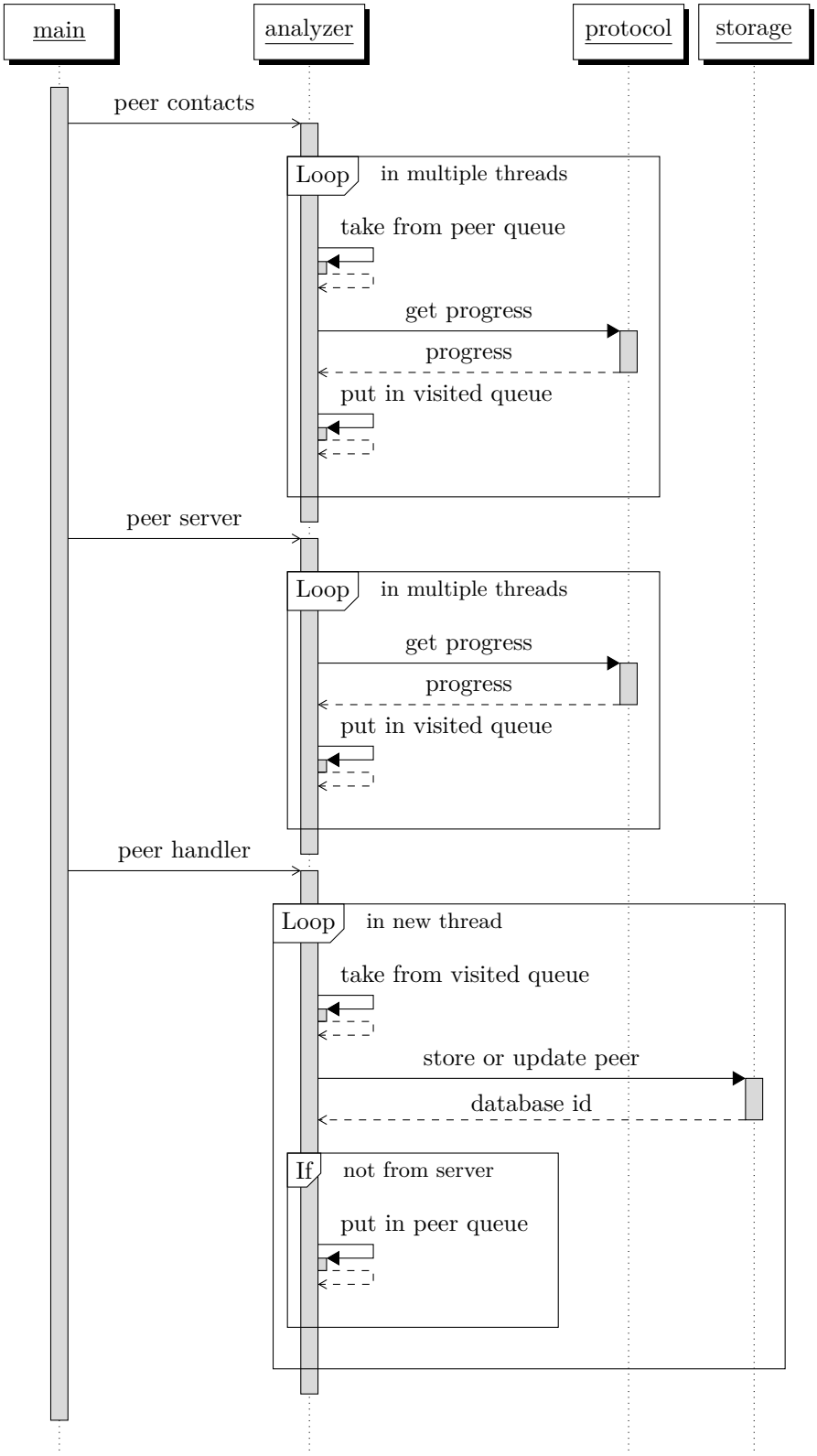

Figure 3.2: Second part of the sequence diagram of the BitTorrent Download Analyzer. Modules are identical to part one in figure 3.1, the diagram was split due to the lack of space on one page. Information about a peer's download progress is put in the *visited queue* together with the corresponding peer.

analyzer.py This is an application module which contains the main program logic. It defines all initialization and analysis routines. An coordinated shutdown procedure between all threads is realized with multi-threading lock mechanisms.

torrent.py The torrent module defines parsers for torrent files and magnet links. It is used initially after the analysis is started.

tracker.py This module provides communication methods with tracker serves. It is able to perform announce and scrape requests using standard HTTP Tracker Protocol as well as the UDP Tracker Protocol. Announce statistics from scrape responses and peer lists from announce responses are parsed and returned.

dht.py The *pymdht* DHT node has to be started separately and is controlled by this module using a localhost Telnet connection. As *pymdt* is written in Python 2, it could not be integrated directly. The support for control via Telnet was already a feature of *pymdt*, but was slightly altered to only accept local connections from the same machine. The UDP node port and the Telnet control port must be given as arguments when starting *pymdt* and should reflect the values as written in `config.dht_node_port` and `config.dht_control_port`. The command used in this work to start a *pymdht* node was:

```
./run_pymdht_node.py --port=17000 --telnet-port=17001
```

protocol.py All communication with other peers is handled in the protocol module. It defines methods for sending and receiving bytes using standard socket programming. Methods for sending and receiving handshakes and messages of the Peer Wire Protocol built upon these. Support for the Extension Protocol, as well as the Extension for Peers to Send Metadata Files are also established. Finally, routines are defined for receiving all messages from a peer, evaluating the download progress from these, and actually requesting metadata from a peer with the mentioned extension.

storage.py The possibility to write results to a SQL database is given with this module. It defines table schemata for a torrent table, a request table, a peer table and a statistics table and provides an API for these tables. Additional IP based geolocation with tools provided by MaxMind as described in section 3.1 is also performed.

util.py The util file provides utility methods and classes which are used in all other modules. They are not necessarily specific for the application in this software. Notable content is a custom queue implementation, which rejects duplicate items even if they were deleted meanwhile, gives feedback whether or not an item was rejected, sorts its content and is thread-safe.

3.3 Functionality

The main task is the counting of confirmed downloads. A download is considered as confirmed, when a peer crosses a threshold of downloaded pieces as defined in `config.torrent_complete_threshold`. To count a peer, there must be contact with him at least twice because of this – with the amount of downloaded pieces once below and once equal or above the threshold. This can be achieved with the BitTorrent Download Analyzer as described in the sequence diagram of figures 3.1 and 3.2. Its operation will now be explained in detail. For all connections to tracker servers and peers, the same peer ID was used.

3.3.1 Import Torrents

Beforehand, the torrents to be analyzed are imported. The torrent files are read from the input directory and are detected by their `.torrent` file name extension. Following the specification of metainfo files from [BEP 3], the announce URL, the info hash and the count and size of pieces are extracted. An additional parameter which may be present in the metainfo file is `announce-list`, which adds support for torrents with multiple trackers. All of them are stored and later used for collecting peers.

Magnet links which should be imported have to be placed in a file as defined by `config.magnet_file`. There must be one magnet link 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, to receive the info dictionary using the Extension Protocol and the Extension for Peers to Send Metadata Files. Peers are contacted sequentially until this process succeeds. The metadata and source of each imported torrent is stored in the *torrent* table of the database for later reference.

3.3.2 Requesting Peers

For every registered torrent, an own thread performs announce and scrape requests in an interval defined in `config.tracker_request_interval`. While announce requests to collect peers are sent to every tracker of a torrent, scrape requests are only performed on the main tracker from the `announce` key of the metainfo file. From the scrape request, the three given values of seeders, leechers and completed downloads are recorded. To monitor the operation of the torrent threads, any errors are counted and written in an extra file with the suffix `_tracker-error.txt` in the output directory.

Equally, peers are collected from the DHT network, although only in a single thread. As specified in `config.dht_request_interval`, requests for peers are sent for each info hash periodically. Peers received from both sources are placed in a *peer queue*, where they are actively contacted and evaluated later. Only new unique peers are placed in the *peer queue*. A peer is defined by its IP address and port number, since a peer may change its peer ID at any time. There is only one *peer queue* for all torrents, but peers are assigned to a certain torrent and may actually be in the queue for multiple torrents. The number of unique and doubly received peers as well as data from scrape requests is stored in the database's *request* table to get statistics about different peer sources.

3.3.3 Contact Peers

In `config.peer_revisit_delay` the time between active visits of a peer from the *peer queue* is specified. To observe this delay, every peer in the *peer queue* has a timestamp assigned to it and must not be contacted prior to this time. When a peer is placed in the queue first, the timestamp is set to zero. After a peer was visited and is put back in the *peer queue*, it is set to the according time in the future. The *peer queue* is sorted ascending according to the timestamp, so peers with small timestamps are evaluated first. The peers of this queue are contacted in parallel in a number of threads as set in `config.peer_evaluation_threads`. If one thread gets a peer with a timestamp in the future despite the ascending timestamps, it sleeps until the attached time is reached. For threads to be able to react to new unique peers from tracker or DHT requests, the sleep duration is capped in `config.evaluator_reaction`. When this limit is reached, the peer is put back in queue and the next one will be chosen.

When a peer is chosen for evaluation, a TCP connection is established and the download progress evaluation initiated. The same download progress evaluation is performed on peers who connect to `config.bittorrent_listen_port`, where a TCP server is listening for connections. Incoming peers who send a handshake with an unknown info hash are ignored. Also, an incoming peer from an IP address which was successfully actively contacted before is ignored, to not count this peer twice. All errors from failed peer evaluations, incoming or outgoing, are counted and noted in an extra file with the suffix `_peer-error.txt` in the output directory to monitor overall success rates.

```
CREATE TABLE peer (
    id INTEGER NOT NULL,
    host VARCHAR,
    client VARCHAR,
    continent VARCHAR,
    country VARCHAR,
    latitude FLOAT,
    longitude FLOAT,
    first_pieces INTEGER,
    last_pieces INTEGER,
    first_seen INTEGER,
    last_seen INTEGER,
    max_speed FLOAT,
    visits INTEGER,
    source VARCHAR(8),
    torrent INTEGER,
    PRIMARY KEY (id),
    CHECK (source IN ('tracker', 'incoming', 'dht'))
);
```

Listing 3.1: Schema of the peer table in SQL.

3.3.4 Extracting the 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 messages are received until there is no message for a certain amount of time as defined by `config.network_timeout`. The timeout is restarted after every message. To prevent potentially infinite sessions, there is a limit on the number of messages named `config.receive_message_max`.

The peer contact data, the list of messages and the peer's ID are placed in a tuple and then put in in a separate *visited queue* in order to be processed by another thread. This is the task of the *peer handler* thread from figure 3.2. Using the received messages, it compiles a combined bitfield from these messages and counts the present data pieces. The number of total torrent pieces from in the torrent metadata helps validating the result. The download progress of the peers is then stored in the database, which takes care itself to recognize if a peer was already contacted earlier and can be updated instead. Finally, the peer is returned to the *peer queue*. If `config.torrent_complete_threshold` is reached, the peer will be discarded and not further contacted. Incoming peers, which were placed in the *visited queue* by the server, are not written to the *peer queue*, as their BitTorrent port is not known.

3.3.5 Database

When a peer is first stored to the database, an IP address based geolocation lookup is performed using the *GeoLite2 City Database* mentioned in section 3.1. Two-letter codes of the country and continent, and latitude and longitude coordinates are determined. Then a reverse DNS lookup is performed on the peer's IP address to get information about used hosting providers or ISPs. Also the client identifying part of the peer ID is recorded, as defined in [BEP 20]. Regarding relevant data to count confirmed downloads, 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. The download speed of peers is calculated between each two consecutive contacts, while

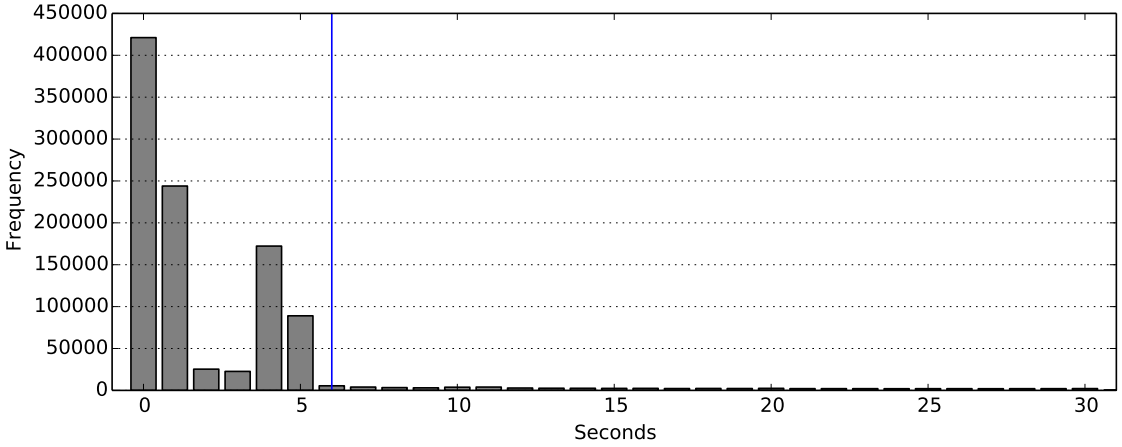

Figure 3.3: Duration of receiving one peer message

only the maximum of speeds measured between two contacts is saved. The SQL table schema of the peer database if given in listing 3.1.

The database ID as returned by SQLite is used for later reference to the peer by the analyzer module, to enable updates of the entry without storing any IP address or peer ID data. When the program is terminated, there is no way to connect a peer in the database to the original person.

3.4 Justification of Configuration Values

network_timeout The timeout for network operations is six seconds. It is used when asking the BitTorrent tracker or the DHT node for peers and when asking other peers for metadata. These cases are uncritical since they were observed to be much faster. 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 3.3. 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 %. The recording of this value can be enabled with `config.rec_dur_analysis`.

receive_message_max When collecting all messages from a peer, a maximum of *128 messages are considered. This limit is in place, to prohibit infinite peer sessions. Only *0 % of evaluations reached this maximum on the main analysis pass, others stopped because of the timeout or another error.

peer_revisit_delay Peers were contacted for their download progress every five minutes. This delay should be as small as possible, to also consider peers who stop seeding a torrent immediately after they finished downloading it. Five minutes caused no problem with system resources.

tracker_request_interval and dht_request_interval Both request intervals were set to five minutes. As discussed later in section 4.2, the rates of doubly received peers are very high. This shows five minutes is enough to cover the majority of peers.

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

torrent_complete_threshold This is actually just the threshold, where the analysis module stops to keep track of a peer and will no longer contact him. The threshold used for confirmed download calculation can be chosen separately when compiling download numbers from the database. However, the threshold for download counting must be equal or lower than this threshold, because peers above were not tracked in the first place. The value of 98 % was chosen, because peer's progress appeared often to stop between 98 % and 100 %. So ±2 % seems to be the precision of the measured download progress. This suggests, that a threshold above 98 % causes traffic without further benefits.

peer_evaluation_threads This is the number of threads who are contacting peers from the *peer queue*, which contains addresses from tracker servers and the DHT network. In this analysis 1,024 threads were used. Every peer evaluation needs at least six seconds, because the message receiving has to time out. In order to process peers quickly, many threads are needed. Since threads are mostly idle waiting for a timeout, this number of threads is not a problem for system load.

3.5 Restrictions

To get most complete results, the BitTorrent Download Analyzer aims to collect data from as many peers as possible. Unfortunately, there are still a few restrictions regarding peer sources in comparison to real BitTorrent clients. This is due to the high implementation and development effort which would be necessary to integrate each of those. For each restriction it will be discussed shortly, why it does not harm this work at large.

- No support for IPv6 addresses from HTTP announce requests [BEP 7] or the separate IPv6 DHT network [BEP 32]. A measurement study [15, sec. 4.2.] from 2011 for IPv6 traffic in BitTorrent networks found between 1 % and 4 % of peers using IPv6.
- No support for the Micro Transport Protocol (μTP or uTP) [BEP 29], which enables UDP communication between peers. Peers can support this in addition to the Peer Wire Protocol for better bandwidth management. Users can configure clients to only use uTP, but this may harm their download speeds.
- No support for peer exchange (PeX). This enables peers to exchange peer lists among each other with simple messages. Currently there are different implementations in use, an official BEP does not exist, yet.
- No support for the Tracker exchange extension [BEP 28], which enables peers to exchange announce URLs of tracker servers. This is useful for magnet links without any trackers or torrent files with missing trackers. All torrents investigated in this work have at least *0 trackers, see section 4.2.
- No support for the Azureus DHT network, which is a separate DHT network from *Mainline DHT* described in [BEP 5]. Peer numbers in the Azureus DHT network are significantly lower [3, table 5].

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 August 20 9:52 to August 22 10:04 UTC, 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.

[Thread workload and system load were normal as described earlier*.]

TODO 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.

[mention the great R scripts!]

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. [24] 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 [27] 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:

- Set A: The 8 most popular torrents below 1 GB
- Set B: The 8 most popular torrents above 1 GB
- Set C: The 8 most popular torrents above 10 GB

Popular torrents were chosen regarding leecher numbers, despite the website sorting by the sum of seeders and leechers. Set B only contains only torrent smaller than 10 GB since larger torrents are less pupular in comparision. The 27 chosen torrents are listed in table 4.2. These three sets will be evaluated separately where appropriate.

4.2 Getting Addresses of Peers

Peers from all sources were considered, namely from the tracker server, the DHT network and incoming connections. The total number collected across all torrents during this analysis in regards

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 [24] global traffic ranking. Only sites with a rank below 5,000 are listed. Data is accurate as of July 16, 2015.

Set	ID	Torrent name	Size	Leechers	Seeders
A	1	torrent_name	3.0 GB	42	23
	4	torrent_name	3.0 GB	42	23
	8	torrent_name	3.0 GB	42	23
	11	torrent_name	3.0 GB	42	23
	13	torrent_name	3.0 GB	42	23
B	3	torrent_name	3.0 GB	42	23
	6	torrent_name	3.0 GB	42	23
	9	torrent_name	3.0 GB	42	23
	10	torrent_name	3.0 GB	42	23
	14	torrent_name	3.0 GB	42	23
C	5	torrent_name	3.0 GB	42	23
	7	torrent_name	3.0 GB	42	23
	12	torrent_name	3.0 GB	42	23
	15	torrent_name	3.0 GB	42	23
	16	torrent_name	3.0 GB	42	23

Table 4.2: * List of the 20 most popular torrents according to meta-search engine Torrentz [27]. 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.

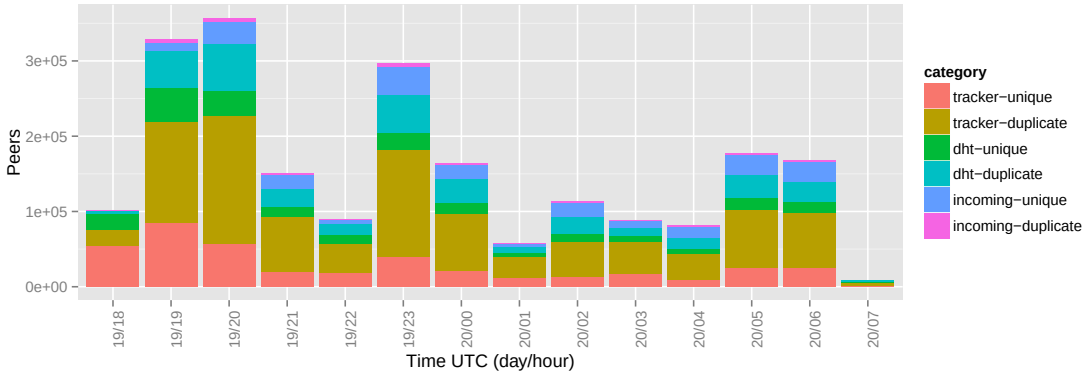

Figure 4.1: Development of received peer **addresses** per source for all torrents, on average. New peers from the tracker server and the DHT network are requested every five minutes. UDP trackers returned always 200 addresses per request, trackers with HTTP access 50. Torrents are usually registered with few trackers. DHT request were observed to return between 100 and 1,300 peers each. The values *incoming-unique* and *incoming-duplicate* only include peers after their download progress was extracted successfully. The analysis lasted from August 20 9:52 to August 22 10:04 UTC, 2015.

to their source is shown in table 4.3. Since the *incoming peers* data point in this table 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.

The progression of received peer addresses during the analysis period is shown in figure 4.1. Again, this is a summary for all torrents. From the incoming peers, only the ones represented by *incoming-duplicate* are candidates to be counted as confirmed downloads, since the were evaluated at least twice. The ratio between new and duplicate received peer address information is slightly higher in the first hours*.

[In general the amount of duplicate received peers is above xx%*, so presumably many of the addresses known to trackers and the DHT network are collected in this process.]

[view tracker response rates]

4.3 Counting Confirmed Downloads

Table 4.4 shows the confirmed download numbers measured during this analysis per torrent set. For comparison, numbers reported by the tracker in scrape requests are given. * [Compare sets and unique peers]

[Note all things in the plots, even the obvious] Scrape is higher than confirmed.

[downloaded (event=complete) vs complete (seeders), [16] uses seeders]

[try different thresholds]

Torrent	Confirmed	Reported	Confirmed/Reported	Unique Peers
Set A	5	2	3 %	
Set B	10	2	3 %	
Set C	1	2	3 %	

Table 4.4: Confirmed **downloads** per torrent set. In comparison, *reported* downloads are responses from scrape requests to the tracker during the analysis. *Unique peers* are all peers successfully evaluated for the torrents in question, although not crossing the confirmed download threshold.

[ignore first and last hour]
[reported per hour, confirmed per hour]
The history of download numbers during the analysis period is shown in figure 4.2.
[Check single torrents for outliers]
[correlation with scrape bachelor and ieee bt]

4.3.1 Problems of this Method

The Evaluation of a peer's download progress was described in section 3.3.4. This process can fail for various reasons. The superficial problems which occurred in this analysis are listed in table 4.5. The table shows every attempt of connecting to a peer and performing the evaluation procedure. Outgoing evaluation attempts were usually* successful. The mainly* appearing error is a timed out peer, which indicates outdated or wrong peer address information. However, the value of failed evaluations on a second or later contact is very low. Regarding incoming peers, most evaluations* failed due to an unknown info hash. This is probably because of other torrents evaluated beforehand from the same IP address and port. The response rate between peers obtained via trackers and peers received through the DHT network was not examined.

The search for more fundamental explanations for errors merely speculation. Peers may blacklist the IP of the analysis program over time, since the don't receive any data at all. Also, peers could only support the UDP based Micro Transport Protocol (μTP), which is not supported by this analysis tool. When peers have no free upload capacity, they may block all incoming connections. When the evaluation on a peer failed once, no second try will be made. If this only happens due to a temporary network problem and the peer is in fact downloading, this will not be noticed. Presumably the biggest problem are BitTorrent clients without an exposed port, in other words peers behind a NAT. Since this is the case with a computer behind a standard home router, this setup is very common. Several of these machines will connect to our program after receiving our IP address from a tracker or the DHT network*, but there is no guarantee* [see actual incoming dpulicate stats] to capture all of them. Finally, the BitTorrent client could be terminated immediately after the download finished, leaving no time for us to register any further progress.

[No BitTrorrent Protocol support for getting download progress info]

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

[maybe move this whole section to Conclusion]

4.4 Further Analysis of Peers

4.4.1 Download Speed

filter peers for visits
filter countries for peers
TODO: Graph of speed per country with standard derivation indicators, n>=10 per country
todo: Map of Distribution of download speeds
Map: Distribution of downloads (TODO)

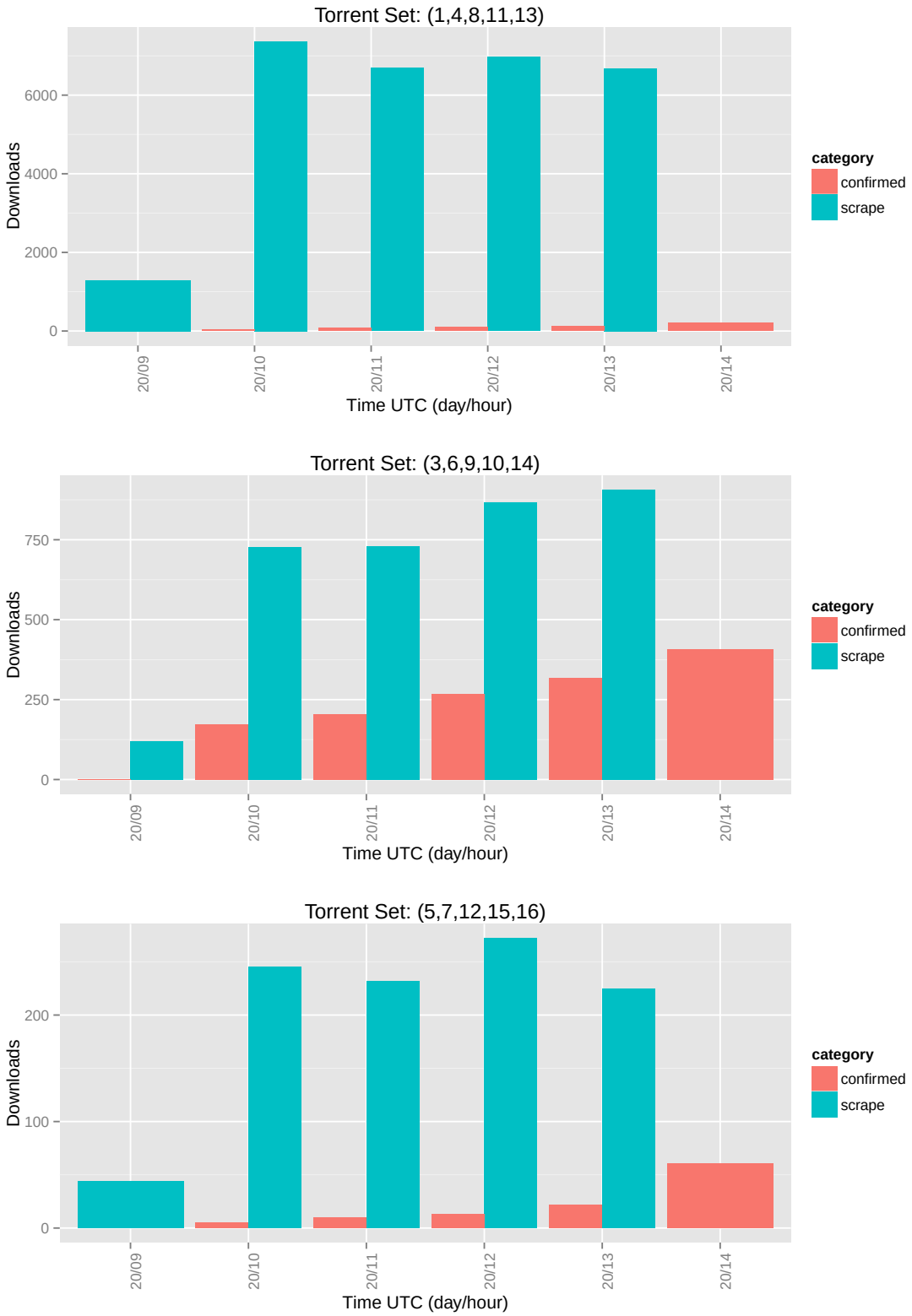

Figure 4.2: Development of *confirmed* and server reported **downloads** per torrent set. From top to bottom: Set A (< 1 GB), set B (1 GB to 10 GB) and set C (> 10 GB). Server reported numbers are gathered using *scrape* requests. Analysis duration was from August 20 9:52 to August 22 10:04 UTC, 2015.

Event	Error	Quantity	Share per Event
First contact	timed out	182209	63.8%
	Socket connection broken	47690	16.7%
	Connection refused	47028	16.5%
	No route to host	4938	1.7%
	Connection reset by peer	3330	1.2%
	Network is unreachable	213	0.1%
	Peer speaks unknown protocol	5	0.0%
	Total	285413	
Later contact	Socket connection broken	3847	55.9%
	timed out	2244	32.6%
	No route to host	369	5.4%
	Connection refused	249	3.6%
	Connection reset by peer	176	2.6%
	Total	6885	
Incoming peer	Unknown info hash	399331	68.3%
	Peer speaks unknown protocol	124794	21.4%
	timed out	57021	9.8%
	Connection reset by peer	1821	0.3%
	Socket connection broken	1427	0.2%
	Total	584394	

Table 4.5: Outcome of peer evaluations during the analysis process with error and success rates. *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.

4.4.2 BitTorrent Clients

TODO: Table of frequency of BT clients

4.4.3 Peer's Host Names

Due to the time it takes to perform a reverse DNS lookup, the lookup was disabled in the main analysis run. The data of host names analyzed in this section was collected in an earlier pass[1].

TODO: Table: Examine hostnames by ISP or seedbox provicers

[1] Files: ...

5 Conclusion and Future Work

The number of successful revisits of peers is often too low, as shown in figure 5.1. This could indicate, that peers only participate very shortly in a torrent. However, this could also suggest a problem in the process of recontacting peers.

...

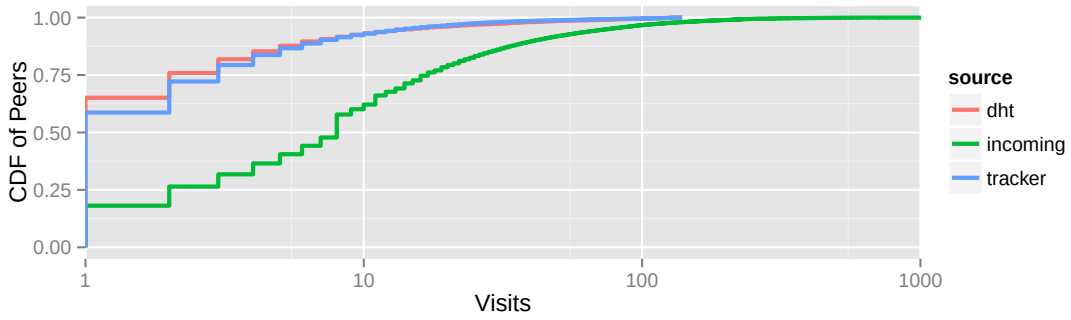

Figure 5.1: CDF of successful peer visits per source. Only peers of which the download progress was evaluated successfully at least once are included in this figure. A lower graph is better as it indicates more visits per peer. Most of the peers who were contacted actively could only visited once. Incoming peers try to get pieces usually a few times, while many of them stop trying after the first or the eighth attempt. Some even never stop trying.

Bibliography

Literature

[1] A. Bhakuni, P. Sharma, and R. Kaushal. "Free-rider detection and punishment in BitTorrent based P2P networks". In: *Advance Computing Conference (IACC), 2014 IEEE International*. Feb. 2014, pp. 155–159. DOI: 10.1109/IAdCC.2014.6779311.

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

[3] 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.

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

[5] 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.

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

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

[8] 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.

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

[10] Michel Meulpolder et al. "Public and private BitTorrent communities: a measurement study." In: *IPTPS*. 2010, p. 10.

[11] 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.

[12] 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.

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

[14] 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.

[15] Eric Vyncke and Martin Defeche. "Measuring IPv6 Traffic in BitTorrent Networks". In: (2012).

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

Software

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

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

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

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

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

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

Online

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

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

[26] Theory.org. *BitTorrentSpecification*. 2006. URL: https://wiki.theory.org/BitTorrentSpecification.

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

Standards

[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.

[BEP 0] David Harrison. *Index of BitTorrent Enhancement Proposals*. BEP 0. 2008. URL: http://www.bittorrent.org/beps/bep_0000.html.

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

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

[BEP 7] Greg Hazel and Arvid Norberg. *IPv6 Tracker Extension*. BEP 7. 2008. URL: http://www.bittorrent.org/beps/bep_0007.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.

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

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

[BEP 20] David Harrison. *Peer ID Conventions*. BEP 20. 2008. URL: http://www.bittorrent.org/beps/bep_0020.html.

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

[BEP 28] Arvid Norberg. *Tracker exchange extension*. BEP 28. 2008. URL: http://www.bittorrent.org/beps/bep_0028.html.

[BEP 29] Arvid Norberg. *uTorrent transport protocol*. BEP 29. 2009. URL: http://www.bittorrent.org/beps/bep_0029.html.

[BEP 32] Juliusz Chroboczek. *BitTorrent DHT Extensions for IPv6*. BEP 32. 2009. URL: `http://www.bittorrent.org/beps/bep_0032.html`.

[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`.

[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`.