Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Torrent Recommendation System Based on Data Gathered from the Mainline DHT

Bachelor Thesis

Pascal Widmer

pawidmer@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Christian Decker
Prof. Dr. Roger Wattenhofer

April 22, 2015

Abstract

BitTorrent is the most widely used peer-to-peer file sharing protocol and attracts millions of users every day. It allows for easy distribution of all sorts of files including movies, TV shows, music, games, and other applications. In this thesis a closer look is taken at what files users are sharing. To acquire the necessary information a list of active torrents and in particular their infohashes is extracted from a popular torrent indexing website. In a second step all peers sharing a given torrent are found using look-ups in the Mainline DHT and aggregated in a database. After gathering enough data torrent recommendations are calculated using the technique of collaborative filtering. These are then presented to visitors on a website.

Contents

Chapter 1

Introduction

Ever since distributed hash tables (DHT) became popular as an addition to trackers a lot of work has been done in regards to monitoring the DHT and its users. Probing the DHT and capturing messages exchanged between nodes helps uncover performance bottlenecks and design flaws and it allows to analyze the composition of its users. In particular the user's geographic locations, their seeding behavior within swarms and their average stay in the network have been examined. Only rarely did studies investigate which torrents users of the DHT were sharing. In this paper the torrent indexing website kickass.to[1] is accessed and infohashes of over 6 million torrents are extracted and stored in a database. They are used to uniquely identify the torrent and serve as a means to find all peers sharing the torrents in the Mainline DHT. A complete recommender system is built on top of this data helping users of the DHT find similar torrents. Naturally it is not possible to monitor all torrents and their peers at any given moment using the DHT, but to build a recommender system complete information is not required and a representative sample of peers suffices. A recommender system which uses the common technique of collaborative filtering similar to what the online shop Amazon [1] has implemented is used to calculate predictions.

1.1 Related Work

Much research has been done on the topic of monitoring users of torrent DHTs. Jünemann et al. developed a DHT crawler which monitors the number of peers in the DHT and the peer's IP addresses, used port numbers, countries of origin and session lengths [2]. Memon et al. developed a crawler which monitors traffic within the DHT [3]. Scott Wolchok and J. Alex Halderman [4] show how a torrent search engine of over one million torrents can be recreated by crawling the DHT used by Vuze, a BitTorrent client previously known as Azureus. Their DHT crawler makes use of a well known exploit called *sybil attack*. The attack

[1] https://kickass.to/

works by simulating thousands of nodes and waiting for the DHT to replicate values of neighboring nodes to these nodes. Michael Piatek et al. [5] show that law enforcement agencies are regularly monitoring both tracker servers and the DHT for copyrighted material. They show how easy it is to accredit arbitrary users with sharing illegal files successfully misleading law enforcement agencies. Zeilemaker et al. [6] use a different approach to gather user data. They present a way to monitor millions of peers in the Mainline DHT by exploiting caches of the DHT's bootstrap servers. This allows to determine geographic location of peers but no information about the files that are shared. In this thesis data is gathered by inserting a few nodes into the Mainline DHT and actively querying for infohashes found on a popular torrent indexing website. While gathering all peers at all times is not feasible a representative sample of data is enough to build a recommender system on top of it.

Chapter 2

Background

2.1 BitTorrent Protocol and Torrents

BitTorrent is a protocol used for peer-to-peer file sharing, it was designed by Bram Cohen in the year 2001 [7]. Various BitTorrent clients implementing the protocol have since been developed and roughly 15 percent of Europe's internet traffic can be attributed to the BitTorrent network [8]. Every day millions of user upload and download files using one of the many BitTorrent clients. In a traditional client-server model a download can often not be resumed once the connection is interrupted and users are then forced to restart the transmission. In the client-server model the file itself is stored on a central server and its operators are responsible for the content they provide. When many users access files on the same server scalability issues arise and the download speed decreases for all users. BitTorrent mitigates some of these problems. It allows users to pause and resume downloads and distributes network traffic on many different connections without having to rely on a central server. This is achieved by first segmenting the files into multiple equally sized parts and then sharing them simultaneously with all users interested in the same file making use of each users upload capacity. Because files are transmitted from peer to peer it allows for no easy way to remove copyrighted material from the network. This is arguably one of the main factors that contributed to BitTorrents tremendous growth since its inception.

2.1.1 How BitTorrent works

Downloading a file from the BitTorrent network involves several steps. Initially a user has to find a torrent file corresponding to the file she is interested in. A *torrent file* or simply *torrent* does not contain the file itself but instead provides information needed to find users sharing the file. *Torrent indexing websites* such as piratebay.se[1] provide a convenient way of finding torrents. Users can search a

[1] https://thepiratebay.se/

specific title of a file they are interested in or browse various categories. A list of torrents is then displayed and a user can download any one of them. Alternatively some websites offer *magnet links* instead of torrent files. A magnet link serves the same purpose but does not need to be downloaded and can directly be parsed by a BitTorrent client. Besides being more convenient for users magnet links also reduce storage consumption on torrent indexing websites. Upon completing the download the torrent is opened with a BitTorrent client which downloads the actual file described by the torrent. Note that depending on the context torrent might refer to the content as well. The torrent downloaded from the indexing website contains among other things addresses of one or more trackers, a list of SHA-1 checksums for each part of the file and an infohash which uniquely identifies the torrent. *Infohashes* are SHA-1 checksums and can also be calculated using all other information in the torrent file as input. A *tracker* is a server which keeps track of users sharing the file. It can be contacted by the BitTorrent client and it will return a list of peers sharing the same file. A *peer* is a BitTorrent client instance sharing one or more files with other users. The SHA-1 checksum or hashes are used to check the integrity of the downloaded parts. Some well-known BitTorrent clients for opening the downloaded torrent include μtorrent, Vuze, BitTorrent and Transmission, all of which are freely distributed on the web. When a torrent file is opened the BitTorrent client proceeds with finding peers which share the same file. All peers sharing the same file are referred to as the torrent's *swarm*. Finding peers in the torrents swarm can be done by either asking the trackers provided in the torrent file or by querying the DHT for the torrent's infohash. The method involving the DHT is covered in more detail in the following section. When peers are found the BitTorrent client tries to connect to them and request the missing parts of the file. At the same time requests from other peers are processed and requested parts are uploaded. Peers which are still downloading part of the file are called *leechers* while peers which have acquired the complete file but still share it with others are called *seeders*. When all parts of a file are downloaded they are reassembled to yield the complete file. The completed file is usually distributed until the user decides to stop sharing the file. The ratio between seeders and leechers plays a crucial role in the speed of which a file is distributed in the swarm. When a torrent has lots of seeders and few leechers the torrent is considered healthy and downloads generally finish fast. When no peer possess a certain part of a file the torrent download is stalled and the file can no longer be downloaded. Oftentimes when a new torrent file is published on piratebay.se it experiences a surge of new downloaders but users rarely seed after their download has finished.

2.2 Mainline DHT

The Mainline Distributed Hash Table, from now on referred to simply as DHT, provides a way for peers to find each other without contacting one of the centralized trackers listed in the torrent file. Its design is based on a DHT called Kademlia introduced in 2002 [9]. Hash tables are data structures that map keys to values. In the case of the DHT the values are lists of peers and keys are torrent infohashes. The hash table is distributed among its participants called nodes. Each node in the network is responsible for storing part of the hash table and providing the peer list if asked for it. In effect each node using the DHT becomes a tracker itself and provides information on where to find peers. Note that BitTorrent client instances participating in the DHT are called nodes while instances participating in a swarm of a torrent are called peers. Upon joining the DHT a BitTorrent client generates a random 160 bit address called *node ID* which uniquely identifies the node and determines the node's location in the DHT. The *KRPC protocol* specifies how DHT nodes may interact with each other. It allows for three different types of messages, namely queries, responses and error messages. The most common query is *get_peers* which includes a torrent's infohash as an argument and is used to find peers sharing the file. Messages are sent in a special format called *bencoding*, which is also used to encode torrent metafiles. Bencoding is used to structure the messages and helps make BitTorrent platform independent. Each node maintains a routing table containing a list of buckets. Each bucket contains a fixed number of nodes, usually around eight. A node consists of a node ID, an IP and a UDP port. The routing table is designed in a way such that in the beginning only one bucket covers the whole 160 bit address space of node IDs. New nodes are inserted into this bucket until it is full. When it is full the bucket is split into two buckets, each covering half of the address space. The bucket containing the node ID associated with the routing table is further split when it becomes full while the second bucket is never split. This is done recursively while at the same time removing and replacing nodes which have not sent any message within 15 minutes. It ensures that the node associated with the routing table knows a lot of close and active neighbors in the DHT but only a few that are far away. Closeness is measured with the XOR metric. If two node IDs are interpreted as a binary number and XOR is used on them then the resulting number is defined as the distance between them. The same procedure can also be used on an infohash and a node ID. When the resulting value is small then the node ID is more likely to know about a given infohash. When a node receives a get_peers query from another node it either replies with an answer containing a list of peers which are sharing the file or it returns nodes from its routing table which are more likely to have an answer, i.e., are closer to the queried infohash. Finding peers can then be seen as iteratively asking closer nodes about the infohash until a node knows about it.

Chapter 3

Implementation

The implementation can be roughly divided into four separate steps, although technically they could all be run simultaneously. As a common interface all data is aggregated in a SQL database.

1. **Torrent Crawler**
   The Torrent Crawler accesses all webpages of kickass.to, parses the HTML code and uses regular expressions to extract information about torrents. In particular it extracts each torrent's infohash, name, IMDb[1] (Internet Movie Database) reference number, weblink to kickass.to and number of leechers and seeders. The information is stored in a SQL table.

2. **DHT Crawler**
   In a second step the DHT Crawler is run. Its job is to look up the infohash stored in the database by the Torrent Crawler and find all peers currently in the torrents swarm. This is done by querying the DHT for the infohashes and then aggregating this information in the SQL database.

3. **Recommender System**
   Using the aggregated data about the composition of each torrent's swarm recommendations are calculated. The recommender system works on an IP basis, which means two peers sharing an IP but using different ports are considered to be the same user. Calculating the recommendations is done using a collaborative filtering technique called cosine similarity.

4. **Website**
   The final recommendations are presented on a website. A visitors IP is automatically determined and predictions are calculated based on what she has shared in the past. Additionally users can manually enter links to kickass.to webpages of their favorite torrents. These links are parsed and interpreted as if the user downloaded the torrent to provide more accurate predictions.

[1] http://www.imdb.com/

3.1 Torrent Crawler

In a first step a list of torrents has to be retrieved to provide infohashes for the DHT crawler. The torrent indexing website kickass.to provides an archive of all the torrents they index which is updated every 24 hours. It consists of around 6 million entries and includes an infohash, a name and a link to the torrent's webpage for each torrent. Because some of the listed torrents are not active anymore, i.e., they can not be downloaded anymore because parts of it are not shared by anyone and some information is missing in the archive file, the listed webpages still have to be accessed to find the missing information. Some webpages have already been removed due to legal issues. The remaining webpages are downloaded and all data is extracted using regular expressions. This includes the torrent's name, the amount of seeders and leechers, its genre as specified by kickass.to, its kickassaffiliation and if available a link to IMDb. When two torrents are of similar content, e.g., two episodes of the same TV show, they have the same kickassaffiliation. In the case of two TV show episodes the *kickassaffiliation* is simply the name of the TV show. Two torrents which either have the same kickassaffiliation or reference the same IMDb entry are later viewed as one entity when recommendations are calculated. Kickass.to also has a page that lists links to the more recent torrents which are not contained in the archive file yet. This page is periodically rescanned and all linked webpages are accessed and information extracted just as with the links in the archive. All gathered information is eventually stored in the database and later accessed by the DHT crawler. Kickass.to is used as opposed to thepiratebay.se because it provides an easier way to extract the categories of individual torrents and piratebay.se has sometimes been offline due to legal issues.

3.2 DHT Crawler

The DHT Crawler is responsible for finding all peers associated with each torrent infohash by querying the DHT. Instead of populating a routing table with around one hundred active nodes and repeatedly sending get_peer requests to the closest nodes the way it is described in the protocol specification [10] the decision was taken to restructure the routing table. The 160 bit address space is divided into 65536 (2^{16}) equally sized buckets, each of which covers some predefined address space. The first two bytes of each node ID is interpreted as an integer n and put in the n-th bucket together with the node's IP and UDP port. This speeds up infohash look-ups as finding peers for a given infohash is now a matter of simply asking the nodes in the bucket corresponding with the infohash and possibly its neighboring buckets.

Bootstrapping Initially the buckets are empty and need to be populated. To reduce stress on the bootstrap server *router.bittorrent.com* only one hundred nodes are requested. Further nodes are found by iteratively querying the returned nodes for random infohashes. After a few minutes each bucket averages around twenty nodes amounting to a total of around 1.5 million starting nodes.

Finding peers Two identical threads continually fetch the 10 least recently used infohashes from the database. For every infohash they send get_peer queries to all nodes of the corresponding bucket and its neighboring buckets. When peers are returned they are accumulated and later inserted into the database. If closer nodes are returned they are reinserted into the buckets and asked again. This is done around 10 times before fetched the next 10 infohashes. All replies are processed and decoded in background threads. Reinserting closer nodes has the advantage that buckets become even denser populated and fewer loops are necessary to find most peers for an infohash. Because the database itself limits the speed at which infohashes can be queried there is no need to immediately remove inactive nodes from the buckets.

3.3 Recommender System

The purpose of a recommender system is to predict and recommend additional items a particular user might be interested in given the users previous likings. The two classical approaches of doing so are content-based filtering and collaborative filtering. Content-based filtering works by finding items with similar properties to the items the user liked. In case of torrents, imagine a user downloaded a movie starring Robert De Niro. Then a content-based recommender might suggest a different movie where the main actor is Robert De Niro. In this thesis a more common method was implemented called collaborative filtering. The idea behind collaborative filtering is that if user 1 downloaded item A and user 2 downloaded item A and item B, then there is a greater chance that user 1 likes item B as well. The main advantage of collaborative filtering is its ability to predict what a user might like without analyzing the items themselves.

3.3.1 Item-item-based Collaborative Filtering

While there are many different collaborative filtering methods, not all of them suit the gathered data equally well. Torrents tend to be very short lived as most users don't seed their torrents once their download has finished. The vast amount of different users using the DHT compared to the relatively few active torrents suggests using an item-item based similarity table as opposed to a user-user based similarity table. In an item-item based similarity table entries change only little when new users rate items because items are rated by many users

and a single user's rating has little effect on its value. The DHT itself does not provide a way for users to rate torrents, as such the implicit ratings are used. In other words, being an active seeder or leecher of a torrent is equivalent to liking it. Thus all peers which participated in the swarm of a torrent and were active in the DHT at the time of the crawl have implicitly rated the torrent 1 and all others rated the torrent 0. This also removes the need to normalize ratings, i.e., subtract the user's mean rating from every rating.

Cosine-based Similarity

Item Similarity Calculation In cosine-based similarity [11] each torrent is represented as a n-dimensional vector $\vec{v}$ where n is the total number of peers found by the DHT Crawler. The i-th component of the vector is 1 if the i-th peer has downloaded the torrent and 0 otherwise. The similarity between two torrents $\vec{v_1}$ and $\vec{v_2}$ is then defined by

$$similarity(v_1, v_2) = \cos(\vec{v_1}, \vec{v_2}) = \frac{\vec{v_1} \cdot \vec{v_2}}{\|\vec{v_1}\| * \|\vec{v_2}\|} \tag{3.1}$$

where $\cdot$ is the scalar product. Since all vector components are either 0 or 1 the resulting similarity values lie in the interval [0,1]. A value of 0 signifies no correlation between the two torrents, i.e., no peers in common and 1 complete correlation, i.e., all peers are identical. Calculating the similarities for all combinations of torrents yields a (possibly sparse) $m \times m$ item-item similarity matrix, where m is the number of torrents. In certain cases where two torrents have only very few peers and those few peers happen to coincide the equation 4.1 will return similarity 1 even though not much can be said about the two torrent's similarity. To reduce this effect torrents with less than 3 common peers are never considered to be similar and their similarity is set to 0.

Prediction calculation Having calculated the similarity between each pair of torrents a list of predictions for a user has to be determined. Let an element of the item-item similarity matrix be denoted as $s_{i,j}$. Let n be a torrent similar to torrent t and $r_{u,n}$ the rating user u gave to torrent n. $r_{u,n}$ is 0 if user u did not download torrent n and 1 otherwise. Then the predicted rating user u would give to an item t is defined by

$$prediction(u, t) = \frac{\sum_n s_{t,n} * r_{u,n}}{\sum_n s_{t,n}} \tag{3.2}$$

The resulting predicted rating estimates how user u would rate torrent t and is again in the interval [0,1]. In a Top-N Recommendation System the N torrents with the highest predicted ratings are recommended to the user. Torrents which are already rated by the user are removed from the list of recommendations.

3.4 Website

The website provides a front-end for users to obtain torrent recommendations. A visitor's IP is automatically discovered and handed over to the recommendation tool which in turn calculates and returns a list of torrent recommendations. The list of torrents is displayed as links to the corresponding kickass.to webpage. A visitor can additionally enter a list of kickass.to links from individual torrents she liked. These are then parsed and interpreted as if the user downloaded the torrents and are similarly used to calculate recommendations.

Chapter 4

Evaluation

The torrent crawler periodically checks the 10'000 newest torrents on kickass.to and inserts them into the MySQL database. This process takes around 15 minutes for each run. From the total 6'166'840 inserted torrents 192'639 have a combined amount of at least 45 seeders and leechers and are considered active. The chart in Figure 4.1 shows the distribution of categories among these active torrents. Only active torrents are further analyzed by the DHT crawler. After a few days of crawling the DHT crawler found 20'118'613 unique peers. When viewing peers with the same IP but different ports as belonging to the same user this results in 9'337'841 users sharing a total of 34'192'448 torrents. On average each user partakes in 3.6 swarms. Constructing the an item-item similarity matrix takes around 15 hours and another 1 or 2 seconds is needed to generate a list of recommendations for an individual user.

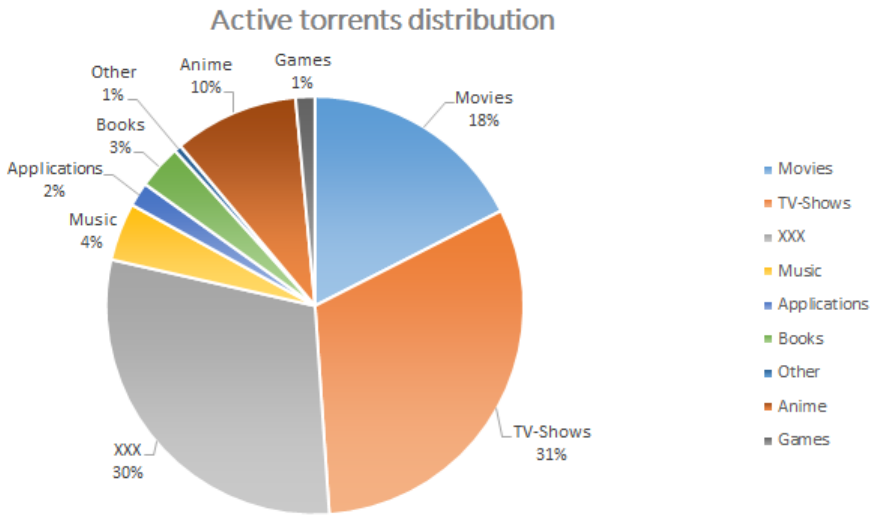

Figure 4.1: Genres of torrents published on kickass.to

Recommender system evaluation For the purpose of measuring the recommendation quality the recommender system is compared with a system that predicts torrents randomly [12]. Note that in the list of predictions torrents that a user has rated are not removed yet. The recommendation quality for a single user is calculated by first summing up the position in the sorted list of predictions of each torrent the user has rated. A lower position in the list corresponds to a higher predicted value. This sum is then divided by the total number of ratings the user has given yielding an average position in the list. The average position is again divided by the number of evaluated torrents to obtain a value between 0 and 1. If the resulting value is lower than 0.5 we can conclude that the recommeder system works better than a random recommender system because the rated torrents are on average in the lower part of the prediction list. This procedure is repeated for all users and the resulting values are averaged to get a more accurate measure. Let u be a user and t a torrent rated by user u. Let $p_{u,t}$ be the position of torrent t in the list of predictions calculated for user u and $ratings$ the total number of ratings from all users. Then the final measure of quality can be written as

$$quality = \frac{\sum_u \sum_t p_{u,t}}{ratings} \tag{4.1}$$

Because of performance reasons not all users can be included but instead a random sample is chosen. Using roughly 20000 users a quality of 0.39 is achieved.

Chapter 5

Conclusion

A great amount of data was gathered on the course of this thesis. While many researches have monitored the DHT to gain insights into its composition and analyze its weaknesses not many have used their gathered data in a practical application as is done here in the form of recommendations. It is interesting to think about how BitTorrent users are most likely not aware of being monitored. Even if some users heard about the fact that law enforcement agencies monitor certain files they might not expect their torrent download history to be archived by researchers. Visitors to the website might be surprised when a list of torrent recommendations is automatically displayed.

Bibliography

[1] Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing **7** (2003) 76–80

[2] Jünemann, K., Andelfinger, P., Dinger, J., Hartenstein, H.: Bitmon: A tool for automated monitoring of the bittorrent dht. In: Peer-to-Peer Computing'10. (2010) 1–2

[3] Memon, G., Rejaie, R., Guo, Y., Stutzbach, D.: Montra: A large-scale DHT traffic monitor. (2012) 1080–1091

[4] Wolchok, S., Halderman, J.A.: Crawling bittorrent dhts for fun and profit (2010)

[5] Piatek, M., Kohno, T., Krishnamurthy, A.: Challenges and directions for monitoring p2p file sharing networks. In: In 3rd USENIX Workshop on Hot Topics in Security (HotSec '08. (2008)

[6] Zeilemaker, N., Pouwelse, J.: 100 million dht replies. In: Peer-to-Peer Computing (P2P), 14-th IEEE International Conference on, IEEE (2014) 1–4

[7] Cohen, B.: Incentives build robustness in bittorrent (2003)

[8] Incorporated, S.: Global Internet Phenomena Report, 1H 2014. Technical report (April 2014)

[9] Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric (2002)

[10] Loewenstern, A., Norberg, A.: BEP 5: DHT protocol. http://bittorrent.org/beps/bep_0005.html (2008)

[11] Badrul Sarwar, George Karypis, J.K., Riedl, J.: Item-based collaborative filtering recommendation algorithms

[12] Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: In IEEE International Conference on Data Mining (ICDM 2008. (2008) 263–272