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

Bachelor Thesis

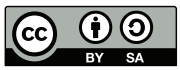
Analysis of BitTorrent Trackers and Peers

Counting Confirmed Downloads in BitTorrent

Stefan Schindler

Erlangen, September 2, 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, September 2, 2015

Stefan Schindler

Abstract

BitTorrent is the most used technology for file sharing to date and discussed to damage the creative industry. However, due to its distributed structure the extent of downloaded copies can only be estimated. The common method is to collect all IP addresses of a torrent swarm by issuing scrape and announce requests to tracker servers. After testing their reachability, this number is used as an estimation of downloads, but falsely includes peers who did not finish the download. This thesis will extend this method by contacting each peer continuously and learning the download progress through the BitTorrent protocol. A tool was written to do this after collecting addresses from trackers and the DHT network, and accepting incoming connections. A confirmed download was registered when a peer crossed the threshold of 98 %. For small and large torrents respectively, between 9 % and 55 % of the downloads reported by the main trackers in scrape responses could be confirmed over 34 hours and 19 torrents. For less than 2 % of unique peer addresses a download was confirmed. Often subsequent progress evaluations of peers failed, which lowers this numbers significantly. Further adjustments to the used implementation are necessary to obtain better results.

Zusammenfassung

BitTorrent ist die meistgenutzte Filesharing-Technologie, wobei die Schhädlichkeit für die Kreativindustrie diskutiert wird. Jedoch kann man den Umfang an heruntergeladenen Kopien nur abschätzen. Üblicherweise werden dafür alle IP-Adressen eines Torrent-Schwarms mithilfe von Scrape- und Announce-Anfragen an die Tracker-Server gesammelt. Nachdem deren Erreichbarkeit getestet wurde, wird diese Zahl als Abschätzung der Downloads genutzt, wobei Peers, die den Download nicht abgeschlossen haben, fälschlicherweise mitgezählt werden. Diese Arbeit wird diese Methode erweitern, indem jeder Peer wiederhohlt kontaktiert und der Download-Fortschritt mit Hilfe des BitTorrent-Protokolls extraihert wird. Es wurde ein Programm geschrieben, das dies tut, nachdem es Addressen von Trackern und dem DHT-Netzwerk sammelt, und eingehende Verbindungen annimmt. Ein bestätigter Download wurde registriert, wenn ein Peer den Grenzwert von 98 % überschreitet. Über 34 Stunden und 19 Torrents konnten für kleine beziehungsweise große Torrents zwischen 9 % und 55 % der von Haupttrackern in Scrape-Antworten gemeldeten Downloads bestätigt werden. Für weniger als 2 % aller eindeutigen Addressen wurde ein Download bestätigt. Wiederhohlte Auswertungen von Peers sind oft fehlgeschlagen, was diese Zahlnen wesentlich reduziert. Die verwendete Implementierung muss weiter angepast werden, um bessere Ergebnisse zu erhalten.

Contents

List of Figures

List of Tables

1 Introduction

Bram Cohen published [2] his idea for a decentralized file sharing protocol called *BitTorrent* in 2003. 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 [6] 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" [14, 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 [11] or even positive [15] 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 [17] from the University of Ballarat, Australia, finds numbers between 90 % and 97 %.

1.1 Motivation

When assessing popularity of torrents by 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 managed by a tracker. When successful, servers answer with the torrent-identifying info hashes together with the number of current downloaders, current uploaders and completed 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. The download number acquired in a scrape request may be to their best knowledge, but can also be flawed. And the peer addresses collectable from tracker servers may be out of date or peers actually never finished the download after joining the torrent swarm.

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. A threshold is used to exclude peers who are seeding a torrent and do not complete the download during the period of analysis. This way, a number downloads per hour can be calculated accurately. This method has flaws which will be discussed later, but it gives a fix lower bound for download numbers.

		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* [6]. *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

To observe the law in every way, it is 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.

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 code base used for the analysis routines and peer communication was written from scratch. The need for communicating with peers without downloading any torrent payload disqualifies other related projects like *libtorrent*.

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 [7], 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, hostname 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 [17], 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 [4] 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. [18], 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 [8] and punishment [7, 1] of free-riding peers, who do not upload any data after downloading from other peers, or the special implications of private BitTorrent communities [12], which promise higher download speeds by enforcing upload to download ratios.

1.4 Results

Depending on torrent size, only between 9 % and 55 % of tracker reported downloads could be confirmed (see table 4.4). The value is even worse when compared to unique peer addresses: For less than 2 % of them a download could be confirmed. These values are too low and are not usable for absolute estimations for this reason. The fundamental method of counting confirmed download works, but succeeds too rarely in its current implementation.

A big problem is the high drop-off rate of 85 % after the first active visit (see figure 4.8). The fact that a minimum of two successful evaluations is mandatory to decide whether a peer crossed a threshold, makes this value important and problematic. In contrast, the drop-off rate for incoming peers is only 22 % after the first visit.

1.5 Outline

Chapter 2 explains the background of the BitTorrent technology as it is applied in the written analysis tool. The BitTorrent protocol will be described according to the official BitTorrent Enhancement Proposals (BEPs). This is important to understand how measured values were

generated and should be handled. The concept of a DHT network will be introduced and the utilization within BitTorrent demonstrated. Additionally, a short overview about German copyright laws regarding the download of illegal content is given.

In chapter 3, the data collection tool written for this thesis, called *BitTorrent Download Analyzer*, is introduced. First its dependencies from other software are documented. The modular architecture and the command line interface will be explained. The main part is the description of functionality and implemented features. Some important configuration values for the analysis are explained and also justified. Finally, the restrictions of the tool in comparison to real BitTorrent applications are stated.

Chapter 4 will present a data set gathered with the analysis tool and evaluate the success of measuring confirmed downloads through various diagrams for data visualization. Beforehand, the selection of the 19 analyzed torrents is documented. The recognized problems of the implementation are highlighted. The last section provides additional analysis of the collected data, namely measured download speeds per country and a ranking of observed Internet service providers.

The final chapter 5 will draw conclusions about the performance of the BitTorrent Download Analyzer. Possible solutions for detected problems are suggested.

1.6 Acknowledgments

I want to thank Philipp Klein for discussing methods and implementation as well as managing the virtual machines 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.

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.

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 2.2. 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:

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 **??**.
`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.

```
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
```

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

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.

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 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.

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.

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 communicates by sending bencoded messages over UDP. The DHT used in BitTorrent follows the Kademlia design as described by Maymounkov and Mazières [10] 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

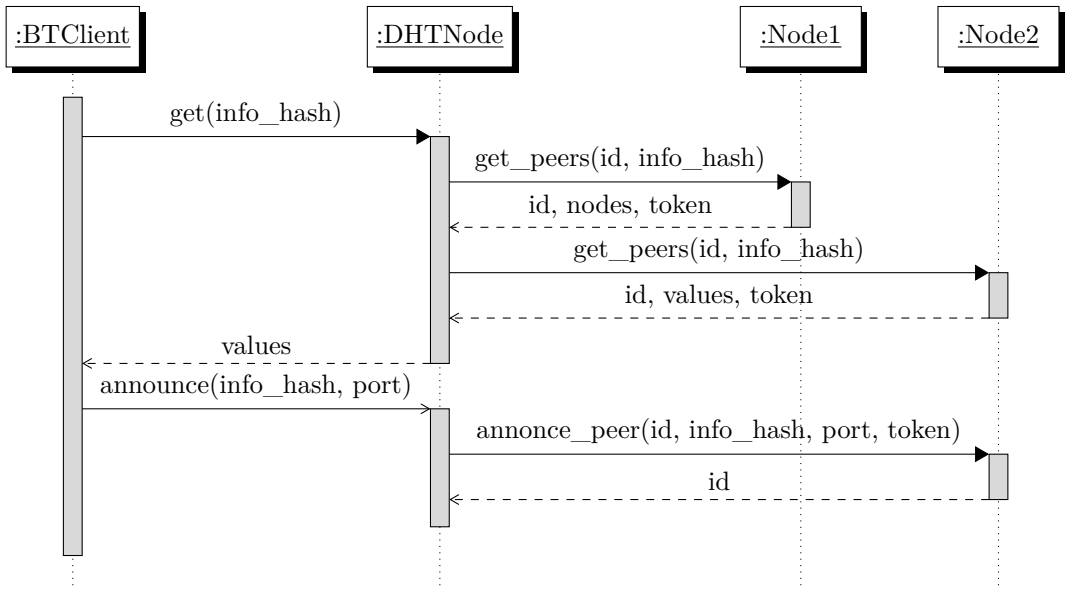


Figure 2.1: A sequence diagram of a request for peers in the DHT network as described in [BEP 5]. *BTClient* asks its own *DHTNode* for peers for a specific info hash. The DHT node issues a request to remote *Node1*, wich does not know any peers and answers with information about other nodes, including *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.

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 [13] 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 [9], 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. In this chapter, this tool will be discussed in detail. Section 3.1 will inform about its dependencies, section 3.2 explains structure and usage. The implemented procedures will be described in 3.3 and 3.4, while the latter one will focus on configuration values. In section 3.5 some restriction of this implementation will be discussed.

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* [19] is used to store evaluation results in the *SQLite* database format [20]. 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 [22] 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.

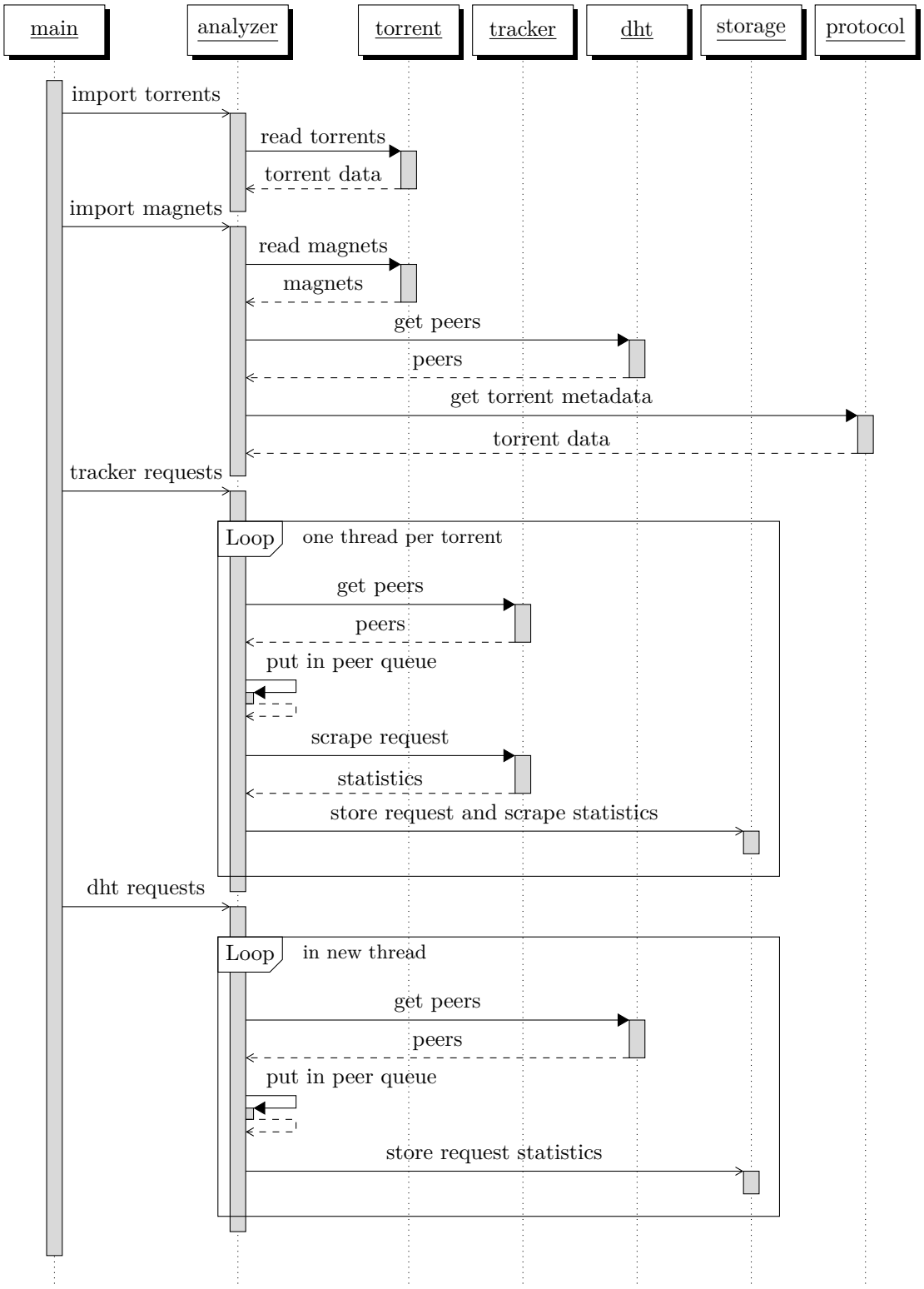


Figure 3.1: First part of the sequence diagram of the BitTorrent Download Analyzer. All loops are actually running in parallel. One additional thread, which is not shown here, writes monitoring statistics such as system load, memory consumption and queue lengths to the database. Only new unique peers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

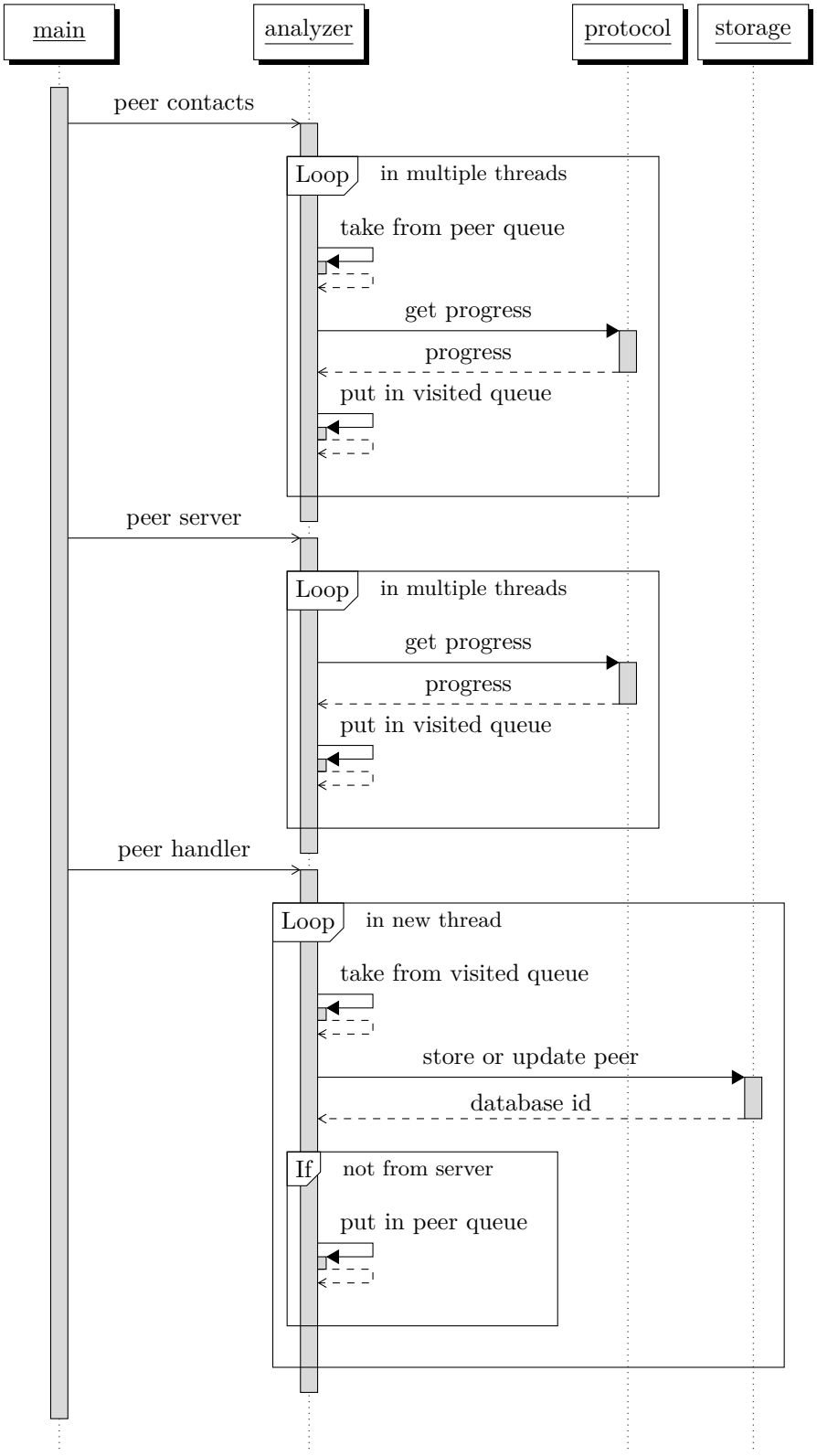


Figure 3.2: Second part of the sequence diagram of the BitTorrent Download Analyzer. Modules are identical to part one in figure 3.1, the diagram was split due to the lack of space on one page. Information about a peer's download progress is put in the *visited queue* together with the corresponding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

`--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.

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:

```
1 ./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.

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 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

```
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.

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. 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 256 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.

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,

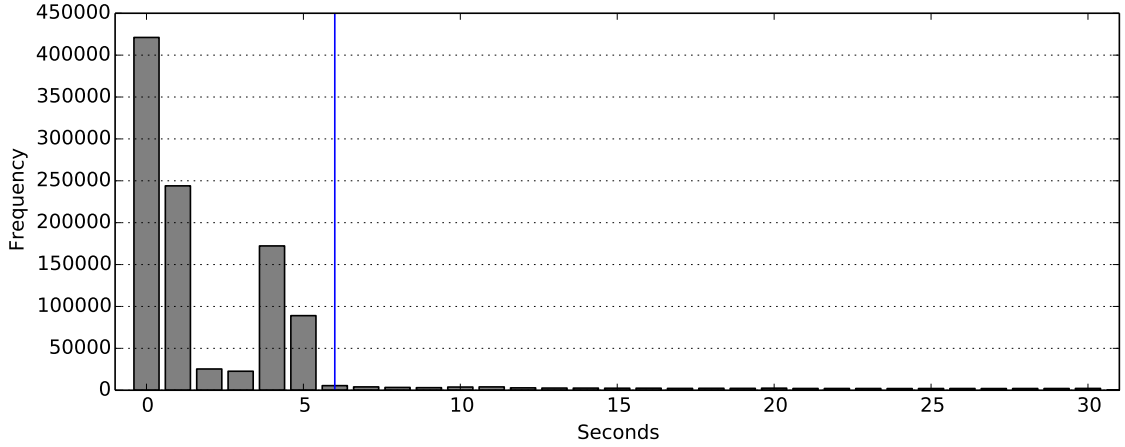


Figure 3.3: Duration of receiving one peer message. Data is taken from files `2015-08-14_17-46-44_faui1-246.sqlite` and `2015-08-14_17-46-44_faui1-246_timeout.txt`.

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 [16, 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 [4, table 5].

4 Evaluation

Data collection with the BitTorrent Download Analyzer tool was performed simultaneously on two identical virtual machines running Ubuntu 14.04 LTS with 2.0 GB of RAM, 3.4 GHz dual-core processors and own IPv4 addresses without NAT. The analysis was performed for 34 hours from August 30 18:40 to September 1 5:10 UTC, 2015. Each machine was running an own *pymdht* DHT node, which did not crash before or during the analysis. The BitTorrent Download Analyzer gave no problematic error messages in the logfile. The used BitTorrent port was changed right before the evaluation started, in order to decrease system load from peers with an unrelated info hash from earlier evaluations.

In order to support the validity of the test results, some statistics were logged to the database with a sample rate of five minutes. These are shown in figure 4.1. Graphs of a low thread workload and the short length of the peer queue make clear, that the number of 1,024 worker threads was enough. The number of used evaluator threads did actually not surpass 500. The length of the visited queue is almost always zero, which proves that the database was not a bottleneck. A maximum of 300 MB RAM is used, which is way below the available 2 GB. The Unix load-average is below the 2 mark for two processor cores.

After the development phase of the BitTorrent Download Analyzer was finished, the *SQLAlchemy* database toolkit started to report errors about a locked database. This occurred despite using the multi-threading mechanisms provided by SQLAlchemy. The problem surfaced very late and could not be resolved because of this. The errors were only reported for the request table and the statistic table for monitoring values and were few in comparison to the number of successfully stored database rows. The used workaround for this problem was to apply failed SQL statements, which were all recorded in the logfile, to the database afterwards. Finally, the databases of the two virtual machines were combined. The commands used for creating the final database are given in listing 4.1.

4.1 Choosing Torrents

Due to the distributed nature of BitTorrent, there is no complete list of all active torrents. External data about popularity of torrents is needed, even if there is no guarantee of completeness. To get

```
1 scp torrent-vm1:bittorrent-analyzer/btda/output/2015-08-30_20-34-06_faui1-246* .
2 scp torrent-vm2:bittorrent-analyzer/btda/output/2015-08-30_20-41-36_faui1-246* .
3 ./sql_from_log.py 2015-08-30_20-34-06_faui1-246.log
4 ./sql_from_log.py 2015-08-30_20-41-36_faui1-246.log
5 cat 2015-08-30_20-34-06_faui1-246.log.sql | sqlite3 2015-08-30_20-34-06_faui1
  -246.sqlite
6 cat 2015-08-30_20-41-36_faui1-246.log.sql | sqlite3 2015-08-30_20-41-36_faui1
  -246.sqlite
7 cat combine.sql | sqlite3 | sqlite3 2015-08-30_20-combined.sqlite
```

Listing 4.1: Steps to create the final database. The `combine.sql` scripts produces `INSERT` statments for all table rows of both databases and handles torrent IDs appropriately. Filename timestamps are in CEST, which is UTC+02:00. `2015-08-30_20-combined.sqlite` holds the data used for the evaluation.

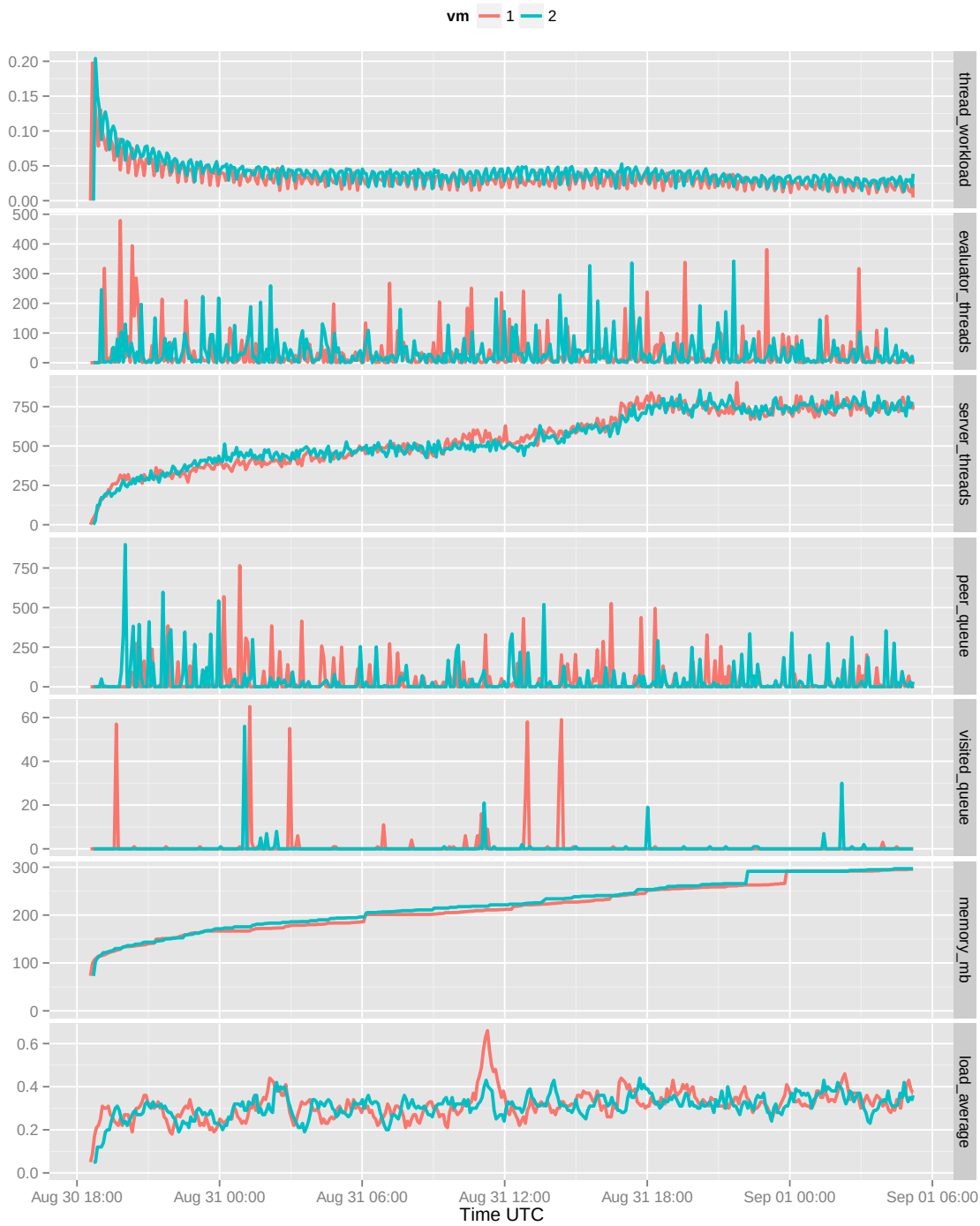


Figure 4.1: System parameters and queue lengths for monitoring. The *thread workload* is a time based value between 0 and 1, stating the average uptime of the peer contact threads, not the server threads. A thread is only considered as inactive when waiting on an empty *peer queue* or waiting for the revisiting timestamp of a peer, see section 3.3.3. The *evaluator threads* graph gives the same value as an absolute number while using the same constraints. *Server threads* is the number of active server threads, which is not limited. *Peer queue* and *visited queue* are the lengths of these queues. *Memory MB* specifies RAM usage, *load average* is the standard Unix processor load parameter.

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.

an overview about torrent directory sites and determine the most popular ones, 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 these 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 sorting and filter options by peer count, torrent age and size. Torrents of various sizes between 1.5 GB and 65 GB were chosen, because the torrent size has high influence on further results. For further reference, three size groups are defined. These three sets will be evaluated separately where appropriate.

- Set A: Between 1 GB and 5 GB
- Set B: Between 5 GB and 20 GB
- Set C: Between 30 GB and 70 GB

In order to observe many downloads, torrents with high amount of leechers were selected. Initially 20 torrents were chosen, but one tracker the torrent with ID 6 sent a bencoded value, which was not standard conform as described in section 2.1. This crashed the tracker request thread for this torrent, which is why this torrent is excluded from the evaluation. The 19 remaining torrents are listed in table 4.2. A metainfo file was available for all torrents.

4.2 Getting Addresses of Peers

Peers from all supported sources were considered, namely from the tracker server, the DHT network and incoming connections. The total number collected addresses across all torrents during this analysis in regards to their source is shown in table 4.3. The overall high rates of duplicate peers reveal, that the majority of peers collectable through these sources was received. The *incoming peers* data point in this table only includes peers, when their download progress was successfully

Set	ID	Name	Pieces	Size	Trackers
Set A	17	[www.CpasBien.pw] Avengers.Age...	5,619	1.47 GB	10
	8	Magic.Mike.XXL.2015.HDRip.XViD.A...	1,426	1.50 GB	10
	12	San.Andreas.2015.HDRip.XviD.AC3-...	5,880	1.54 GB	10
	4	Minions.2015.720p.HDRip.X264.AAC...	3,980	2.09 GB	10
	13	San Andreas 2015 720p WEB-DL x26...	610	2.56 GB	10
	15	Mission.Impossible.5-Rogue.Natio...	11,798	3.09 GB	10
	3	Straight.Outta.Compton.2015.1080...	4,108	4.31 GB	7
	5	Avengers.Age.of.Ultron.2015.720p...	4,369	4.58 GB	7
Set B	11	Avengers Age of Ultron 2015 1080...	1,327	5.57 GB	10
	9	F3_GOTY.iso	1,848	7.75 GB	5
	19	Game.Of.Thrones.S05.Season.5.COM...	2,270	9.52 GB	5
	18	3DMGAME-One.Piece.Pirate.Warrior...	2,657	11.14 GB	5
	16	DRP_15.8_Full.iso	10,711	11.23 GB	5
	2	Narcos S01 720p WEBRip x264-TAST...	1,826	15.32 GB	10
Set C	1	Battlefield Hardline by xatab	3,599	30.19 GB	7
	20	Mortal.Kombat.X.Proper-RELOADED	3,998	33.54 GB	5
	10	The X-Files S01-S09 WEBRip x264-...	12,502	52.44 GB	10
	14	Batman.Arkham.Knight-CPY	6,430	53.94 GB	10
	7	Dexter Season 1, 2, 3, 4, 5, 6, ...	7,774	65.21 GB	10

Table 4.2: List of 19 popular torrents according to meta-search engine Torrentz [27]. The selection was made 30 minutes before the analysis started. The *ID* was chosen by the SQLite database. Some movies are shared in multiple active torrents, but have different sizes or were uploaded by different users.

Source	Total	Unique	New
Tracker server	5,614,412	691,248	12.31 %
DHT network	3,546,070	856,367	24.15 %
Incoming peers	6,538,653	258,939	3.96 %
Total	15,699,135	1,806,554	11.51 %

Table 4.3: Total and unique received peer addresses per source for all torrents. The ratio of new *tracker* and *DHT* peers depends mainly on the request interval for new peers. Numbers of *incoming peers* only include those where the download progress was evaluated successfully. Numbers of *total* received peers may overlap between sources in this table. Numbers of *unique* peers do not overlap between sources, each peer is only counted as unique one time. This is why the distribution between sources may not represent uniqueness per source in general, but only in this data set. Numbers were acquired from the *request* database table.

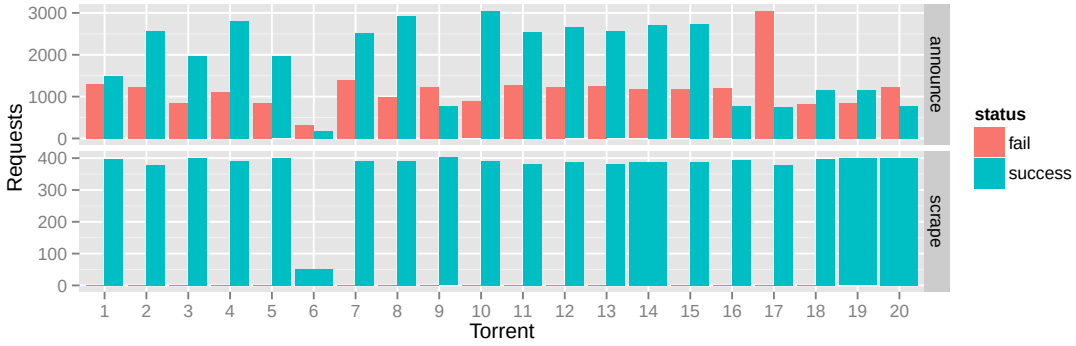


Figure 4.2: Responsiveness of tracker servers per torrent. The thread to perform announce requests for torrent 6 crashed after four hours, which is why this torrent is excluded from the analysis. As apparent from table 4.2, metainfo files for torrents 9, 16 and 18 to 20 had only five announce URLs embedded, explaining the reduced request numbers. Also torrents 1, 3 and 5 only had seven trackers. Data is taken from the file `2015-08-30_20-combined_tracker-error.txt`, which is written by the analyzer tool.

evaluated. Hence, the ratio of 3.96 % new incoming peers implies that 96.04 % of all incoming contacts are second or later visits, although peers did not receive any torrent data. Only these later visits allow to determine the crossing of a threshold.

The responsiveness of the trackers per torrent is examined in figure 4.2. As expected, torrents with less tracker announce URLs embedded in the torrent file show less attempted announce requests. From torrents with ten trackers usually about two third of them respond. On torrents with only five announce URLs about half of them responded. Torrent 17 stands out with a high number of failed requests. Reviewing the logfile reveals, that only two out of ten trackers responded. Other trackers may have been unreliable or actually removed the torrent due to a DMCA takedown notice by the copyright owner.

The number of new collected unique peer addresses is highest in the first few hours, as shown in the the summary for all torrents in figure 4.3. Nearly all peers in the first 30 minutes are new. The value declines and stays steady at 50,000 unique peers per hour for about 18 hours. In the last 15 hours of the analysis, this value declines slowly. The spike at the beginning consists of all peers in the torrent swarm at this point in time. The constant part are new peers who enter the torrent during the analysis period. The decline at the end may be users loosing interest of the torrent's contents, because other torrents raise in popularity. Another valid explanation is a 24-hour cycle, which implies that unique peer numbers would raise again later. The duration of 34 hours is too short to tell the difference.

4.3 Counting Confirmed Downloads

4.3.1 Trying Different Thresholds

Peers were tracked up to a download progress of 98 %. Since the number of pieces on the first and last contact for each peer was recorded, the number of confirmed downloads can be extracted by filtering for peers with a progress on the first visit below and on the last visit above the threshold. Different threshold values were tested in figure 4.4. The maximum value of calculated downloads is nearly identical between 75 % and 98 %, with declining numbers for lower thresholds. The value of 98 % will be used for the further evaluation, in order to only register complete downloads.

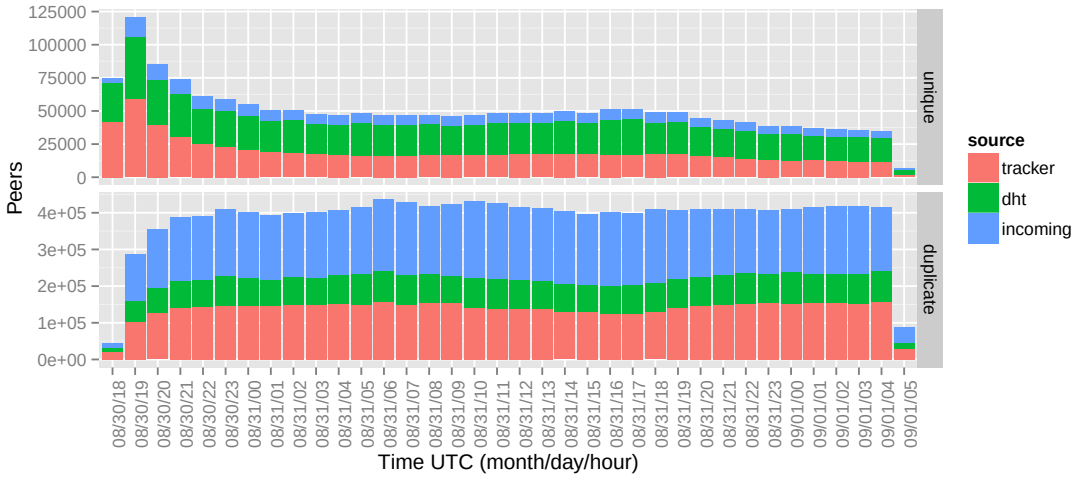


Figure 4.3: Development of received peer addresses per source for all torrents. UDP trackers returned always 200 addresses per request, trackers with HTTP access 50. Torrents are usually registered with a few trackers. DHT request were observed to return between 100 and 1,300 peers each. Values have the same characteristics as described in table 4.3: *Incoming* peers only include successfully evaluated ones, *unique* peers do not overlap between sources, *duplicate* peers may overlap between sources. The unique peers diagram uses a smaller scale for peers than the duplicate ones to increase its readability.

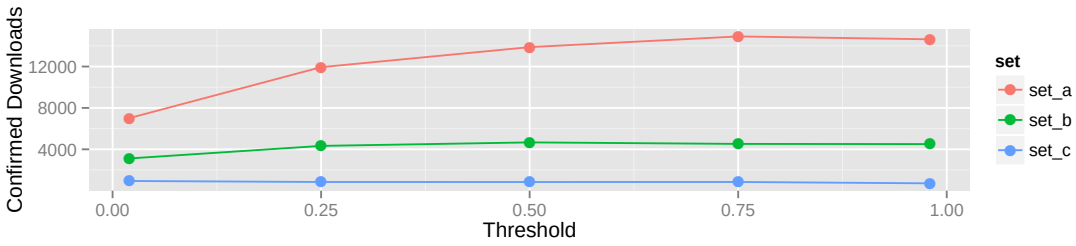


Figure 4.4: Confirmed downloads using different thresholds per torrent set. Peers with pieces on the first visit below and on the last visit above the threshold are counted as confirmed.

Set	Confirmed	Reported	Unique Peers	C./R.	C./U.	R./U.
Set A	14,632	150,629	1,264,472	9.71 %	1.16 %	11.91 %
Set B	4,505	17,872	383,420	25.21 %	1.17 %	4.66 %
Set C	713	1,299	158,662	54.89 %	0.45 %	0.82 %
Total	19,850	169,800	1,806,554	11.69 %	1.10 %	9.40 %

Table 4.4: Confirmed downloads per torrent set. Two comparison values are given: *Reported* downloads are the differences from the number of completed downloads in a tracker's scrape response. *Unique peers* is the number of unique peer addresses collected in the process.

4.3.2 Summary

To judge the success of counting confirmed downloads with the planned method, two comparison values will be considered. First, the number of completed downloads as reported by the tracker server. This is a value available form the tracker through scrape requests, which were issued regularly together with announce requests. For each hour, the difference of this value was calculated. As stated in section 3.3.2, scrape requests were only issued to the main tracker embedded in the torrent file, since the numbers may overlap between trackers otherwise. The second comparison value is the number of collected unique peer addresses as introduced in table 4.3. This may include outdated information of peers who have disconnected meanwhile or never finished the download. Both values can not provide the true number of happened downloads, but allow to assess the performance of the tool. Table 4.4 shows the comparison values per torrent set. Following conclusions are possible:

1. The number of total unique peers is way above confirmed and reported downloads, roughly by factor 100 and 10 respectively.
2. Judging by tracker reported numbers, confirmed downloads are more accurate with larger torrents.
3. For large torrents, there are less reported downloads per unique peer than for small ones.

4.3.3 Comparison with Scrape Requests

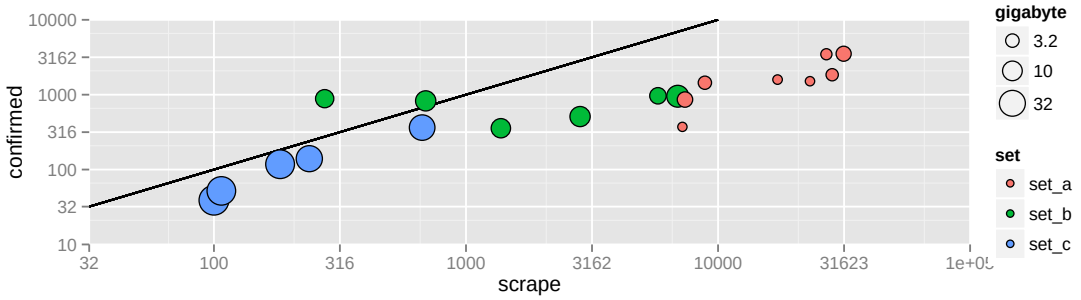
When viewing data for each torrent in figure 4.5 (a), a clear positive relation between confirmed downloads and tracker reported numbers is visible, although confirmed numbers are mostly lower. It is also obvious, that large torrents are less downloaded than smaller ones in general. Part (b) plots the success rates of measuring confirmed downloads, using the tracker's scrape numbers as a reference. The chart shows, that for torrents in set A with less than 5 GB, the numbers of confirmed downloads are only between 4 % and 17 %. Results between 5 GB and 20 GB in set B are all above 13 %. On two torrents, the measured numbers exceed the tracker reported ones by factor 1.2 and 3.2. All five large torrents of set C are between 39 % and 64 % percent. These graphs shows three things:

1. Confirmed download numbers produced by the BitTorrent Download Analyzer are heavily correlated with tracker reported complete downloads.
2. The method of counting confirmed downloads is heavily dependent on the torrent size. Best results above 40 % are achieved with torrents above 30 GB. Presumably, the reason for this is the increased likelihood of recording a peer's download progress, when the downloads takes more time. This is given with large torrents.
3. Scrape requests are not always accurate and can indicate too small download numbers. This is understandable, since the tracker relies on every peer to report a finished download. A BitTorrent client may disconnect without sending an *completed* event to the server. A client may not have used the main tracker of the torrent in the first place, but another one or none at all.

4.3.4 Comparison with Unique Peers

In figure 4.6 similar data is shown, but using the number of collected unique peer addresses as a reference. When considering part (a), a correlation between the number of unique peers and the number of confirmed downloads is still recognizable, although unique peer numbers are way higher than confirmed downloads. This was already apparent from table 4.4. Again, success rates of counting confirmed downloads are given by torrent size in part (b). In this graph, the success rate of set C is noticeable low, which contradicts with the scrape-based success rate discussed above. However, the success rate based on unique peers is tiny for all torrents and varies between 0.1 % and 2.3 %. This is too small to draw conclusions. We can learn from this data:

(a) Confirmed downloads by reported downloads.



(b) Confirmed ratio regarding reported downloads by torrent size.

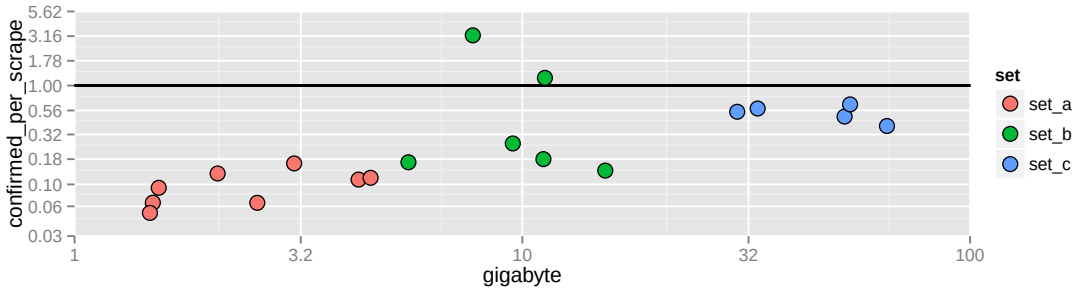
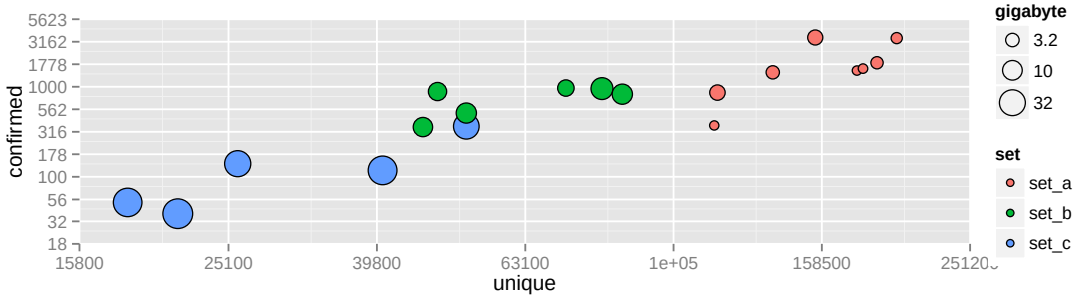


Figure 4.5: Relation between confirmed downloads and tracker reported download numbers from scrape requests per torrent, using a double logarithmic scale. Each point represents a torrent. The black lines mark the identity between confirmed and reported downloads.

(a) Confirmed downloads by unique peers.



(b) Confirmed ratio regarding unique peers by torrent size.

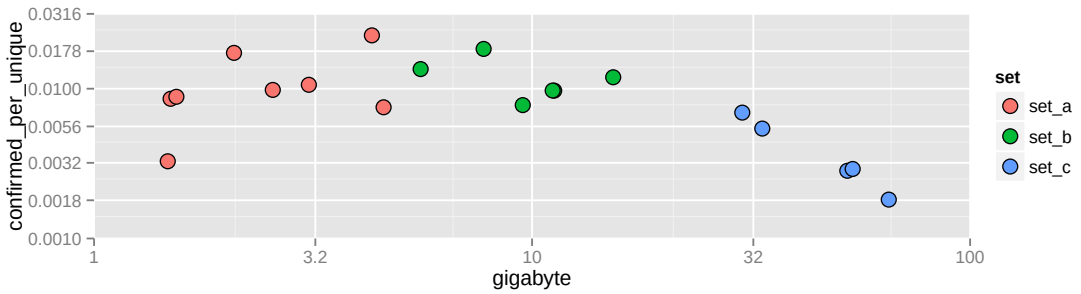


Figure 4.6: Relation between confirmed downloads and unique collected peer addresses per torrent, using a double logarithmic scale. Each point represents a torrent. The identity between confirmed downloads and unique peers is outside of the plot area.

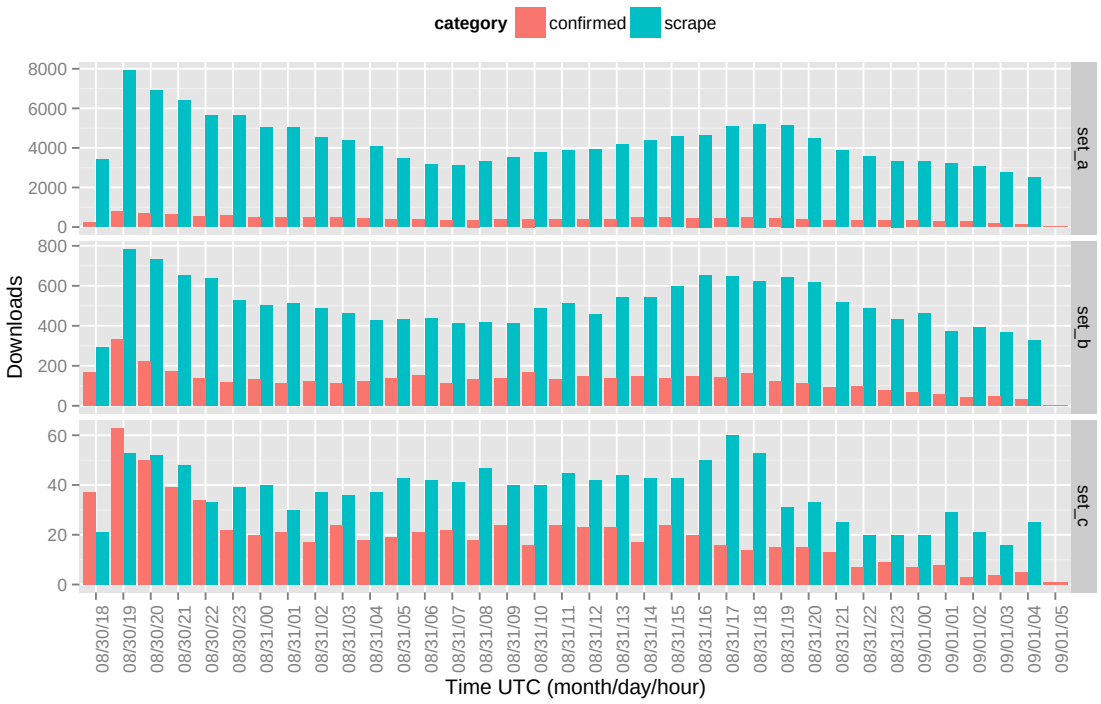


Figure 4.7: Development of confirmed and server reported downloads per hour and torrent set. The timestamp for a confirmed download is the time of first contact with the peer. Analysis duration was from August 30 18:40 to September 1 5:10 UTC, 2015.

1. Confirmed download numbers are also heavily correlated with the number of unique peers collected by the BitTorrent Download Analyzer.
2. The success rate of confirming downloads for unique peers is below 3 %. An important reason for this are seeding peers, which are part of the torrent swarm, but do not download any more. Besides this, outdated peer data reduces this value.
3. The smallest number of confirmed downloads per unique peer was determined for torrents above 30 GB. This matches with the comparison between tracker reported downloads and observed peer addresses in table 4.4.

4.3.5 Downloads per Hour

The history of download numbers over time in figure 4.7 once more illustrates the relation between measured download numbers and torrent size. Unfortunately in retrospect, the timestamp when a peer crossed the download threshold was not recorded, but only one from the first and the last contact. In this plot, the time of first contact is used, which explains the high numbers in the beginning and low values at the end. Likewise to the received peer addresses in figure 4.3, download numbers settle after a few hours.

4.4 Problems

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.

Event	Result	Frequency	Share per Event
First contact	Timed out	978,505	61.40 %
	Connection refused	242,616	15.22 %
	Socket connection broken	177,833	11.16 %
	Success	149,608	9.39 %
	No route to host	22,042	1.38 %
	Connection reset by peer	21,512	1.35 %
	Network is unreachable	1,446	0.09 %
	Peer speaks unknown protocol	24	0.00 %
	Total	1,593,586	
Later contact	Success	166,174	81.74 %
	Socket connection broken	26,559	13.06 %
	Timed out	7,203	3.54 %
	Connection reset by peer	1,725	0.85 %
	Connection refused	1,252	0.62 %
	No route to host	390	0.19 %
	Network is unreachable	5	0.00 %
	Total	203,308	
Incoming peer	Unknown info hash	7,187,788	37.26 %
	Already in outgoing	6,572,026	34.07 %
	Peer speaks unknown protocol	2,283,818	11.84 %
	Timed out	2,144,463	11.12 %
	Success	1,059,055	5.49 %
	Connection reset by peer	26,330	0.14 %
	Socket connection broken	16,266	0.08 %
	Broken pipe	2	0.00 %
	Total	19,289,748	

Table 4.5: Outcome of peer evaluations for different events. An *event* describes the point in the evaluation process. *First* and *later contact* are performed when actively contacting collected peer addresses. After an evaluation of a peer fails, the peer is not contacted another time. An *incoming peer* represents an incoming connection on the BitTorrent listening port. Error data is from the `2015-08-30_20-combined_peer-error.txt` file as written by the analyzer tool, success counts are from the *visits* column of the *peer* table.

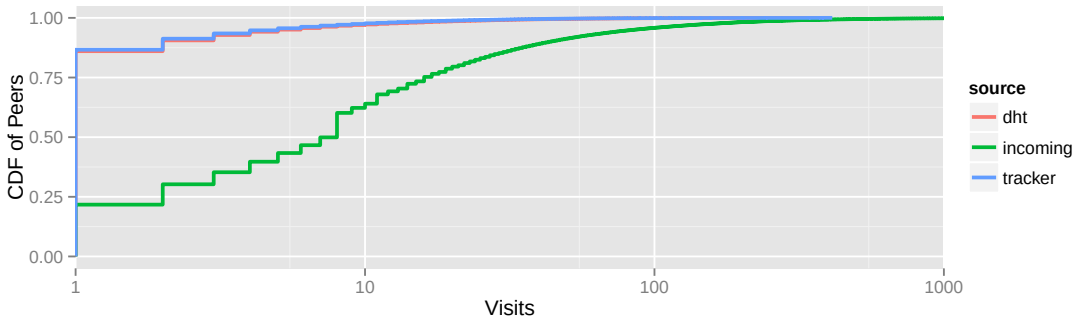


Figure 4.8: CDF of successful peer visits per source. Only peers of which the download progress was evaluated successfully at least once are included in this figure. A lower graph is better as it indicates more visits per peer. Most of the peers who were contacted actively could only be visited once. 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.

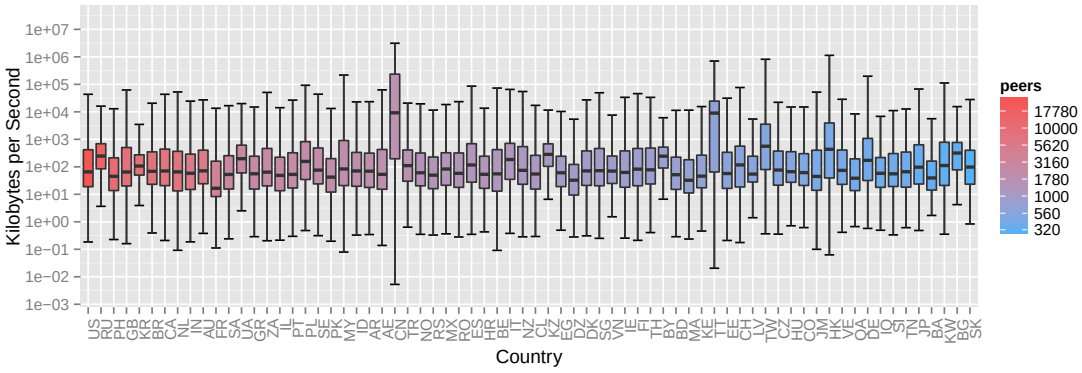


Figure 4.9: Download speeds of the 70 countries with most recorded unique peers, using a logarithmic scale for speeds and peer count. This graph uses a standard box plot with first and third quartiles as box delimiters and the median inside the box. Countries are sorted descending for the number of observed peers. 179 other country codes are not represented in this chart.

The table shows every attempt of connecting to a peer and performing the evaluation procedure. It shows, that the chance of successfully evaluating a peer's download progress on first contact is 9.4 %. The majority of connection attempts timed out. The number of 82 % succeeded later evaluation attempts seems to indicate a high success rate when contacting a peer a second time. However, one has to consider, that a peer is not contacted ever again after a failed attempt. This means every error on a later contact is a lost peer, successful evaluations were performed multiple times per peer. The effect is obvious when viewing figure 4.8: After one successful visit, the chance of a second successful visit is only about 15 %.

Regarding incoming connections, most evaluations failed due to an unknown info hash. Before the data of this evaluation was collected, the analysis tool was tested with different torrents on BitTorrent port 6,883. Before the analysis started, it was changed to 6,884 to reduce unrelated traffic. Peers obviously tried to connect to the next higher port number, successfully. Since these info hashes then were not analyzed, the evaluation failed.

The search for explanations of these errors is largely speculation. Peers may blacklist the IP or used peer ID of the analysis program, since they do not receive any data at all. When peers have no free upload capacity, they may block all incoming connections. If a peer has a temporary network connectivity problem and the evaluation fails because of this, no second try will be made.

4.5 Further Analysis of Peers

The following sections examine additional characteristics which can be extracted from the collected data set, but are not related to torrent download numbers.

4.5.1 Download Speed per Country

When the download progress of a peer changes between two visits, a speed value can be calculated. This has been done for all peers during the analysis, while only keeping the maximum speed measured between too visits per peer. Figure 4.9 shows the results sorted by the number of unique peers. The median speed is mostly about 100 kB/s, while the majority of measurements are lying between 10 kB/s 1 MB/s. Standing out are China and Trinidad and Tobago with a median speed of nearly 10 MB/s. Most peers come from the United States, followed by Russia, the Philippines, the United Kingdom and South Korea. Speed values are drawn in a map in figure 4.10.

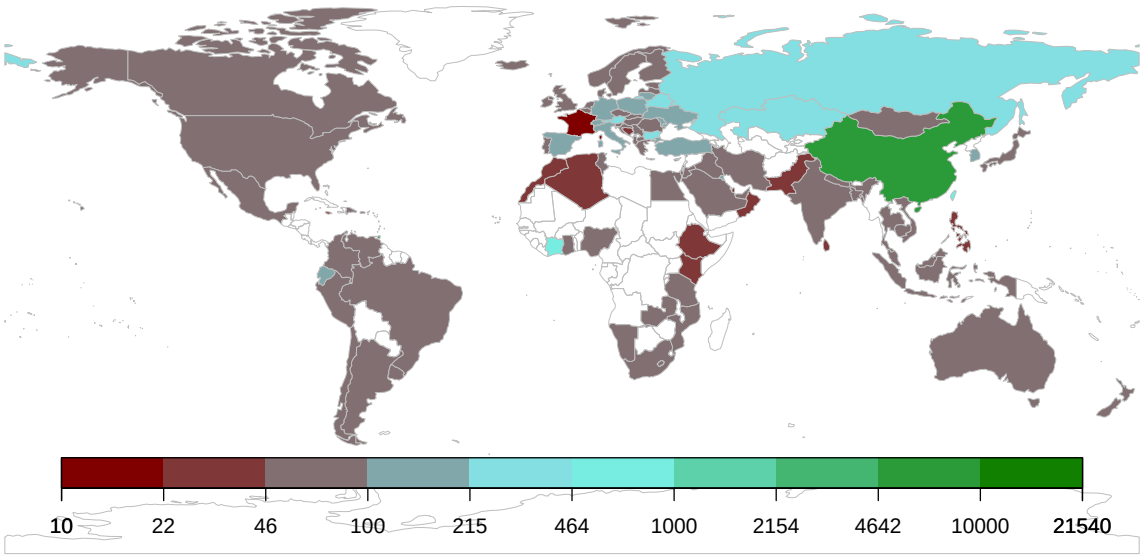


Figure 4.10: Download speed per country as a map. Only countries with a minimum of about 50 observed peers are shown. Speed is given in kB/s on an logarithmic color scale. 149 other country codes are not represented in this map.

4.5.2 Internet Service Providers

The data of hostnames was not extracted from the main analysis pass, but from an earlier one. Hostnames of peers were determined by performing a reverse DNS lookup on the IP address. This was performed by the storage module in the same thread which takes results from the *visited queue* and writes them to the database. Since this lookup took about 300 ms on average, this is too slow for the database thread and was disabled during the main analysis run. A solution to this problem, which is not implemented in the analysis tool, would be to perform the DNS lookups in a dedicated thread.

The data shown in figure 4.11 used the peer contact and server threads to perform the reverse DNS lookups. However, this was not a reasonable fix for the problem, because as most peer contacts are revisits, this produces too many DNS requests for the university's DNS server. The most common hostnames among peers represent the following organizations:

1. ISP Comcast form the United States
2. ISP Virgin Media from the United Kindom
3. ISP Philippine Long Distance Telephone from the Philippines
4. `com.br` is a second-level category for the Brazilian TLD
5. ISP Time Warner Cable from the United States, formerly known as Road Runner

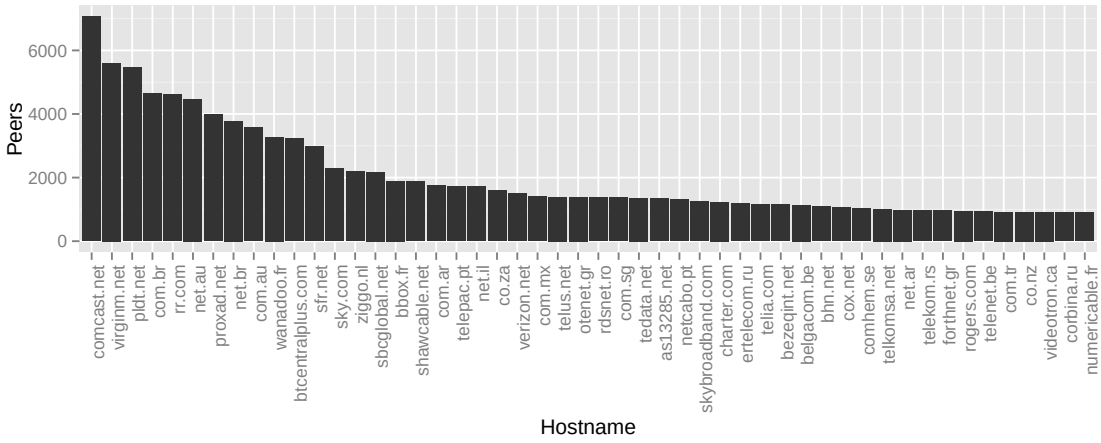


Figure 4.11: Top 50 hostnames of all unique observed peers. Only the TLD and SLD of hostnames were recorded. The majority of hostnames represent the used ISPs or seedbox providers, which are rentable servers optimized for usage with BitTorrent. Data is taken from a separate examination of twelve torrents which were chosen equivalently to the description of section 4.1. Data is taken from the database `2015-08-26_11-24-21_faui1-246.sqlite`.

5 Conclusion and Future Work

Existing problems of the BitTorrent Download Analyzer were discussed in section 4.4. When encountering an error while evaluating a peer, the peer is excluded from further contacts. The evaluation of a newly collected peer address succeeded in 9 % of cases. The success rate on the second visit was again only 15 %, while the delay between two visits is five minutes. These values have to be increased in order to derive conclusive results. The fact that the success rate of a second evaluation for an incoming peers is as high as 78 %, suggests the presence of a systematic problem.

A possible solution is to retry evaluation on failed peers after a longer delay. This could improve the chance of clients allowing a connection after receiving no data in the first session. Other helpful clues could be obtained by directly studying the behavior of common BitTorrent applications and make appropriate adjustments to the analyzer tool.

The requirement of two successful evaluations of a peer could be reduced to one, when running the analysis over a longer time. Uploading peers, who finished the download before the analysis was started would then be included in the result. However, after all seeding peers are registered, measured numbers should be comparable. This would reduce the complexity of the analysis method. The process can be simplified even further by only sending ping requests to collected peer addresses to test their reachability, but this would defeat the goal of measuring confirmed downloads.

To increase the accuracy of comparison values for the number of downloads, scrape requests from all trackers should be included. The overlapping of peers between these trackers has to be compensated by considering the subset of peers which is known to each tracker. This can be done by requesting all peers through normal announce requests from all trackers.

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] Bram Cohen. "Incentives build robustness in BitTorrent". In: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. 2003, pp. 68–72.

[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.

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

[6] Sandvine, Inc. *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.

[7] 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.

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

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

[10] 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.

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

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

[13] 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.

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

[15] 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.

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

[17] 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.

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

Software

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

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

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

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

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

Online

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

[25] MaxMind, Inc. *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.