

Peer-to-peer networking with BitTorrent

Jahn Arne Johnsen
jahnarne@stud.ntnu.no

Lars Erik Karlsen
larserka@stud.ntnu.no

Sebjørn Sæther Birkeland
sebjorns@stud.ntnu.no

Department of Telematics, NTNU - December 2005.

Abstract

Peer-to-peer networking has in recent years received a lot of attention due to the ongoing battle with the music and movie industry. Despite many beliefs it is not a new concept but, in its simplest form, has existed for over four decades and can be traced back to the original implementation of the Internet.

BitTorrent is a distributed peer-to-peer system which, it is stated, has the potential to change the landscape of broadcast media and file distribution. It uses a symmetric (tit-for-tat) transferring model in an attempt to reach Pareto efficiency. Its protocol employs various mechanisms and algorithms in a continuous effort to try to ensure that there are no other variations in the network arrangement which will make every downloader at least as well off and at least one downloader strictly better off. In its original implementation, BitTorrent base its operation around the concept of a torrent file, a centralized tracker and an associated swarm of peers. The centralized tracker provides the different entities with an address list over available peers. Later improvements tries to eave the weakness of a single point of failure (the tracker), by implementing a new distributed tracker solution.

Amongst BitTorrent's most prominent future uses are applications which combine BitTorrent and RSS (Really Simple Syndication), with the powerful ability to transform the Internet into the world's largest TiVo.

Table of contents

1	INTRODUCTION	1
2	PEER-TO-PEER NETWORKING	2
2.1	PEER-TO-PEER NETWORK TOPOLOGIES	3
3	INTRODUCTION TO BITTORRENT	4
3.1	THE HISTORY OF BITTORRENT	4
3.2	AREAS OF USAGE	5
4	THE BITTORRENT ARCHITECTURE	7
4.1	ALGORITHMS	8
4.1.1	<i>The Piece Selection Algorithm</i>	8
4.1.2	<i>Resource Allocation</i>	10
4.1.3	<i>Concluding remarks</i>	11
4.2	IMPORTANT IMPROVEMENTS	11
4.2.1	<i>Bulk traffic marking</i>	11
4.2.2	<i>Decentralized tracker</i>	12
5	THE FUTURE OF BITTORRENT	17
6	CONCLUSION	18
7	REFERENCES	19

1 Introduction

Currently software using peer-to-peer networking solutions are becoming increasingly popular. Compared to the more common server-client solution, a peer-to-peer approach has several advantages including increased robustness and resource providing, such as bandwidth, storage space and computing power, by peers. One area where robustness and utilization of resources is important is file distribution, especially of large files. An example of a peer-to-peer solution that has proven to be an efficient and reliable alternative to the classical server-client solution is BitTorrent.

This essay will focus on BitTorrent as a peer-to-peer solution and explain the architecture and concepts that make up BitTorrent. The self-configuring parts will be highlighted as this essay is a part of the course TTM3 Self Configuring Systems. A brief look into area of use, history and the future of BitTorrent is also presented and discussed.

Part one of this essay gives a brief introduction to peer-to-peer networks in general. Part two focuses on BitTorrent.

2 Peer-to-peer networking

“The emergence of peer-to-peer computing signifies a revolution in connectivity that will be as profound to the Internet of the future as Mosaic was to the Web of the past”

- Patrick Gelsinger, Vice President and CTO, Intel Corp.

Although peer-to-peer networking has received a lot of attention recently due to the ongoing battle with the music and movie industry, it is not a new concept. In its simplest definition, peer-to-peer is described as [23]:

“A communications model in which each party has the same capabilities and either party can initiate a communication session” [23]

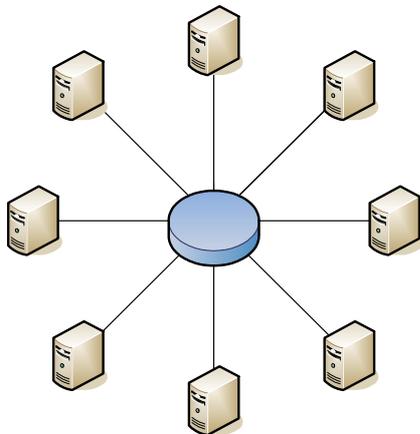


Figure 1

This means that conceptually, peer-to-peer computing is an alternative to the traditional client / server architecture where there typically is a single (or small cluster) server and many clients (figure 1).

Sticking with the previous definition, the concept of peer-to-peer can be traced back to the 1960's when the early implementation of the Internet (ARPANET) was a peer-to-peer network in which all its peers were equals. However, this definition can be fitted into many scenarios. The Domain Name System (DNS) is a good example of a blend between the traditional peer-to-peer networking and a hierarchical model of information ownership. A more precise definition is stated in [17] as:

“A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P,...) network, if the participants share a part of their own (hardware) resources (processing power, storage capacity, network link capacity, printers,...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servant-concept).” [17]

As there exists many forms of peer-to-peer networks, both with and without some with a form of central entity, this definition was further refined in [17]. This refinement introduced a classification of peer-to-peer networks as either pure (Figure 2) or hybrid (Figure 3). This was done so one is be able to distinguish between peer-to-peer networks with or without the previously mentioned central entity.

2.1 Peer-to-peer network topologies

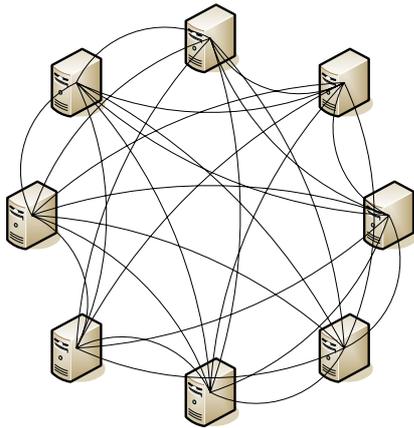


Figure 2

2.1.1 Pure peer-to-peer

The “pure” peer-to-peer concept is a network in which the peers themselves are the only entities allowed within. Or as stated in [17]:

“A distributed network architecture has to be classified as a “Pure” Peer-to-Peer network, if it is firstly a Peer-to-Peer network according to Definition 1 and secondly if any single, arbitrary chosen Terminal Entity can be removed from the network without having the network suffering any loss of network service.”

This concept is shown in Figure 2. Here we can clearly see that only what seem to be equal peers are present in the network. All are interconnected which means that any peer can be removed, without this having any fatal consequences on the network, i.e. there is no single point of failure.

2.1.2 Hybrid peer-to-peer

Different from a “pure” peer-to-peer concept, where only the peers themselves were allowed in the network, a “hybrid” network will always include some sort of central entity. This is shown in Figure 3. In [17] the hybrid peer-to-peer concept is defined as:

“A distributed network architecture has to be classified as a “Hybrid” Peer-to-Peer network, if it is firstly a Peer-to-Peer network according to Definition 1 and secondly a central entity is necessary to provide parts of the offered network services.”

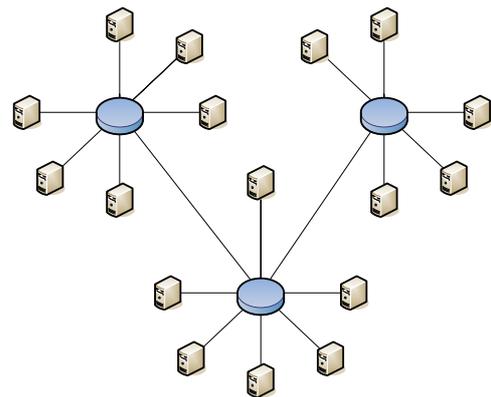


Figure 3

As Figure 3 shows, there are now hub nodes in the network which connects different networks. If one of these nodes go down, e.g. the center hub node of Figure 3, some parts of the network are suddenly separated. This makes the Hybrid peer-to-peer network more vulnerable to attacks or failure.

3 Introduction to BitTorrent

BitTorrent is a technology/protocol which makes the distribution of files, especially large files, easier and less bandwidth consuming for the publisher. This is accomplished by utilizing the upload capacity of the peers that are downloading a file. A considerably increase in downloaders will only result in a modest increase in the load on the publisher hosting the file.

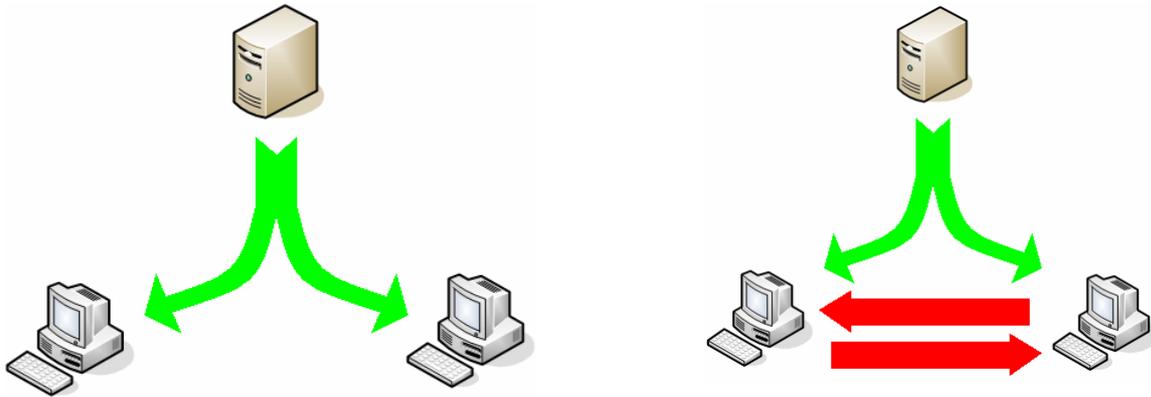


Figure 4 – The basic flow of the BitTorrent protocol.

The illustration in Figure 4 shows the basic flow of BitTorrent. The figure on the left shows a client-server approach to download. The peers download from the server simultaneously. If we assume the upload capacity of the server is the same as the download capacity of a peer, the time for the download to finish will be two times the time if only one peer were downloading from the server. The figure on the right shows an approach similar to BitTorrent. By splitting the file and send one part to each peer, and let the peers download the part they are missing from each other, both download time and load on the server is reduced. Of course, the BitTorrent protocol is much more sophisticated than this simple example, but this shows the basic idea.

3.1 The history of BitTorrent

BitTorrent is by far the most popular peer-to-peer programs ever. Analysis shows that it accounts for about 35% of all Internet traffic [22]. How did it become so popular, and what makes it so special?

By the end of the 90's, Bram Cohen got tired of jumping between dotcoms which never released any product before they went bankrupt. He decided to develop something on his own, and got inspiration from his last job. This company's idea was to keep a file safe and secure by breaking it into small pieces, encrypt the pieces and store them in different locations. Bram realized that the concept of breaking a file into smaller chunks also could be used in file sharing. [20]



In the summer of 2001, Cohen released his first beta version of BitTorrent. In 2002 he presented it at a conference. His goal with this software was to give people a quick and simple way of distributing and swapping Linux software online. But, as we all know, the movie-geeks soon saw the potential in the BitTorrent technology. In 2004, pirate copies of movies and TV-shows began dominating the BitTorrent traffic, and after that the growth has been explosive. It is projected that by the end of this year about 40 million

people have downloaded the BitTorrent application¹. Bram Cohen's only income is voluntary donations from satisfied and kind BitTorrent users. [20]

BitTorrent is completely free and open-source. This is doubtless an accomplice reason for the big success. But the way BitTorrent works also differs considerable from the other peer-to-peer protocols out there.

The problem with many "traditional" peer-to-peer file sharing protocols, in the eyes of Bram Cohen, is that most users have different speeds downlink and uplink. This means that even though a user has plenty of bandwidth downlink, the speed of a file transfer to him will be restricted by a much smaller bandwidth uplink from the user he downloads from [20]. Let us take a simple example: assume that a user wants to fetch a file from another user. They both have the same connection; 2,5Mbit/s downlink and 600kbit/s uplink². What determines the speed of the transfer between the two is thus the uplink of 600kbit/s (i.e. the bottleneck), and the capacity of the downlink is unessential. Most people have an uplink below 1Mbit/s, and sometimes as little as about 10% of the downlink bandwidth. Sure, recently we have seen SHDSL and similar, but the main majority of users have asymmetric rates and this is highly likely to continue in the near future.

This implies that one-to-one file sharing is not the optimal solution. It is, however, the way traditional file sharing protocols (like KaZaa) work. Bram Cohen managed this problem by splitting files into smaller pieces. When requesting a file, the user's computer sniffs around on the internet for people having one or more pieces of the wanted file. He then downloads different parts of the file from different users at the same time and utilizes his downlink bandwidth better. The effect is of course that the file arrives many times faster as compared to downloading from just one user. Sounds simple, doesn't it? Chapter 4 will explain in detail how BitTorrent makes this possible.

3.2 Areas of usage

Piracy and illegal distribution is the first thing brought to mind when you hear about peer-to-peer file sharing. Not unreasonable, since the majority of traffic is illegal exchange of copyrighted material. But the BitTorrent protocol has features which makes it usable for totally legitimate purposes.

It is already possible to download Linux distributions using BitTorrent which enables much faster download than the regular FTP or HTTP can provide. As mentioned earlier, it was also for this purpose Bram Cohen presented his protocol. The internet browser Opera can also be downloaded using BitTorrent³. In addition to increased download speed, it has the benefit that it eases the pressure on the servers when a new version is published. When they released version 8 of their browser their servers went down due to a massive demand [14][15]. Legal usage is growing and many use the protocol to distribute their home-made music or short films, game demos and other legal content [9].

BitTorrent also has potential business usage. Distribution of ISO-images, operating systems, large software and patches can be done at higher speeds using BitTorrent [22]. How often haven't you waited too long for the newest security update from Windows to be finished

¹ These numbers are for the original BitTorrent software made by Cohen. In addition, we have the numerous other clients based on the BitTorrent-protocol which have emerged the last few years. Consequently, the total number is much larger.

² These are the bandwidths for the most popular ADSL-package from NextGenTel [13]

³ They have also incorporated a BitTorrent client in a Technical Preview of the Opera browser released July this year. [15]

downloading? Using the BitTorrent protocol the spreading of this kind of software can be much quicker and more satisfactory for the end users. Within an organization one can also use the protocol to distribute applications and updates more rapidly. Improvements in the protocol facilitate this usage, as explained in section 4.2.

BitTorrent is best suited for new, popular files which many people have interest in. It is easy for a user to find different parts of the file and download them quickly. This can be called the multiplier effect, and a slightly popular show or movie can become insanely popular within days, or maybe within hours. Old or unpopular files will be difficult to find and there will be few users to download from. This makes the protocol effective when dealing with highly demanded files, and obscure files tend not to be available. The reason for this property will appear in section 4.1.

4 The BitTorrent architecture

“I call architecture frozen music...”

- Goethe

This section will explain the standard BitTorrent architecture as described in [4]. Later a new pure peer-to-peer approach will be explained.

The BitTorrent architecture normally consists of the following entities [27]:

- a static metainfo file (a “torrent file”)
- a ‘tracker’
- an original downloader (“seed”)
- the end user downloader (“leecher”)

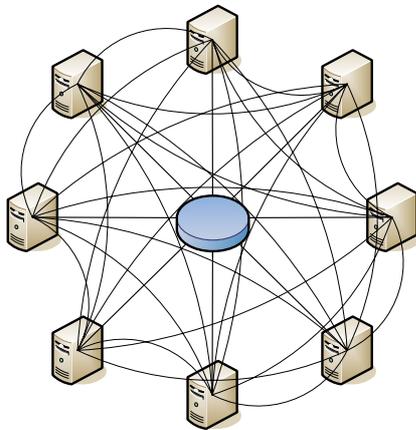


Figure 5 – BitTorrent in its original form matches the “hybrid” peer-to-peer concept. It’s all about the torrent file, the centralized tracker and the associated swarm of peers. The centralized tracker provides the different entities with an address list over the available peers. These peers will then contact each other to download pieces of the file from each other.

The first step in publishing a file using BitTorrent is to create a metainfo file from the file that you want to publish. The metainfo file is called a “torrent”. The torrent file contains the filename, size, hashing information and the URL of the “tracker”. The “torrent” is needed by anyone who wants to download the file the torrent is created from. The torrent file can be distributed by e-mail, IRC, http etc.

The torrent is created by using a free program. This functionality is also commonly included in the BitTorrent clients. To download or “seed” a file, you need a BitTorrent client. The BitTorrent client is a free application that administrates the download procedure. There are several different BitTorrent clients available [28]. They all support the standard BitTorrent protocol, but may differ and be incompatible with each other regarding certain features [29]. A BitTorrent download is started by opening the torrent file in the BitTorrent client.

The tracker keeps a log of peers that are currently downloading a file, and helps them find each other. The tracker is not directly involved in the transfer of data and

does not have a copy of the file. The tracker and the downloading users exchange information using a simple protocol on top of HTTP. First, the user gives information to the tracker about which file it’s downloading, ports it’s listening on etc. The response from the tracker is a list of other users which are downloading the same file and information on how to contact them. This group of peers that all share the same torrent represents a ‘swarm’.

However, making a torrent file is not enough to make the file you want to distribute available. An original downloader known as a “seed” has to be started. A “seed” is a user that has the entire file. A downloading user that has nothing or only parts of a file is called a “leecher”. The “seed” must upload at least one complete copy of the file. Once an entire copy is distributed amongst the other downloaders, the ‘seed’ can stop uploading and the download will still continue for all downloaders as long as there are enough people downloading the file, and all pieces of the file are available. For a popular file one complete copy from the seed

may be enough while for a less popular file, continues uploading by the seed may be necessary. The result is that the bandwidth requirements for the publisher are less if many people are downloading. This is the complete opposite of what would happen if any server-client solution were to be used. The bandwidth requirement of the tracker is also very low even though the number of downloaders is high.

When creating the torrent file from the original file, the original file is cut into smaller pieces, usually 512 KB or 256Kb in size. The SHA-1 hash codes of the pieces are included in the torrent file. The downloaded data are verified by computing the SHA-1 hash code and comparing it to the SHA-1 code of the corresponding piece in the torrent file. This way the data is checked for errors and it guarantees to the users that they are downloading the real thing. Whenever a piece is downloaded and verified, the downloading peer reports to the other peers in the swarm about its new piece. This piece is now available for other peers. We will now go deeper into piece selection and other details of the protocol.

4.1 Algorithms

In many other peer-to-peer file sharing protocols the swapping of files happens one-to-one, meaning that the user himself chooses another peer to download from. As explained earlier, the concept of BitTorrent is to be able to download from many other peers simultaneously. This requires a way of knowing which peers to download what pieces of the file from, with the goal of receiving the complete file as quickly as possible. This needs to happen without the involvement of the end user, and thus needs to be self-configuring.

Choosing peers to connect to is a two-sided problem. First, we need a way of finding the best sequence of downloading the pieces. This is determined by the piece selection algorithm. But this is not enough. A peer, who has the piece you want, might not let you download it. Strategies for peers not allowing other peers to download from them is known as choking, and concerns resource allocation. These two concepts are examined in the following pages.

4.1.1 The Piece Selection Algorithm

How BitTorrent selects what pieces of the file to download have great impacts on the performance of the protocol. It is important to be smart when selecting pieces, in order to not end up a situation where every peer has all the pieces that are currently available and none of the missing ones.

The goal is to replicate different pieces on different peers as soon as possible. This will increase the download speed, and also make sure that all pieces of a file is somewhere in the network if the seeder leaves. We will go through several policies which combined make up the piece selection algorithm.

4.1.1.1 Sub-pieces

BitTorrent uses TCP and it is thus crucial to always transfer data or else the transfer rate will drop because of the slow start mechanism. The pieces are further broken into sub-pieces, often about 16kb in size. The protocol makes sure to always have some number of requests (typically five) for a sub-piece pipelined at any time. When a new sub-piece is downloaded, a new request is sent. Sub-pieces can be downloaded from different peers. [4]

4.1.1.2 Policy #1: Strict Policy

Once a sub-piece has been requested, the remaining sub-pieces for that particular piece are requested before sub-pieces from any other piece. This helps us getting a complete piece as quickly as possible. [4]

4.1.1.3 Policy #2: Rarest First

The main policy in BitTorrent is that of “rarest first”. This means that when a peer selects the next piece to download, it selects the piece which the fewest of their peers have. There are several reasons for this being a good policy [4][12]:

- **Spreading the seed:** Rarest first makes sure that only “new” pieces are downloaded from the seed. In the beginning, the seed will be a bottleneck since it is the only one with any piece of the file. A downloader can see what pieces their peers have, and the “rarest first”-policy will result in that the pieces fetched from the seed are pieces which have not already been uploaded by others.
- **Increased download speed:** The more peers that have the piece, the faster the download can happen, as it is possible to download sub-pieces from different places. We want to replicate rare pieces so they can be downloaded faster.
- **Enabling uploading:** A rare piece is most wanted by other peers, and by getting a rare piece others will be interested in uploading from you.
- **Most common last:** It is sensible to leave the most common pieces to the end of the download. Since many peers have it, the probability of being able to download them later is much larger than that of the rare pieces.
- **Prevent rarest piece missing:** When the seed is taken down, it is important that all the different pieces of the file are distributed somewhere among the remaining peers. By replicating the rarest pieces first, we reduce the risk of missing one or more pieces of a file when the seeder leaves.

4.1.1.4 Policy #3: Random First Piece

Once you start downloading, you don’t have anything to upload. It is important to get the first piece as fast as possible, and this means that the “rarest first”-policy is not the most efficient. Rare pieces tend to be downloaded slower, because you can download it’s sub-pieces from only one (or maybe a few) other peers. As mentioned earlier, multiple peers with the same piece increase the download speed. The policy is then to select the first piece randomly. When the first piece is complete, we change to “rarest first”. [4]

4.1.1.5 Policy #4: Endgame mode

Sometimes a piece might be downloaded from a peer with a slow transfer rate. This can potentially delay the finishing of a download. To prevent this we have the “endgame mode”. Remember the pipelining principle, which ensures that we always have a number of requests (for sub-pieces) pending, the number often being set to five. When all the sub-pieces a peer lacks are requested, this request is broadcasted to all peers. This helps us to get the last chunk of the file as quickly as possible. Once a sub-piece arrives, we send a cancel-message indicating that we have obtained it and the peers can disregard the request. Some bandwidth is of course wasted by this broadcasting, but in practice this is not very much because of the short periode of the endgame mode. [4]

4.1.2 Resource Allocation

No centralized resource allocation exists in BitTorrent. Every peer is responsible for maximizing its download rate. A peer will, naturally, try to download from whoever they can. To decide which peers to upload to, a peer uses a variant of the “tit-for-tat” algorithm. The “tit-for-tat”-strategy comes from repeated game theory, and is a strategy of cooperation based on reciprocity. The essence is to do onto others as they do onto you [11]:

1. On the first move cooperate.
2. On each succeeding move do what your opponent did the previous move.
3. Be prepared to forgive after carrying out just one act of retaliation (i.e. have a recovery mechanism to ensure eventual cooperation).

4.1.2.1 The Choking Algorithm

Choking is a temporary refusal to upload to another peer, but you can still download from him/her. To cooperate peers allow uploading, and to not cooperate they “choke” the connection to their peers. The principle is to upload to peers who have uploaded to you recently; i.e. “if you scratch my back, I’ll scratch yours”. The goal is to have several bidirectional connections at any time, and achieve “Pareto efficiency”⁴ [4].

So, the big question is how to determine which peers to choke and which to unchoke. A peer always unchokes a fixed number of its peers (the default is four). Deciding which peers to unchoke is determined only by the current download rates. It has been chosen to use a 20-second average to decide this. Due to the usage of TCP it’s not desirable to choke and unchoke too rapidly. Thus, this is calculated every ten seconds. [4]

The result is that any peer will upload to peers which provide the best download rate. The other way around; if your upload rate is high more peers will allow you to download from them. This means that you can get a higher download rate if you have many uploads. This is the most important feature the BitTorrent protocol. It prohibits a large number of “free riders” which are peers who only download and don’t allow uploading. In order for a peer-to-peer-network to be efficient all peers have to contribute to the network. This restriction is not present in most other peer-to-peer protocols and applications, and is one of the reasons why BitTorrent has become so popular.

4.1.2.2 Optimistic unchoking

BitTorrent also allows an additional unchoked peer, where the download rate criterion isn’t used. This is called optimistic unchoking. The reason for this is to see if there are any currently unused connections which might be better than the ones in use. Which peer is the optimistic unchoke is shifted every 30 seconds. This is considered to be enough time for the upload to boost up to full speed and for the download to start and obtain full speed. If this new connection turns out to be better than one of the existing unchoked connections, it will replace it [4]. This is quite similar to stage one of “tit-for-tat”.

4.1.2.3 Anti-snubbing

What happens if a peer suddenly is choked by all peers it was downloading from? We then have to find new peers, but the optimistic unchoking mechanism only checks one unused connection every 30 seconds. To help the download rate recover more rapid, BitTorrent

⁴ An allocation is “Pareto efficient” if there is no other allocation in which some other individual is better off and no individual is worse off [12].

introduces “snubbing”⁵. If a client hasn’t got anything from a particular peer for 60 seconds, it presumes that it has been “snubbed”. Following the mentality of “tit-for-tat” it retaliates and refuses to upload to that peer (except if it becomes an optimistic unchoke). It will then increase the number of optimistic unchokes in order to try to find new connections quicker. [4] [12]

4.1.2.4 Upload only

We see that using the choking algorithm implemented in BitTorrent we favor peers which are kind to us. If I can download fast from them, they are allowed to upload from me. But what if I don’t have any downloads? Then it’s impossible to know which peers to unchoke using the presented choking algorithm. Thus, when a download is completed we use a new choking algorithm which unchokes the peers with the highest upload rate [4]. This ensures that pieces get uploaded faster and that they get replicated faster. Peers with good upload rates are also probably not being served by others. [12]

4.1.3 Concluding remarks

The BitTorrent protocol selects pieces by using the following four simple policies:

- **Policy #1: Strict Policy:** Until a piece is assembled, only download sub-pieces for that piece.
- **Policy #2: Rarest First:** Determine the pieces that are most rare among your peers and download those first.
- **Policy #3: Random First Piece:** Select a random piece of the file and download it.
- **Policy #4: Endgame mode:** When all the sub-pieces that a peer doesn’t have are actively being requested, these are requested from every peer.

These policies ensure that the pieces are replicated, and that every peer has the largest probability of retrieving the complete file, as quickly as possible.

Inspired by “tit-for-tat”, the choking algorithm tries to prohibit “free riders” from destroying the dynamics of the peer-to-peer network. If a peer blocks other peers from uploading, he will soon be choked by them and thus affecting his download rate. The choking algorithm correlates the download rate to the upload rate. Uploading is encouraged and the price in return is a better chance for faster download.

All this functionality is implemented inside the protocol. BitTorrent configures the connections to the peers without the involvement of the user, and is thus self-configuring. Decisions are made inside the client of every peer, based on the list of possible peers and the algorithms presented. The way the protocol configures these connections makes it very dynamic and it “achieves a higher level of robustness and resource utilization than any currently known cooperative technique.” [4].

4.2 Important improvements

4.2.1 Bulk traffic marking

Version 4 of the protocol, released in March this year, had one technical improvement which made it more business-friendly. The BitTorrent traffic is now marked as bulk traffic which makes traffic shaping a lot easier [5].

⁵ “If you snub someone, you insult them by ignoring them or by behaving rudely” [3].

Before this refinement, the large volume of BitTorrent traffic could easily have great negative impacts on real-time traffic such as VoIP. The files transferred are often hundreds of megabytes and for a LAN with many users the total amount of traffic can be a major problem and slow down the network. When the BitTorrent file transfers are marked as bulk, almost any standard traffic shaping tool can be used to manage the network traffic. [22]

4.2.2 Decentralized tracker

With one centralized tracker the BitTorrent network is not very fault tolerant. If the tracker goes down the file will no longer be available, since there are no way the peers could know about each other. One centralized tracker also makes the network vulnerable to denial of service attacks.

In May 2005 a version 4.1 was released by Cohen and the big newsflash was support for a decentralized tracker. In this new version (a beta release), all you have to do is to publish your .torrent-file on a webpage, blog etc, and no tracker specification is necessary. Use of a dedicated tracker is still supported, and the publisher can choose the preferred mode.

The tracker is distributed in the sense that every client or node in the network now acts a lightweight tracker. The solution is based on distributed hash tables (DHTs). This makes it possible to share files with minimal resources, but no guarantees can be made as respect to reliability [6].

In addition to making it easier for a more novice user to share his files through a simple website or a blog, one of the other driving forces for the trackerless mode was economy. A website owner might have to pay (to the web server owner) according to the traffic on his website. The more people who want to download your file, the larger the traffic the website generates the larger the bill will be for the owner. Cohen thinks this is unfair as compared to broadcast by a traditional TV-station: whether five people or five million people receive your broadcast, your cost is the same. By removing the tracker from the web server, the only traffic generated will be the fetching of the .torrent file. The web server (tracker) will no longer be involved in keeping track of where all the users are located, resulting in large bandwidth savings and near-broadcast economies.

The trackerless version is still in its childhood and the majority of BitTorrent file sharing will probably involve a traditional tracker for quite a while. The new solution is very promising and as it matures it will probably become increasingly popular. The next section will explain the fundamentals of DHTs which the distributed tracker is based on.

4.2.2.1 Distributed hash tables (DHTs)

A DHT is a decentralized distributed system [25]. The system consists of a set of participating nodes and a set of keys. The DHT performs the function of a hash table. Key and value pair can be stored and a value can be looked up if the correct key is provided. What separates a DHT from an ordinary hash table is that the storage and lookups are distributed among the nodes (machines) in the network. All nodes are peers that can join and leave the network freely. DHTs makes provable guarantees about performance despite the seemingly chaos created by random joining and leaving peers. Any DHT protocol emphasizes the following characteristics;

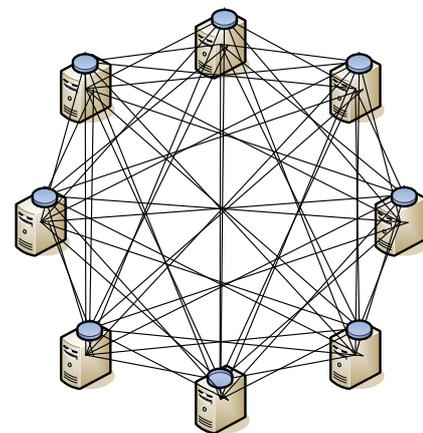


Figure 6 – BitTorrent with a decentralized tracker structure

- **Decentralization:** the system is collectively created and maintained by the nodes without any central coordination.
- **Scalability:** the system performs efficiently even with thousands or millions of nodes.
- **Fault tolerance:** the system should be reliable even with nodes continuously joining, leaving or failing.

The DHT structure consists of two parts, the keyspace partitioning and the overlay network. The keyspace partitioning handles the partitioning of keys among the nodes in the network. The overlay network connects nodes and let them find the owner of a key.

There are several DHT protocols in use [25]. Chord, Pastry, Tapestry and Kademlia are a few of them. These protocols are more similar than they are different. “(.) the various available schemes all fit into a multidimensional matrix. Take one, make a few tweaks and you end up with one of the other ones” [24].

The Kademlia protocol can be seen as an evolved version of Chord. The Kademlia protocol is used in the Azureus and the BitTorrent client. There are also several other network that use these protocol. We will therefore use these two as examples of DHTs in the rest of this paper.

4.2.2.1.1 Keyspace partitioning

Each node in the network is assigned a single key which will be its identifier. To map keys to nodes DHT variants uses consistent hashing [9]. This type of hashing provides hash table functionality without the need to change the mapping of keys if a bucket is added or removed. In most traditional hash tables, a change in the number of buckets causes nearly all keys to be remapped. In consistent hashing the mapping of keys to buckets does not significantly change. This is essential considering the number of nodes who could at anytime be removed or added to the network. A change in ownership of keys would result in bandwidth intensive movement of objects in one node to another. By minimizing this reorganizing of keys, the network can

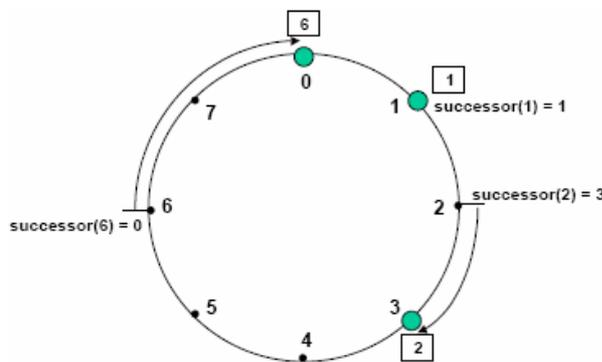


Figure 7 - Example of keyspace partitioning from [19].

The figure shows an identifier circle with $m = 3$. The circle has three nodes: 0, 1, and 3. The successor of identifier 1 is node 1, so key 1 would be located at node 1. Similarly, key 2 would be located at node 3, and key 6 at node 0.

If a node were to join with identifier 7, it would capture the key with identifier 6 from the node with identifier 0.

support a high number of nodes entering and leaving the network.

The DHT protocols Chord [19] and Kademlia [10] uses the same type of consistent hashing. The scalability of consistent hashing is improved by avoiding the requirement that every node know about every other node.

In Chord and Kademlia each node is assigned a unique ID of m bits. Consistent hashing assigns keys to nodes as follows [19]. Identifiers are ordered in an identifier circle modulo 2^m . Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the successor node of key k , denoted by $\text{successor}(k)$. If identifiers are represented as a circle of numbers from 0 to $2^m - 1$, then $\text{successor}(k)$ is the first node clockwise

from k . The keyspace is therefore split into contiguous segments. The endpoints are the node identifiers.

To maintain a consistent hash mapping when a node n joins the network, certain keys previously assigned to n 's successor now become assigned to n . When node leaves the network, all of its assigned keys are reassigned to n 's successor. No other changes in assignment of keys to nodes need occur.

Each node in the network is in it self a standard hash table. To access this table for storing or retrieving of data you need to find the right node in the network and then do normal store or lookup. To determine which node is the appropriate for a specific key the same approach as for finding the next node is used. Generate hash of m bits for the key you want to find. This hash is treated as a node ID and a search for a successor is started. The search can start at any point on the ring progressing clockwise until the node that owns the keyspace is found. This node will be the node which is closest but still is greater than the key. This node will be responsible for storage and lookup of this key. By generating the key using hash the keys are distributed evenly across the nodes in the network.

4.2.2.1.2 Overlay network

The overlay network is maintained by making each node maintain a set of links to its neighbors. The neighbor nodes are picked in a structural way that ensures that the hop number of a route and neighbor nodes per node is low. The set of links maintained by a node also has the property that for any key K , the node owns the K or has a link to a node that is closer to K .

By closer, we mean closer in the sense of distance the between node in the virtual overlay network. The distance metric may differ in the different DHT protocols. In Chord, the distance is defined as the distance between two keys on the circle. The distance traveling around the circle from a key k_1 to a key k_2 is $\delta(k_1, k_2)$. Kademlia uses another approach explained below.

4.2.2.2 Kademlia

Kademlia [10] is a commonly used overlay network protocol for decentralized peer to peer computer networks. It provides several useful features simultaneously unlike previous peer-to-peer system. In Kademlia the number of configuration messages between nodes for node detection is minimized. Key lookups spread this information automatically. The knowledge in a node is enough to make the nodes able to route queries through low latency paths. The queries are parallel and asynchronous. This to avoid timeout delays from failed nodes. The nodes in the network communicate by using the UDP protocol.

Each node in the network has its own ID. This 160 bit identifier is created when the node joins the system. The identifier is created in the same way as in Chord [see section 4.2.2.1.1]. Each message a node sends out contains the ID. This allows receiving nodes to record the sender's existence. This contact information is stored in a triple that contains IP address, UDP port and node ID. For each $0 < i < 160$ the node keeps a list for nodes of distance between 2^i and 2^{i+1} from itself. These lists are called k-buckets.

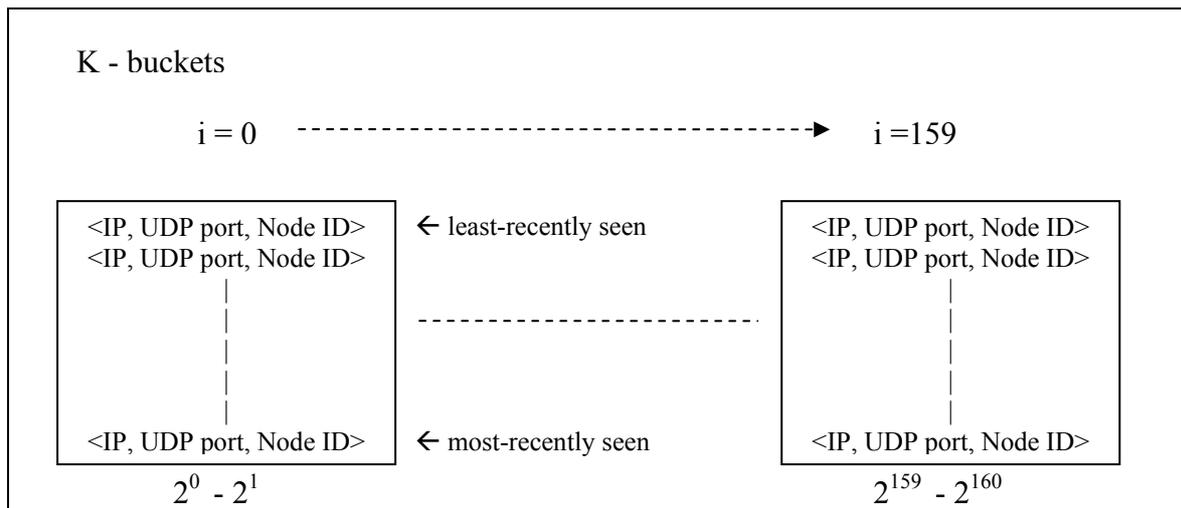


Figure 8: K-buckets

The keys stored in the nodes are also 160 bit identifier, for instance the SHA-1 hash of some data. The publishing and lookup of key, value pairs relies on a notion of distance between two identifiers. This is where Kademlia differs from Chord. Instead of using the circle distance, the distance between two identifiers is defined as the bitwise exclusive or (XOR) interpreted as an integer. The XOR is symmetric and allows nodes to receive lookup queries from the same distribution of nodes contained in their routing tables. This property makes the system learn useful routing information from the received queries. It also makes the routing more flexible compared to other peer-to-peer systems where routing is more rigid.

When a node receives a message (request or reply) from another node, the k-bucket corresponding to the sender's node ID is updated. K-buckets are implemented in a way that evicts least-recently seen nodes without removing live nodes. Studies have shown that the probability of a node being online in the future increases with the time a node has been online [26]. By keeping the oldest live contacts around, the probability that the nodes contained in the k-buckets will remain online is maximized [10]. The k-buckets also provide resistance to some types of DoS attacks. For instance, if new nodes are created rapidly in order to push out good nodes, it won't be noticeable.

Because of the constant traveling of data or requests through the nodes, buckets will stay fresh. In case of no traffic, a node refreshes its bucket by performing a node search for a random ID in its bucket. This is performed in hourly intervals if no traffic.

The protocol consists of four remote procedure calls (RPCs), PING, STORE, FIND_NODE and FIND_VALUE. PING is used for probing network nodes to check if it's online. STORE instructs a node to store a $\langle \text{key, value} \rangle$ pair in the nodes hash table. FIND_NODE and FIND_VALUE behave in the same way. Both take a 160-bit ID as an argument and the recipient returns $\langle \text{IP address, UDP port, Node ID} \rangle$ triples for the k nodes it knows about closest to the target ID with one exception. If a node receives a FIND_NODE RPC and previously has received a STORE RPC for the key, it only returns the stored value. The most important procedure of a node in a Kademlia network performs is to locate the k closest nodes to a given ID. This procedure is known as node lookup and performed recursively.

Lookup procedure

The initiator starts by picking α nodes from its closest non-empty k-bucket and sends parallel asynchronous FIND_NODE RPCs to the chosen nodes.

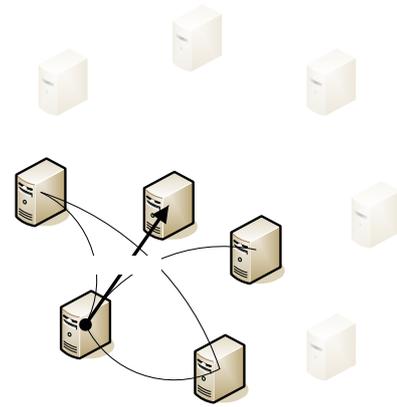


Figure 9

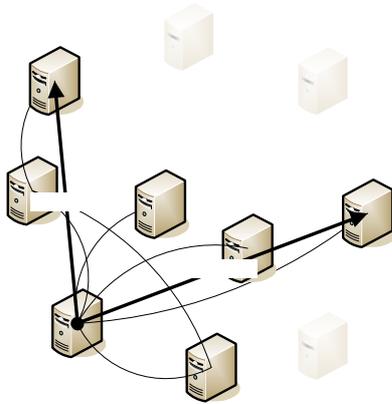


Figure 10

The initiator resends FIND_NODE to nodes learned about from previous RPCs. This can start before the result of all the α nodes has answered the previous query. Of the k nodes has heard of, α nodes that has not yet been queried is picked. A FIND_NODE RPC is then sent to them. Nodes that do not respond quickly enough are removed from consideration until they do respond. If no node closer than any node already seen is returned from a FIND_NODE round, FIND_NODE is sent to all of the k closest nodes that have not yet been queried. The lookup procedure terminates when the k closest nodes that is seen has been queried and a response is received.

A node joins the Kademlia network by contacting a node that is already in the system. The new node puts the existing nodes ID in the right k-bucket. Then it performs a look up for its own ID and finally refreshes all its k-buckets that are not its closest neighbor. This way it populates its own buckets and lets the other node add this new node to their buckets.

5 The future of BitTorrent

“All hell’s about to break loose (...)”

- Brad Burnham, Union Square Ventures

BitTorrent’s followers are numerous, and it has been stated to be the “(...) indisputable leader in download performance.” [16], as well as providing an “(...) effective foundation for dissemination of files that are multi-gigabyte or larger” [1]. It seems to be feasible to conclude from these statements and this paper’s discussion that BitTorrent represents a technology which is likely to change the landscape of broadcast media and file distribution.

There are already applications which combine BitTorrent with RSS⁶ (Really Simple Syndication) information out there. These applications will as an example allow a user to specify that he or she wants i.e. all new episodes of the television-series “Desperate Housewife’s”. This is done through utilizing the metadata which is provided in the RSS files. In other words; BitTorrent has the capability of “(...) transforming the Internet into the world’s largest TiVo⁷[21]” [20].

This, of course, has caused some disgruntle from the television and movie industry. Since the material in question is copyrighted, the providers of the RSS information and the hosts of the BitTorrent tracker are in infringement of this copyright by publishing the information. These content providers has already launched attacks on BitTorrent’s biggest weakness; the centralized tracker ([2]). They have also announced law suits against the creator of BitTorrent, Bram Cohen. According to legal experts ([18][8]) though, these law suits will most likely not hurt BitTorrent it self.

BitTorrent is a cheap and reliable alternative for normal business file distribution. It saves the content publishers considerable money in bandwidth by utilizing their own customers in distributing the companies’ content. Large corporations like Blizzard Entertainment Inc, has already begun utilizing the power of BitTorrent in the distribution of its patches for the online game, The World of Warcraft.

⁶ RSS files contains a publication of BitTorrent files as well as metadata for these files, e.g. the title, genre, key words etc.

⁷ TiVo is a media box which can be attached to a regular tv and will record tv-programs on the basis of user’s wishes. This can be “record all comedy shows starring Jim Carrey” or “record all nature shows about elephants”.

6 Conclusion

In just a few years BitTorrent has become the most popular protocol for peer-to-peer networking. It's simple in use, and has powerful capabilities which enable faster download of files, and more fairness among the peers, than most traditional peer-to-peer protocols. It has proven to be a well-designed and powerful file sharing protocol, based on the extensive adaptation and usage by the Internet users. But, it is by no means optimized, and thus has the potential to only get better as time goes by.

Originally the protocol involved a central tracker in order to let peers find other peers, i.e. a hybrid peer-to-peer protocol. New versions of the protocol has removed this central tracker and distributed this function onto the peers themselves. This improvement has made BitTorrent a pure peer-to-peer protocol.

The protocol is highly self-configuring. Peers find each other using the centralized or distributed tracker, without any user involvement. Peers choose which peers to download pieces of the file from using policies and algorithms embedded in the protocol.

BitTorrent seems to have the potential to revolutionize the landscape of broadcast media and file distribution. The internet can become the world's largest source for Video-on-Demand. We can envision a future where people won't watch entire shows; they'll just download and watch the parts they care about. BitTorrent can also leverage the costs of distributing shows and movies, making broadcasting possible for almost every Internet user. This can have tremendous effects for the large networks, and especially the content providers. The industry needs to change and adopt the new technology before they are over run.

"The cat is out of the bag. The content people have no clue. I mean, no clue."

- Bram Cohen

7 References

- [1] Anthony Bellissimo, Brian N. Levine and Prashant Shenoy. *Exploring the Use of BitTorrent as the Basis for a Large Trace Repository*.
<http://prisms.cs.umass.edu/brian/pubs/bellissimo.bittorrent.pdf>
- [2] John Borland. *BitTorrent file-swapping networks face crisis*. 20.12.2004.
http://news.com.com/BitTorrent+file-swapping+networks+face+crisis/2100-1025_3-5498326.html?tag=nl
- [3] Clue for Windows, version 6.
- [4] Bram Cohen. *Incentives Build Robustness in BitTorrent*. May 22, 2003.
<http://www.bittorrent.com/bittorrentecon.pdf>
- [5] Bram Cohen. *BitTorrent version notes*.
http://www.bittorrent.com/bittorrent_versions.html
- [6] Bram Cohen. *BitTorrent Goes Trackerless: Publishing with BitTorrent gets easier!*
<http://www.bittorrent.com/trackerless.html>
- [7] K. Kant, R. Iyer and V. Tewari. *A framework for classifying peer-to-peer technologies*. Cluster Computing and Grid 2nd IEEE. 2002.
<http://doi.ieeecomputersociety.org/10.1109/CCGRID.2002.1017163>
- [8] Andrew Kantor. *Despite reports, Grokster decision is a win for file sharing*.
http://www.usatoday.com/tech/columnist/andrewkantor/2005-07-01-grokster-decision_x.htm
- [9] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin and Rina Panigrahy. *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*.
<http://portal.acm.org/citation.cfm?doi=258533.258660>
- [10] LegalTorrents.com. <http://www.legaltorrents.com/>
- [11] Petar Maymounkov and David Mazières. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*.
<http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf>
- [12] Chris Meredith. *The story tit-for-tat*. 1998.
<http://www.abc.net.au/science/slab/tittat/story.htm>
- [13] Iqbal Mohamed. *Understanding BitTorrent*.
<http://www.cs.toronto.edu/~walex/mpvc/UnderstandingBitTorrent.ppt>
- [14] NextGenTel. <http://www.nextgentel.no>
- [15] Opera. *Press release: Faster, more efficient downloads in Opera technical preview with BitTorrent*. <http://www.opera.com/pressreleases/en/2005/07/07/>
- [16] Opera. *Torrents*. <http://www.opera.com/download/torrents/>
- [17] J.A. Pouwelse, P. Garbacki, D.H.J. Epema and H.J. Sips. *A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System*. Delft University of Technology.
<http://www.pds.ewi.tudelft.nl/reports/2004/PDS-2004-003/>
- [18] R. Schollmeier. *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*. First International Conference on Peer-to-Peer Computing (P2P'01) IEEE. August 2001.
- [19] Supreme Court of the United States. *Metro-Goldwyn-Mayer Studios Inc. v. Grokster Ltd.* October 2004. <http://www.law.cornell.edu/supct/html/04-480.ZS.html>
- [20] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. MIT Laboratory for Computer Science.
http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

- [21] Clive Thompson. *The BitTorrent Effect*. WIRED Magazine, issue 13.01. January 2005. <http://www.wired.com/wired/archive/13.01/bittorrent.html>
- [22] TiVo. <http://www.tivo.com/>
- [23] Steven J. Vaughan-Nichols. *There Is No Conspiracy Against BitTorrent*. June 23, 2005. <http://www.eweek.com/article2/0,1759,1831018,00.asp>
- [24] Whatis.com. <http://www.whatis.com>
- [25] Brandon Wiley. *Distributed Hash Tables, Part I*. 01.10.2003. <http://linuxjournal.com/article/6797>
- [26] Wikipedia, Distributed hash tables. http://en.wikipedia.org/wiki/Distributed_hash_tables
- [27] Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. Technical Report UW-CSE-01-06-02, University of Washington, Department of Computer Science and Engineering, July 2001.
- [28] Documentation: Protocol <http://www.bittorrent.com/protocol.html>
- [29] Wikipedia, List of BitTorrent clients http://en.wikipedia.org/wiki/List_of_BitTorrent_clients
- [30] Wikipedia, Comparison of applications supporting BitTorrent, http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_clients