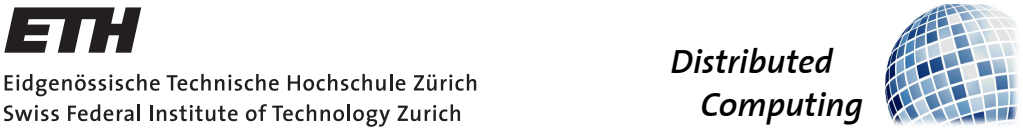

A Torrent Recommender based on DHT Crawling

Bachelor Thesis

Julian Fuchs

fuchsju@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Christian Decker
Prof. Dr. Roger Wattenhofer

April 22, 2015

Abstract

The DHT Mainline is a significant extension to the BitTorrent protocol. The DHT Mainline has several million users and is the largest DHT network. This thesis uses the DHT Mainline to generate a recommendation system for torrents.

A program was written crawling the entirety of the torrent search engine *kickass.to* gathering metadata about torrents. The DHT Mainline was then crawled to search for downloaders of torrents. The gathered data was clustered and filtered by a third program. A system using cosine based item similarity and collaborative filtering was implemented to generate recommendations based on the aggregated and clustered data. A website using the visitor's IP address and asking for favourite torrents displays the generated recommendations.

Contents

Chapter 1

Introduction

Torrents are a widely encountered phenomenon in today's world, contributing about 15% of Europe's Internet traffic in 2014 [1]. Movies, Games, TV Series, Books and Music are widely shared across the globe. This Bachelor Thesis examines torrents and their participants to generate a recommendation system.

Recommendations are based on vast amounts of data collected during the course of this thesis. The information was gained by crawling the torrent hosting service *kickass.to*[1], which is used by downloaders worldwide to gain access to the newest torrents. Over 6 million torrents got analysed by a program called the *torrent crawler*. By crawling the DHT network, a network used by torrent programs to find participants of torrents, a largely untapped resource was incorporated in this thesis. A program called the *DHT crawler* uses the DHT network to find the addresses of downloaders of torrents.

The gathered data is clustered and filtered by the *database merger* before being read by the *recommendation system*. Recommendations are generated on the assumption that users who download the same torrents have a similar taste. Collaborative filtering and cosine based item similarity are used to generate recommendations. The *website* displays recommendations for visitors. It inputs IP address and favourite torrents to the recommendation system. Having access to such vast data, this thesis illustrates the nature and composition of today's torrents.

1.1 Related Work

DHT crawling has previously been done for the Vuze DHT by Scott Wolchok [2]. Wolchok's approach shows that the transition in the file sharing arms race from centralized trackers to DHT based tracking makes torrent networks more resilient. The DHT Mainline has been analysed by Liang Wang and Jussi Kan-

[1] https://kickass.to/

gasharju [3], showing the size of the DHT Mainline.

A recommendation system for torrents was developed by Mittal et a.l. [4]. Their system based its recommendations on crawling various torrent search engines. The system developed in this thesis makes recommendations based on both crawling torrent websites and crawling the DHT Mainline. Another approach to recommendations for playlists in peer-to-peer networks was introduced by Pouwelse et a.l. [5], using the concept of taste buddies.

This thesis is based on the fundamental works about the BitTorrent protocol by Bram Cohen [6] and the introduction of the DHT protocol to BitTorrent by Andrew Loewenstern and Arvid Norbert [7].

Chapter 2

BitTorrent Basics

The phenomenon of peer-to-peer file sharing began in 1999 with a program called *Napster* [8]. The service, which allowed users to share their music libraries without paying anything, attracted 26 million users before being shut down 2001 due to a legal lawsuit.[1]

Napster was built as a centralized peer-to-peer network. Its weakness being a central server which was responsible for the exchange of files. This weakness was then also used to shut down Napster once and for all. The success of this very early peer-to-peer sharing network inspired others.

One who tackled the problem of centralized peer-to-peer networks was Bram Cohen, a former graduate student, who proposed the BitTorrent protocol in 2001 [9].

2.1 The BitTorrent Protocol

How to efficiently distribute data over a network of computers has been an issue from the beginning of the Internet. The classical server/client structure comes to its limits as soon as the number of clients grows too high for the server to handle. Obvious scalability problems arise. The BitTorrent protocol tries to overcome these.

While previous systems used one centralized server to store which user had what music and connected users who wanted to share files, the BitTorrent protocol has a distributed approach.

Every file gets split into pieces of the same size. These pieces are usually between 32 KB and 16 MB large. The number of pieces a file is split into varies from several hundred to several thousand. Pieces are then uploaded and downloaded

[1] https://news.bbc.co.uk/2/hi/business/1449127.stm

independently of one another. If one has all the pieces, one can combine them to the original file.

This has some advantages to the common way of downloading a file as a whole from one source. For one, a user can stop the download of that file at any time and resume from where she left off by just downloading the still missing pieces. Also, a user can download single pieces from different sources, which is an advantage when downloading large files. When splitting a large file first into parts and then downloading each part, it does not matter when a single source of a piece goes offline. The downloader keeps the progress, i.e., the pieces she already downloaded, and finds a new source to get the still missing pieces.

Every file shared by means of the BitTorrent protocol has it's own metadata file which contains all the information needed to download a copy of that original file. This metadata file is called a *torrent* file. Torrent files are spread over torrent search engines, websites, blogs, mails, or any other means. Typically, a torrent file is about 10-100 KB large. This makes the BitTorrent protocol more resilient than previous systems which used more centralized approaches. If a website is taken down, one can easily host another which provides the same functionality. Any website or service who offers a search engine and returns the according torrent files will do. As long as you have access somewhere to the torrent file and there are uploaders who have complete copies, you can download the content since all the information needed for the download is contained in the torrent file.

This differs fundamentally from the way Napster, the first peer-to-peer sharing platform, worked. The central Napster server was used for every connection between downloaders, without the central server, no one could download anything. With the BitTorrent protocol, the downloader is no longer dependant of a single central entity.

A torrent file consists of the following:

A list of trackers:
Trackers are servers which store the IP-Addresses of users who have some, so called *leechers*, or all pieces of the desired content, so called *seeders*. When a user wants to start downloading, she asks the trackers contained in the torrent file to send a list of seeders and leechers. The pieces are then downloaded from the contacts on the list.

Information about the pieces of this file:
The amount of pieces and the length of these pieces.

Hash values:
For each piece, the torrent file contains a 20 byte SHA1 hash of that piece.

This is used to check the integrity of the downloaded pieces.

Other meta info:
The name of the content, a description of the folder structure and their size.

Each file shared by the BitTorrent protocol is also associated with a 160-bit long *infohash*. This SHA-1 hash is generated from the file's name and size, the directory structure the piece length and other information from the torrent file. This infohash is unique for each file [6].

An alternative to torrent files are *magnet* links. A magnet consists only of the infohash of a torrent and a list of trackers from which a user downloads the torrent file from. Instead of asking the trackers listed in the torrent file to find seeders and leechers, we contact a *distributed hash table* or *DHT*. The concept of DHTs is explained in the next chapter.

There are several advantages of magnets over the classic torrent files. For one, since they are only a string, they use far less space. This is not only practical for the users of torrent networks but also for torrent websites and hosters. For the users, magnets are simpler to use since they don't need to download a torrent file any more but only need to click on the magnet link. This automatically starts the torrent client and with it the download of the file.

For torrent websites magnets are useful since they can be embedded in webpages and use far less space than torrent files. This is the reason why the infamous *thepiratebay.se*[2] switched from using torrent files to magnet links in 2012. The storage requirements were reduced massively. To store all the millions of torrents on thepiratebay.se in the magnet format only requires 90 MB of compressed data, small enough to fit on a flash drive. Also, since less potentially incrimination data is stored on the website, it would be easier to claim that thepiratebay.se purely functions as a search engine as users don't have to download any files in order to find content.

2.2 The DHT Mainline

A *torrent client* is a program used to download content on the internet by using a torrent file or a magnet link. There are various torrent clients available which implement the BitTorrent protocol. One of the most common torrent clients is µtorrent which has over 150 million users.[3]

[2] https://thepiratebay.se/
[3] http://www.reuters.com/article/2012/01/09/idUS162488+09-Jan-2012+BW20120109

Once a user downloads a torrent or a magnet file and starts up her torrent client, the torrent client contacts the trackers stored on the list of trackers in the torrent file. Trackers provide a torrent client with addresses of seeders and leechers of a file. The torrent client then starts to independently download each piece the torrent was split into from the list of seeders and leechers.

Since the torrent file contains a list of trackers, should one tracker go offline or be taken down, the torrent client can just contact another tracker on the list. Even though it's unlikely that all trackers on the list are offline at the same time, trackers represent an undesired centralized component in the BitTorrent protocol. So it comes to no surprise that practically all popular torrent clients do not rely just on centralized trackers.

Since 2005, an alternative to using centralized trackers gained popularity. The torrent client *Vuze* implemented distributed tracking by utilizing a *distributed hash table* or *DHT*.[4] This DHT provides torrent clients with IP addresses of seeders and leechers without the need to contact a central tracker.

When the torrent client Vuze introduced their DHT network, the Mainline BitTorrent client implemented their own called the DHT Mainline, which is now the most widely used DHT network. The DHT Mainline was also used in this thesis.

The DHT Mainline is a large network consisting of 15 to 27 million participants in the course of a day. Most of the participants are located in Europe [3]. Participants of the DHT network are called *nodes*. Nodes communicate with each other using UDP. Each node has it's own routing table which contains the contact information of other nodes. Each node also stores contact information of *peers*.

In our context, peers are computers which run a torrent client. Peers can provide a file for download when they have the complete file or pieces of it. If a peer wants to download a torrent, it contacts nodes to get addresses of other peers to download from.

In the following, it is important to note that we make a distinction between a node and a peer. A peer is a computer who runs the torrent client, the program which downloads torrents. The torrent client controls a node which communicates with the DHT network to find other peers, i.e. computers, who download the same content. So instead of asking the centralized trackers from the torrent file for other peers, the torrent client can now just ask its own node. Thus the DHT Mainline implements distributed tracking. Essentially, each peer becomes

[4] http://plugins.vuze.com/changelog.php

a tracker.

Every torrent has its own unique infohash (see Section 2.1). A node has a similar, also unique *node-id*, which is defined over the same 160 bit long keyspace as an infohash. When a torrent client starts up for the first time, it chooses its node-id at random.

Since node-ids and infohashes have the same length, it is easy to compare node-ids against one another or node-ids with infohashes. Thus we can define a distance function which gives us a metric to compare how close two values, i.e. either infohashes or node-ids, are to one another. This distance function is defined as [10]:

$$distance(A, B) = \|A \oplus B\| \tag{2.1}$$

Where $\|\|$ indicates the cardinality of the set of one bits. So the smaller the distance, the more similar are two arguments.

Each node has a routing table which contains contact information for other nodes in the DHT. A node knows more nodes whose node-ids are close to his own node-id than nodes whose node-ids are far away. A node also knows about the location of peers who provide access to a file with a similar infohash to the node's own node-id.

If a torrent client wants to find peers to download a file from, it uses the routing table of its node. In the routing table, it searches for the node-ids which are closest to the infohash of the file the client wants to download. It then contacts these nodes and asks them if they know about peers who can provide pieces of the file.

If the asked node knows about peers who have the file with that infohash, it returns those peers. The torrent client can now start downloading from these peers. If the asked node doesn't know about any peers, it searches its own routing table for nodes closest to the infohash and returns them.

The original node now asks the received nodes about the infohash, repeating the process. In the end, the original node receives the contact information of peers, which he can download from [7].

The Figure 2.1 illustrates how the DHT Mainline works. At step 1 in the figure, the peer searches for nodes with node-ids close to the file's infohash. Nodes *N2* and *N3* have similar node-ids and are asked if they know about that particular infohash. Node N3 knows that *Peer1* has the file and sends the contact information of Peer1 as a response, denoted as step 2 in the figure. Node N2 doesn't know any peers itself. Shown as step 3 in the figure, N2 searches its own routing table and returns the contact information of node *N5*, whose node-id is close to the asked infohash.

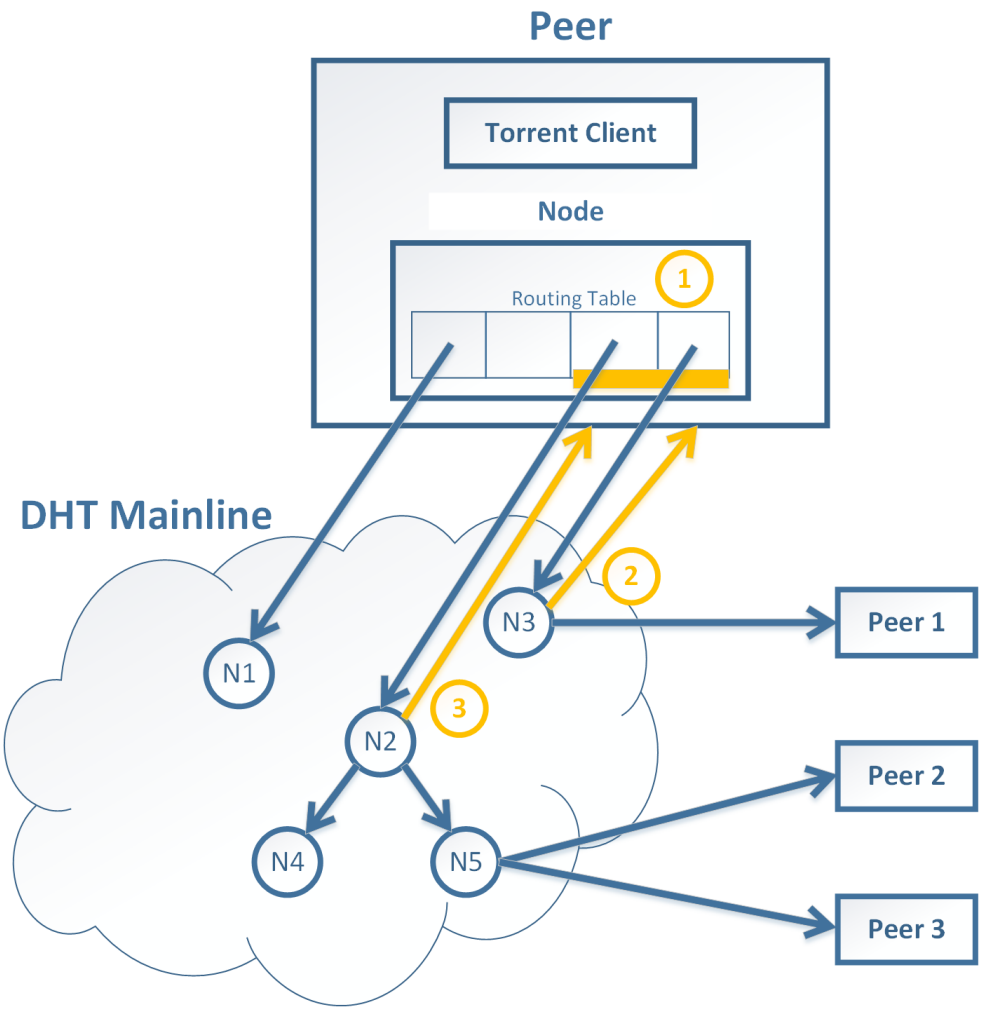

Figure 2.1: Finding peers in the DHT Mainline

2.3 Torrent Search Engines

kickass.to is the leading torrent search engine in the world.[5]At the time of writing, kickass.to is the 145th most popular website on the internet according to alexa and has several million unique visitors per day.

But kickass.to offers not only a search engine, it also provides links to IMDb, an Internet movie database collecting information of all existing movies and shows,[6] has discussion threads about its torrents and categorises and groups its content. These aspects distinguish it from other torrent search engines like *thepiratebay.se* and make it a more resourceful source to gain data from.

kickass.to categorises each torrent into one of nine categories. Torrent uploaders can optionally provide additional information, for example the cast, the release date or a summary of the plot in the case of movies. kickass.to also groups up all torrents about the same content and gives this group a name. In this thesis we call this collection identifier the *kickass affiliation*.

Each hour, about 250 new torrents get added to kickass.to. Overall the website contains about 6 million torrents. Since the founding of kickass.to in 2009, the website has been involved in various censorship lawsuits, forcing it to change domain names several times. The most recent switch was to the Somalian .so domain name.

[5] http://www.alexa.com/topsites/global;2
[6] http://www.imdb.com/

Chapter 3

Implementations

This thesis is divided into 5 parts: the *torrent crawler*, the *DHT crawler*, the *database merger*, the *recommender* and the *website*. A MySQL database was used to store and exchange data between the 5 parts. The structure of this thesis, the interactions between the programs and the tables they modify and access in the MySQL database are illustrated in Figure 3.1.

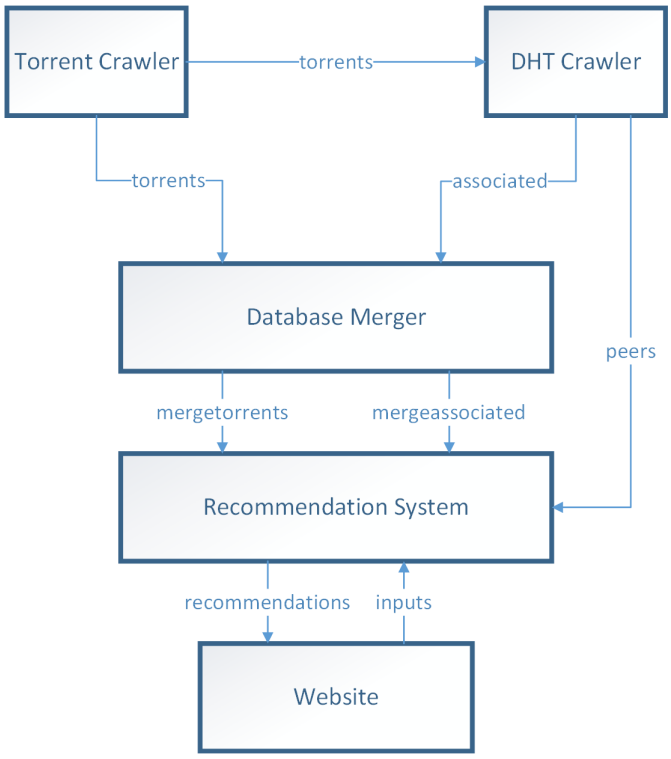

Figure 3.1: The structure of this thesis

3.1 The Torrent Crawler

As building a recommendation system for torrents is the topic of this thesis, a logical starting point was to gather data about torrents. To collect data from the torrent search engine kickass.to comes natural. It is the most frequented torrent search engine in the world and is providing more metadata than similar sites. In this thesis, a program was developed named *torrent crawler*. A MySQL database was used to store the data gathered by the torrent crawler.

The approach to the crawler was twofold: to analyse all currently existing torrents on kickass.to and secondly, to incrementally adding the newly added torrents. Based on this insight, the crawler is split into two parts.

The first part uses an archive file provided by kickass. This file is generated every 24 hours and contains a full data dump of the torrents. The archive file is a large text file and for each torrent contains the infohash and name of that torrent plus the link where that torrent can be found on kickass.to. The crawler calls each link in the archive and analyses it separately, to then store it in the database with the additional information gained from the website. This first part of the crawler uses a multi threaded approach to work through the archive. Memory mapping was used to efficiently access the data. The archive file was downloaded on 12th of November 2014 and is 1.4 GB large. As the crawler worked through the archive, it regularly checked the newly added torrents. This achieves storing all information about torrents from kickass.to by means of incremental updates.

The second part of the crawler would be responsible to analyse the newly added torrents to kickass.to. Again a multi threaded approach was used to access a page on kickass.to where all newly published torrent would appear.[1] Each entry in this list is a link to the website about a new torrent. Like in the first part, each of these torrent pages would be analysed and then together with the gathered information be inserted into the MySQL database.

On kickass.to, the webpage of a torrent provides declarations which are obligatory and others which are optional. Obligatory entries are a magnet link, the category and the amount of *seeders* and *leechers* of the torrent.

One of the optional fields is the *kickass affiliation* mentioned in Section 2.3. It groups torrents which have the same content together by the kickass affiliation. For example a Spanish language version and an English language version of a movie may have the same kickass affiliation. For movies and TV-shows another optional entry is an IMDb link. There are other optional fields but in this thesis,

[1] https://kickass.to/new/

only the kickass affiliation and the IMDb link are used.

Either from working through the newly added torrents or the archive, the torrent crawler calls the webpage of a specific torrent. Since it is possible the kickass-server does not respond, the program tries to get the webpage for up to 3 times before moving on. Kickass does work with legal authorities and responds to take down requests, some of torrent webpages are therefore not accessible any more since they got taken down.

Once the torrent crawler program gets a response from the server containing the webpage in the HTML format, the torrent crawler uses regular expressions to parse the response. We first need to get the infohash of the torrent. This is done by analysing the magnet contained in the webpage. As explained in Section 2.1, the magnet contains the infohash. Next we need to get the amount of seeders and leechers of the torrent, which is done by using regular expressions on the website to filter for leechers and seeders. Identifying the category of the torrent works in similar fashion.

In a last step we search the wepgage for optional information like a kickass affiliation or an IMDb link, again doing so by using regular expressions.

Having gathered all the information needed about a single torrent the torrent crawler inserts the data into the MySQL database. This information consists of: the category of the torrent, the amount of seeders and leechers, the infohash and, if available, the IMDb link and the kickass affiliation.

3.2 The DHT Crawler

The second part of this thesis is dedicated to gathering IP addresses of peers who download specific torrents. As explained in Section 2.2, instead of contacting central trackers to find the addresses of peers for a specific torrent, one can communicate with the DHT network to achieve the same. For this purpose, another program was written during the course of this thesis called the *DHT crawler*.

The DHT crawler maintains its own routing table containing contact information of nodes in the DHT. At first, the torrent crawler doesn't know any nodes. It therefore contacts the BitTorrent *bootstrap node* at `router.bittorrent.com`. The DHT crawler asks this bootstrap node until it knows the addresses of 100 nodes in the DHT.

Having contact information for 100 nodes is not enough to efficiently find peers

of torrents. So the program asks the nodes in its routing table for new nodes and inserts them into the routing table as well. This process is repeated until the routing table of the DHT crawler contains contact information of about 1.5 million nodes. At this time, the program knows enough nodes in the DHT to efficiently find peers for torrents.

After the initialization of the DHT crawler has finished, it can start searching for peers of a specific torrent. However, the program doesn't search all torrents we found with the torrent crawler from Section 3.1. A lot of torrents are old or outdated and have very few seeders and leechers left. Finding peers for these torrents would be futile and not contribute in a noticeable way to the recommendations. Since there are only few users downloading these torrents, the overlap with other users, on which the recommendations are based, would be negligible. Because of this, the DHT crawler only analyses torrents whose number of seeders and leechers surpass a predefined threshold. We can access the amount of seeders and leechers for a torrent in our database. The threshold of seeders and leechers was chosen to be 45.

For each torrent which has more than 45 seeders and leechers the DHT crawler now looks for peers. It does so by reading the infohash of that torrent from the MySQL database and searching for nodes in its routing table which have a similar node-id as this infohash. The found nodes are then contacted and asked if they know about that particular infohash. If they do know about peers and return the contact information of these peers, we store them. If the asked nodes don't know any peers, they return other nodes which the crawler inserts in its routing table. The DHT crawler does that several times, receiving peers in the process.

These peers are inserted into two new tables in the MySQL database. The *associated* table stores which peers download or uploads which torrent, while the *peers* table stores the IP address and port number of each peer.

3.3 The Database Merger

We have now discussed two tasks of this thesis. The first being the collection of torrents and information about them (see Section 3.1) and the second being the gathering of peers through the DHT network (see Section 3.2). The next task is the actual recommendation system based on the collected data.

But instead of running the recommendation on the raw data, we want to modify our data. The main reasoning behind this modification is, that our data treats two different torrents about the same movie as two different entities which are

not connected in any way. For example an English language version of a movie and a Spanish language version of the same movie are not recognized as belonging together. Making recommendations based on the raw data, would mean that downloaders of the Spanish and English version of the movie would be treated separately even though they have the same taste in movies. A third program was written called the *Database Merger* to alleviate this problem.

In Section 3.1, it was discussed that an entry for a torrent in the MySQL database also entails the two arguments *IMDb link* and *kickass affiliation*. These two attributes will become very useful when merging the tables in the database. The goal of this merging was to create two new MySQL tables named *mergedtorrents* and *mergedassociated*. The recommendation system would later use these two tables to generate recommendations.

Some torrents found with the torrent crawler have IMDb links. Only torrents about movies or TV-Shows can have these entries. Every torrent on the other hand can have a kickass affiliation. The kickass affiliation is an id used by kickass.to to group torrents about the same content together.

The database merger program uses both of these attributes to create the mergedtorrents and the mergedassociated tables. The program first loads the raw data form the MySQL database into memory. These tables are the torrents table created by the torrent crawler program and the associated table created by the DHT crawler program. It is important to note that only torrents above a predefined threshold of seeders and leechers are considered, the same threshold used in the DHT crawler. When reading entries from the MySQL database into memory for the first time, two dictionaries are used to group torrents which have the same IMDb link or kickass affiliation. Dictionaries are hash-tables, which in this case, get used to store a list of torrents for each kickass affiliation string or IMDb link. Using this method, we can easily find all torrents which have the same IMDb link or kickass affiliation.

After having read the raw data into memory and creating the two dictionaries, the program iterates through each torrent. It then looks up all torrents which have the same kickass affiliation or IMDb link in the dictionaries and inserts them into the mergedtorrents table with the same merge id. So all torrents about the same content have the same merge id. Similarly, the program goes through all entries in the associated table and stores all downloaders we grouped before under the same merge id.

This procedure results in the two tables mergedtorrents and mergedassociated which contain IP-addresses of downloaders for each merge id. Torrents which have the same content are assigned the same merge id. This allows the recommendation system to generate more useful recommendations. Persons who have

the same taste will now be recognised by the system even when they download different versions of the same content.

3.4 The Recommender

Item based collaborative filtering was used to generate recommendations. A program named *recommender* was written to generate recommendations based on the data collected and merged previously.

The recommender first reads the two tables *mergedassociated* and *mergedtorrents* form the MySQL database. Item based collaborative filtering first calculates the similarity between two torrents for all torrents in the database. Cosine based item similarity is used in this thesis and is calculated in the following way, where i and j are torrents:

$$sim(i,j) = cos(i,j) = \frac{i \cdot j}{\|i\| * \|j\|} \tag{3.1}$$

To express this equation in words, the similarity of two torrents is equal to the number of people who downloaded both torrents divided by the number of people who downloaded either one of the torrents.

As soon as the similarity for each pair of torrents is known, the program can compute recommendations. The recommender computes a prediction value for each torrent for each user. The torrents with the highest prediction values are then the recommendations for a user. The following formula was used to compute the prediction value for a user u and a torrent t:

$$P_{u,t} = \frac{\sum_i s_{t,i} \cdot h_{u,i}}{\sum_i s_{t,i}} \tag{3.2}$$

Where $s_{t,i}$ is the similarity of the two torrents t and i and $h_{u,i}$ indicates whether the user u has downloaded torrent t or not. The prediction value of a torrent is the sum of all similarities between the torrent and the torrents downloaded by the user divided by the sum of all similarities between the torrent and all other torrents [11].

3.5 The Website

The last step in this thesis was to provide access to the service through a website. The website offers information about the thesis and instructions how to use the recommendation service.

The website first checks if the visitor's IP address is already in the system. If the system already knows about the visitor, it does not require any input from the visitor and displays the recommendations for him or her directly. If the visitor is unknown to the system, the website provides an input table where the visitor can insert links to his or her favourite torrents on *kickass.to*. The links then get sent to the recommendation system to generate a list of recommended torrents for that user, which in turn get displayed on the browser of the visitor.

Technically, the website consists of only one HTML webpage which communicates with a PHP script running on the server. On first contact, the server uses a message passing system to forward the IP address of the visitor to the recommendation system. If the user is known, the recommendations are passed to the server which in turn sends the information back to the webpage. If the visitor is unknown to the system, the wepage dynamically displays an input form using jQuery. The visitor enters links to his or her favourite torrents, which get sent to the server and passed on to the recommendation system. The generated recommendations get displayed in the webpage as soon as they are available.

Chapter 4

Evaluation

The Torrent Crawler

To analyse the newest 10'000 torrents added to kickass.to, the torrent crawler takes about 15 minutes. After analysing each torrent on kickass.to with the torrent crawler, the MySQL database now contains over six million torrents. A distribution of the categories of active torrents can be found in Figure 4.1. A torrent is considered active if it has more than 45 seeders and leechers. There are a total of 192'639 active torrents in the MySQL database with a total of 6'166'840 torrents. This means about 3% of all torrents kickass.to can be considered active.

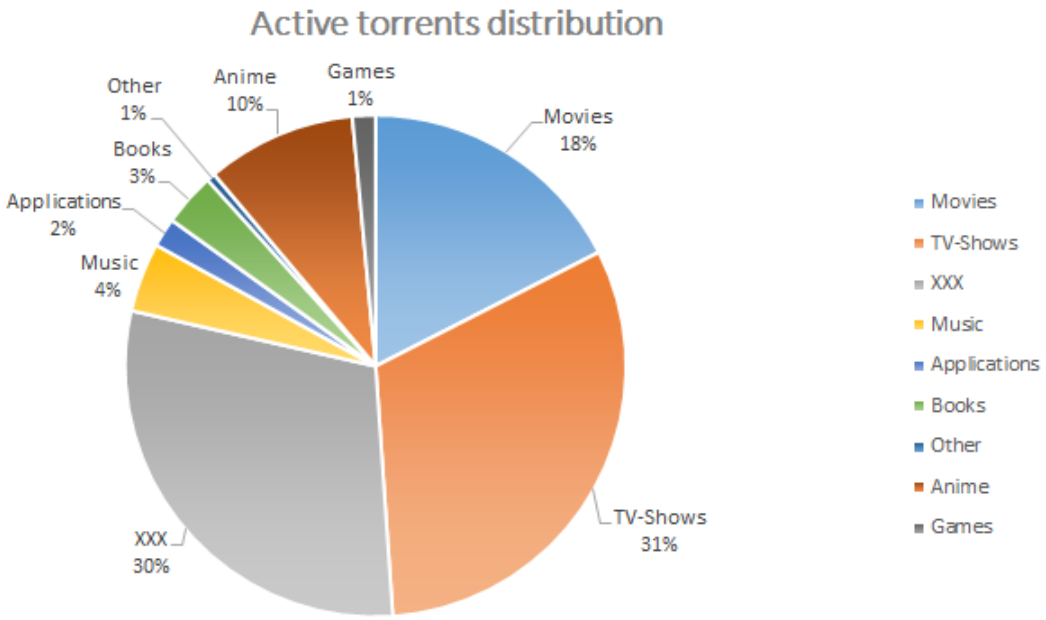

Figure 4.1: The structure of this thesis

About 23% of all torrents have either a kickass affiliation or an IMDb link. Of the active torrents, about 50% have either a kickass affiliation or an IMDb link. Another interesting fact is that roughly one sixth of all uploaded torrents are music files but 99% of all music files uploaded to kickass.to have less than 45 seeders and leechers.

The DHT Crawler

The DHT crawler found 9'337'841 distinct IPs downloading 34'192'448 torrents. This means, the DHT crawler found a user downloading 3.6 torrents on average.

The Database Merger

As was mentioned in the evaluation of the torrent crawler, about 50% of active torrents have neither a kickass affiliation nor an IMDb id. This means the merger can't merge 50% of the active torrents, since the merging is done using these two attributes. Merging the other 50% results in clusters of an average size of 7.3. This means there are on average 7.3 active torrents about the same content. This can be expected since there are for example not only versions of the same movie in different languages and qualities that get merged together, but the merger also merges episodes of TV shows together, i.e., all episodes of the same TV show are merged together.

Merging the information generated by the DHT crawler, about 18.5% of the data gets merged together. 34'192'448 entries get merged to 27'894'837 entries. This can be explained by the insight that TV shows make 31% of the total of active torrents and that TV shows get merged together by the merger. It is very likely that a user, when she likes a TV show, downloads several episodes of that TV show. If the DHT crawler catches that user downloading several episodes, the merger merges these entries into one.

The Recommender

The recommender has about 15 hours to compute the item to item similarity matrix. After the similarity matrix is computed, it takes the recommender one or two seconds to generate recommendations for a single user.

To measure the quality of the recommender, a system was used which compares the recommendations made by the recommender with random recommendations [12]. An evaluating mechanism was used to generate a value between zero and one for the recommendations generated by the recommender. A result below 0.5 means that the recommendations are better than random recommendations. Evaluating the recommendations for about 20'000 users, a result of 0.39 was measured.

Chapter 5

Conclusions

BitTorrent search engines like *kickass.to* and the DHT Mainline provide an enormous source for data. An interesting fact is that all of this information is publicly available and easily accessible. This means that everyone can basically see who downloads which torrents, a fact that most torrent users nowadays may be unaware of. That such a distributed system lacks in privacy may seem obvious but many users of torrent networks may be oblivious to the way torrents work and thus be unaware how openly accessible their private information is.

The DHT Mainline is a largely untapped source of data, so collecting and analysing data gathered from the DHT mainline and torrent search engines can offer valuable new insight into the inner workings of peer-to-peer sharing networks.

Generating recommendations is a practical example of how to use the gathered data. Four programs, the torrent crawler, the DHT crawler, the database merger and the recommender were written and a website was created to gather, cluster, generate and display recommendations.

Bibliography

[1] sandvine: "Global Internet Phenomena Report 1H 2014", "sandvine, Intelligent Broadband Networks" (2014)

[2] Wolchok, S.: "Crawling BitTorrent DHTs for Fun and Profit". In: "DEF CON 18". (August 2010)

[3] Wang, L., Kangasharju, J.: "Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT". In: "Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on". (September 2013)

[4] Ritesh Mittal, D.K., Jiwtode, M.: "Torrentzshare Search And Recommendation", "Bharatiya Vidya Bhavan's Sardar Patel Institute of Technology" (2013)

[5] Johan Pouwelse, Michiel van Slobbe, J.W.M.R., Sips, H.: "P2P-based PVR Recommendation using Friends, Taste Buddies and Superpeers", "Delft University of Technology" (2005)

[6] Cohen, B.: "The BitTorrent Protocol Specification". In: "DHT Protocol, BEP 3". (January 2008)

[7] Loewenstern, A., Norbert, A.: "DHT Protocol". In: "DHT Protocol, BEP 5". (January 2008)

[8] Aberer, K., Hauswirth, M.: "Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems", "Distributed Information Systems Laboratory (LSIR), EPFL" (2001)

[9] Cohen, B.: "Incentives build robustness in BitTorrent". (2003)

[10] Maymounkov, P., Mazières, D.: "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: "First InternationalWorkshop, IPTPS 2002". (2002)

[11] John S. Breese, D.H., Kadie, C.: "Empirical Analysis of Predictive Algorithms for Collaborative Filtering". In: "Microsoft Research, Technical Report MSR-TR-98-12". (1998)

[12] Yifan Hu, Yehuda Koren, C.V.: "Collaborative Filtering for Implicit Feedback Datasets". (2003)