Beyond Music File Sharing: A Technical Introduction to P2P Networks

Christian Cikryt

Fachbereich für Informatik, Freie Universität Berlin

Abstract. Peer-to-peer systems have largely replaced the client-server architecture. There are different solutions for the arising problems of efficient object location, scalability, and robustness, resulting in different approaches. We give a short overview of the concepts of structured and unstructured peer-to-peer networks and discuss the handling of those problems in each case. In particular, the BitTorrent Protocol is described, as well as the impacts of selfish peers, attacks on clients, bandwidth throttling, and server breakdowns.

1 Introduction

The traditional method of transferring data within the Internet has followed the client-server paradigm. That means there is a known content provider, the server, to which the clients connect in order to receive the required data directly from the server. Thus, one needs powerful servers, which lead to high costs, to handle all the incoming requests. With home computers getting faster and high-speed Internet connections getting more and more common, relocating the task of content delivery away from the server seemed to be the next logical step. Furthermore, it is quite inefficient that people intending to share their own content have to upload it to a remote server first. This has become even more relevant, as the number of users publishing instead of purely consuming has been growing consistently. These aspects have lead to the birth of peer-to-peer networks. In fact, it started in May, 1999, with the release of the Napster program [1].

Compared to the client-server architecture, the problem of scalability has shifted from the servers' capacities to the signaling between the participants. Moreover, peer-to-peer networks either still possess a single-point-of-failure or have to decentralise their network management. However, decentralised networks raise new problems, such as the question of how to map the geographical location of the peers to the network structure and how to forward messages quickly in the network. Otherwise they might suffer from inefficient zig-zag routes and high response times.

In this report, the different concepts of the peer-to-peer based sharing of information are introduced. Especially, the handling of the above-mentioned difficulties will be examined. Another main focus lies on the BitTorrent protocol, which is the most used peer-to-peer protocol today [2]. It has been extended by Distributed Hash Tables in order to eliminate the tracker as a single point of failure and bottleneck, and thus to improve the availability. Nowadays this enhancement is so well-deployed that the most used BitTorrent tracker, thepiratebay.org, was shut down because its functionality had been fully replaced by DHT [3]. Furthermore, in BitTorrent the efficiency depends on the amount of bandwidth the participants provide for uploading their files. Consequently, users preventing their clients to upload represent a serious threat to functionality. One reason for such behaviour is that people want to use their bandwidth for other applications. Thus, mechanisms have been implemented in the protocol in order to prevent users from taking content without providing anything to other users, which is called free-riding. Additionally, content owners and Internet service providers object to the sharing of possibly copyright protected material and the increased traffic volume respectively, and are undertaking both legal and technical steps to stop BitTorrent applications. These subjects are also enlightened and taken into consideration while examining the robustness of the BitTorrent protocol.

This document is structured as follows. It begins in section 2 with an introduction to the different concepts of peer-to-peer systems starting with unstructured networks and concluding with structured ones. In section 3, the BitTorrent as an example for a centralised peer-to-peer network is described in more detail as well as possible threats to BitTorrent's functionality and countermeasures. Last, section 4 contains a comprehensive discussion and conclusion.

2 Overview of peer-to-peer networks

In contrast to the client-server paradigm, where the content and services are provided by central servers, in peer-to-peer networks these resources are located at the end systems. However, there may be still a central entity to manage the communication between the peers, for example. In peer-to-peer systems, the network organisation is separated from the put-get-strike on the network layer level. Routing and maintenance messages are sent in the so called overlay network, whereas the actual content is typically shared via direct end-to-end connections between the peers. As one computer takes on the role of both client and server in peer-to-peer networks, an application running on a machine is called a peer or servent, which is composed by the first syllable of server and the last syllable of client [4].

There are two common metrics for the efficiency of a network structure. Stress is defined per link and counts the number of identical packets that are sent over this link, whereas stretch indicates the ratio of path length in the overlay network to corresponding path length over IP unicast. One aims to keep these values as low as possible.

In general, peer-to-peer networks can be subdivided into unstructured and structured systems.

2.1 Unstructured peer-to-peer networks

In unstructured networks, a peer providing content and its corresponding IP address are neither related by any structure nor mapped to each other with an algorithm [4]. This means that the mapping between shared data and its typological location is either stored on a central server or has to be distributed within the network. The first generation of these networks includes centralised and pure peer-to-peer systems, whereas hybrid systems belong to the second generation.

Centralised networks As the term implies, centralised peer-to-peer networks contain a central lookup server, whose main purpose is to store information about the content's location and to hand this information to requesting clients. Only the receiving of files and the storage of content is decentralised. Thus, every client participating in the network communicates with the central element, so that the topology of the overlay network can be described as a star network. Via this direct link the peers inform the server about which files they offer and what content they are looking for. The server stores the provided information and answers search requests with IP addresses and port numbers of the sharing peers given that the content is available. The actual transmission of the data takes place apart from the overlay network, via an end-to-end connection between the peers mostly using the HTTP protocol. The downloading peer establishes this connection with the address data obtained from the lookup server.

It takes only one request message to get the location of the desired content or a reply stating that it could not be found. Consequently, the produced overhead is low compared to decentralised systems, where a request has to be flooded throughout the network at worst, because there is no central information storage. Although the server is not involved in the transfer of the shared content, this centralised concept is not scalable, as the needed computing power of the server still grows proportional to the number of connected peers. The lookup server also represents a single point of failure and a bottleneck, because the functionality of the network depends on its availability, processing power, and bandwidth. The example of Napster makes it clear how easily the network can be brought to a standstill by removing the central element. Napster was the first file sharing service to use a centralised approach and was shut down in July 2001 by court order due to the music industry's accusations of massive copyright violations [1]. Another well-known example is BitTorrent, which will be described in detail in section 3.

Pure peer-to-peer networks In this section, we will describe the basic ideas of pure unstructured peer-to-peer systems at the example of the Gnutella 0.4 protocol [5]. Those networks do not rely on a central element for finding content, but instead achieve this via direct communication between the nodes using a routing overlay.

When a new node is about to become part of the network, it has to send a request to a bootstrap server that knows the IP addresses of active nodes. After that, the client joins the Gnutella network by establishing an average of three TCP-connections to other active members [6]. Three has empirically proven to be an well-balanced number, so that not too many resources are used for the connection management and there are still backup capacities if one of the clients drops out. Alternatively, the application can connect to cached peers' addresses from previous sessions. Of course, a client will also connect to new nodes if an active connection breaks down. These new nodes are discovered by flooding Gnutella PING messages through the network. Flooding in this context is a distribution technique, where a node forwards an incoming message to all its neighbours except for the sender if the time-to-live value is a least one. The time-to-live parameter, which signals the number of network hops a packet is transported at most before being dropped, is used to reduce the traffic and prevent the packets from traveling endlessly in circles. In Gnutella 0.4's standard configuration, this value is set to seven hops. The PING requests are simultaneously used as keep-alive patterns in order to notice when a peer has disconnected. The discovery of other participants and the autonomous connection management make the network self-organising.

An equal distribution of content in Gnutella networks is important, because otherwise it may not be found if the offering peer is too many hops away in the overlay network to be reached with the TTL limited search requests. Another problem is that the geographical and topological proximity of the nodes is not considered in the construction of the overlay network, which may result in high stress and stretch values. In fact, it is quite common that a route crosses the Atlantic a few times [4]. These long routes in combination with the high signaling overhead due to flooding leads to a high traffic volume and noticeable latency.

Hybrid networks Hybrid peer-to-peer systems introduce another hierarchical layer with the objective of reducing the high overload of pure peer-to-peer systems which is caused by routing and maintenance messages. The basic ideas of these networks are described at the example of the enhancements proposed for the Gnutella 0.4 protocol, which are integrated in the Gnutella 0.6 standard [7].

To maintain the advantages of a decentralised and self-organised system and to benefit from the performance of a centralised structure, ultrapeers and leaves have been established to form a hub based network. Every leaf node is directly connected to an ultrapeer, like the peers are linked to the server in a centralised network, whereas the ultrapeers form a pure peer-to-peer Gnutella 0.4 network among each other. Since the protocol is decentralised and self-organising, each peer identifies from firewall setting, uptime, available bandwidth etc. whether it is capable of being an ultrapeer. If a dedicated node actually becomes an ultrapeer depends on the current ratio of the number of leaves to number of ultrapeers.

In order to be able to join the network, clients can obtain the addresses of ultrapeers from a bootstrap server. A client asks its ultrapeer about the content's location. In turn, the ultrapeer searches its routing table and forwards the requests to neighbouring ultrapeers. Search requests are only flooded between the ultrapeers, thus reducing the signaling traffic. An ultrapeer only forwards a query to a leaf that is supposed to have the requested content. Consequently, each leaf node has to inform its ultrapeer about what content it shares when joining the network. The ultrapeer stores that data in its routing table, so that it is able to answer requests by giving the corresponding IP address and port. Again, the data transfer itself takes place via an end-to-end HTTP connection.

Although an ultrapeer can only manage a certain number of leaf nodes, depending on its processing power and available bandwidth, this approach is still scalable, as new ultrapeers can be added to the network dynamically [4]. Gnutella 0.6 reduces signaling traffic, but does not take geographical information into account, which may also lead to zig-zag routes and long delays.

2.2 Structured networks

All of the peer-to-peer systems presented in section 2.1 either rely on a central entity and hence contain a single-point-of failure and a bottleneck, or organise themselves by flooding state maintenance messages throughout the network, which causes a huge overhead. Structured peer-to-peer networks, on the other hand, offer a self-organising basic structure for large scale networks without any central entity and without flooding. They allow for locating objects in a small, stochastically bounded number of hops within the network, while requiring only few routing entries per node. The most common systems using this concept for routing are Chord [8], CAN [9], Tapestry [10], and Pastry [11]. The structured overlay networks can be used as a foundation to build services such as scalable multi- or anycast and decentralised object location, e.g. Distributed Hash Tables [12]. The following paragraph describes basic ideas of structured peer-to-peer systems.

A participant in the overlay is called node, where one or many nodes may be hosted at the same IP address. Each node is assigned a uniform random node ID, which is taken from a large identifier space. Application specific objects, like the shared files, are also assigned unique identifiers called keys from the same identifier space using a defined hashing algorithm. As a result, both content and nodes are mapped into the same identifying key space. The hash function maps the content's bytes to a small key that is supposed to identify the content clearly. Which hashing algorithm and corresponding identifier space is used depends on the peer-to-peer protocol. For example, CAN uses a d-dimensional Cartesian identifier space with 128-bit node IDs. Each key is dynamically mapped to a unique active node. This node is then called the key's root. How the keys are mapped to nodes is defined differently in each system. Chord applies a function that maps to the node with the closest node ID clockwise from the key, whereas Tapestry maps to the node whose ID has the longest prefix match. In CAN neighbouring nodes agree on a partition of the identifier space surrounding their node IDs and thus keys are mapped to the node responsible for the related space. A problem that arises from this structure is that the joining and leaving of nodes causes additional overhead due to remapping of keys to peers [12].

In order to route messages efficiently to the target, each node maintains a routing table consisting of the node IDs and IP addresses of close-by nodes in the overlay. When a new node joins the network, it connects to a predefined number of closest neighbours according to their keys. Consequently, the network sustains the following property: For any given object key, a node knows either the location of the content or a node that possesses that information with the highest probability. Messages are forwarded in the overlay to the nodes whose node IDs are stepwise closer in the identifier space to the key.

Structured networks are highly scalable, as the number of exchanged messages to locate content grows logarithmically with the number of participants [12]. Section 3.2 presents a widely deployed example for a structured peer-to-peer system, namely the DHT enhancement to the BitTorrent protocol, which allows trackerless file sharing.

3 BitTorrent - a centralised P2P protocol

BitTorrent is a centralised peer-to-peer protocol designed in 2001 to support the quick distribution of barely spread content with an equal division of the costs among the participants. In order to achieve this, it enables peers to upload those pieces of files that they have already downloaded. This means peers that have just begun downloading can contribute to the network's overall performance and that the load is spread over multiple computers and connections. Furthermore, each peer can upload and download from many links with several other peers at once. Consequently, the speed of the file distribution grows with the number of participants within certain boundaries. Before we describe the behaviour of the BitTorrent protocol in detail in the next section, we have to introduce a few terms first. In BitTorrent, uploading peers which are in possession of the complete content are called seeders, whereas all the other participants are named leechers. One or multiple shared files that are referenced by a metainfo file are called a torrent, and the set of active peers distributing the same torrent is called a swarm.

3.1 Characteristics

A classic BitTorrent network consists of at least one central entity called the tracker, an original downloader, that is in possession of the complete shared content, and clients requesting the content. To provide all the information to the BitTorrent clients that is necessary to begin downloading, the content deliverer generates a metainfo file with the ending .torrent, which is normally stored on a webserver. These metainfo files are bencoded dictionaries [13].

Bencoding Bencoding is a simple encoding that is unaffected by byte order, because numbers are encoded in decimal notation, which is important for cross platform applications like BitTorrent. It supports the four different data types strings, integers, lists, and dictionaries. Strings are prefixed with their length in base ten followed by a colon and the actual string, e.g. 'ham' becomes 3:ham. Integers begin with an 'i' followed by the number in base ten, and end with an 'e'. 3 is represented by i3e, for example. A list is encoded as an 'l' followed by its bencoded elements, and concludes with an 'e'. l4:spam4:eggse corresponds to the list ['spam', 'eggs']. Encoded dictionaries start with a 'd' followed by a list of keys and the corresponding values followed by an 'e', which are bencoded as well. The keys have to be strings and must be sorted before they are encoded. For example {'cow':'moo', 'spam':'eggs'} becomes d3:cow3:moo4:spam4:eggse.

Metainfo file Before sharing, the content is partitioned in equally sized pieces (only the last one may be truncated) in order to support the simultaneous download from multiple peers and the upload of partial available files. After that, SHA1 [14] hash values for each piece are generated, so that clients can check the integrity of the parts. All this information is saved in the .torrent file, whose structure is described in the following paragraph.

The metainfo file contains a bencoded dictionary with the two keys, *announce* and *info*, which are bencoded as well. The *announce* key contains the URL of the BitTorrent tracker, while the *info* key maps to another dictionary. The keys in that dictionary are *piece length*, which indicates the length of the pieces the content is split into in bytes, and *pieces*, which maps to a string containing the concatenated SHA1 hashes of each piece. Depending on whether a single file or multiple files are shared, there is either an additional *length* key or one called *files*. *Length* maps to the length of the single file in bytes, whereas *files* maps to a list of dictionaries, one for each shared file, containing *length* and *path* keys.

Communication with the tracker After a peer has obtained the metainfo file from a webpage, for example, it joins the BitTorrent network by sending a HTTP(S) GET request to the tracker. The base URL for these requests is specified in the .torrent file and mapped to by the *announce* key. Parameters are added to this URL by using standard CGI methods, i. e. appending a '?' followed by parameter=value pairs separated by '&'. The GET requests can contain the following parameters. The *info_hash* key specifies the 20 byte SHA1 hash of the bencoded dictionary at the *info* value from the metainfo file and is used to assure that the clients' metainfo file is uncompromised. Each client generates its own random, 20 characters long ID, when a new download is started. This ID is communicated to the tracker via the *peer_id* key. The *ip* key is an optional parameter that declares the client's IP address and is only necessary if the IP address which the request came from is not the actual public IP address of the client. Otherwise the address can be retrieved from the HTTP request itself. The tracker stores the submitted information together with the client's port, which is specified using the *port* key, in its database. The keys *uploaded*, *downloaded*, and *left* indicate the total number of uploaded, downloaded bytes, and the amount of bytes remaining to be downloaded respectively. As the download might be a resumption, and the downloaded content might at least partially fail an integrity check and therefore needs to be re-downloaded, the last number cannot be computed by subtracting *downloaded* from *length. event* is an optional key which can contain the values *started*, *completed*, or *stopped*. If this key is not initially specified, then it is announced to the tracker at regular intervals. When a download first begins, the value *started* is sent, and when the download is finished, *completed* is used. An announcement containing *stopped* is sent after the peer ceases downloading.

The tracker responds to the GET request by sending a bencoded dictionary with the keys *interval* and *peers. interval* declares the time a client should wait between regular re-requests in seconds, and *peers* maps to a list of dictionaries, each of which contains the keys *id*, *ip*, and *port*, which the peers have sent to the tracker via GET requests. Clients may also re-request that information at nonscheduled times if they need more peers or in case of a certain event.

The peer wire protocol After receiving the location of participating peers from the tracker, the downloading peer establishes direct connections to those clients and starts sharing content via the peer wire protocol, BitTorrent's peer protocol. This protocol relies on TCP and supports symmetrical connections, i.e. messages are alike regardless of the direction which they are sent. It consists of a handshake followed by a stream of messages prefixed with their length. The handshake is required to establish the connection and must be the first message sent by a peer. In general, this handshake has the following structure:

– `19'BitTorrent protocol'00000000<info_hash><peer_id>`

It is 68 bytes long and starts with the number nineteen in decimal notation, which indicates the length of the subsequent string 'BitTorrent protocol'. After that, 8 reserved bytes appear, which are filled with zeros by all current implementations, followed by the *info_hash* and *peer_id* values, which are described in more detail in above section "Communication with the tracker". If the peers do not send the same *info_hash* value, the connection is dropped. The only exception to this rule is if a client wants to transmit multiple downloads over the same port. In this case it may wait for an incoming handshake and responds with the same hash value if it exists in its lists. The received *peer_id* is also checked against the peer list obtained by the tracker. If there is no match, the connection is severed.

All integers in messages after the handshake including the length prefix are encoded as four byte big-endian values. Keep-alives are messages of length zero and are generally sent every two minutes, but timeouts can be done much more quickly when data is expected. Peers may close the connection if they do not receive any message in a defined amount of time.

As stated before, the shared files are partitioned in pieces of equal size, which are referred to by an index starting with zero. As soon as a peer finishes downloading such a piece, it announces that it has this piece to all of its neighbours, provided that the hash value matches. A connection between peers possesses two bits indicating the current state on both ends. The four possible states are interested, not interested, choking, and not choking. Choking is a notification that no data will be transmitted until the state is change to not choking. Data is transferred between the two peers, whenever one side is interested and the other is not choking.

Strategies for downloading Choking is used to improve the download efficiency because the TCP congestion control has a very poor performance when data is send over many connections simultaneously. Furthermore, choking can be applied to get a consistent download rate and to implement a tit-for-tat algorithm, which will be discussed in detail in the section Free-riding. Generally, tit-for-tat is a strategy to reciprocate another's services for mutual benefit. A client applying such a strategy will cooperate at first, then respond depending on its opponent's previous action. If the opponent was previously cooperative, the client is also cooperative and vice versa. A good choking algorithm ought to reduce the number of uploading connections, and try to avoid to choke and unchoke quickly. Additionally, it should reciprocate to peers that provide high download speed, and try out unused connections from time to time, to check whether they are better than the currently used ones, which is called optimistic unchoking. In general, peers download pieces in random order to support high availability of all pieces within the BitTorrent network. Another strategy is to download in rarest-first order. However, randomization among the least common pieces should be added because otherwise many clients would try to receive the least common piece at once, which is counter productive.

3.2 Threats and solutions

Free-riding The functionality of BitTorrent relies on the users providing their bandwidth for uploading. Thus, people configuring their applications to reduce their upload speed threaten the efficiency of BitTorrent. It is also important to offer incentives to prevent peers from stopping their upload after they have finished the download. There are various reasons to cease uploading. Mainly, it slows down one's Internet connection and the user wants to use their bandwidth for other applications. In this section, the impact of selfish or malicious peers to the network is discussed as well as mechanisms in the protocol that encourage seeding.

In BitTorrent each client is responsible for maximizing its own download performance, as there is no central resource allocation. They achieve this by downloading from where possible and using a tit-for-tat algorithm, which is shortly described in section 3.1, to decide how much they upload to which peers. In order to stop sending content to certain clients, they can choke the connection. Of course, this requires a memory of past transactions in the application. However, the resulting overhead is rather small, because a timespan of about half a minute will be sufficient to determine selfish peers. In addition, the optimistic unchoking mentioned in section 3.1 allows finding new, possibly more generous seeders. Thus, selfish peers quickly become isolated and affect the overall performance only insignificantly [15].

It is much more difficult to encourage successful downloaders to keep seeding, because they do not care anymore if their connections get choked. As a consequence, almost all applications save the overall ratio of uploaded to downloaded bytes, and transmit those values to the tracker. Most trackers favour clients with a high ratio, some are even configured in such a way that they only allow clients with a ratio greater than one. However, these measures are easy to bypass, as one can modify a BitTorrent program (a lot of them are free software). Nevertheless, this is a general problem of peer-to-peer systems, because they only work if there are people willing to share their content. People immediately stopping their application after the download has finished do not help the network, but do not prolong the distribution, either. They just leave their share of the load to the other peers.

Throttling Bandwidth throttling is a method to limit the amount of data transmitted by a router. Some Internet service providers apply this technique to reduce the high traffic caused by the BitTorrent protocol [16]. They therefore inspect transferred packets and check which protocol they belong to, which is called deep packet inspection. To prevent a loss of performance due to throttling, the BitTorrent protocol has been extended by the Message Stream Encryption (MSE) [17]. The main goal of MSE is to obfuscate the transmitted content, so that ISPs cannot determine whether the encrypted packets belong to the BitTorrent protocol, and, consequently, cannot throttle them selectively. Secondary goals are to provide some level of authentication and limited protection against man-in-the-middle attacks. Fast cryptographic methods have been chosen in contrast to maximum-security algorithms in order to keep the load on the systems to a minimum.

At the beginning the clients perform a handshake and exchange Diffie-Hellman [18] keys to create the session key used to encrypt subsequent messages. The hash of the *info* value from the metainfo file is used as a weak shared key for authentication, which is known to anyone capable of connecting to the tracker. At least sniffing of the shared key can be averted by using HTTPS for the communication with the tracker. The info hash value is also required to perform successful man-in-the-middle attacks, when the connections are encrypted.

[19] has discovered a significant number of weaknesses in the Message Stream Encryption, that can be exploited to retrieve the encryption keys. However, those attacks can only be applied on specific connections and not to throttle all BitTorrent connections. Nevertheless, they reveal that users cannot rely on MSE in terms of data confidentiality or authentication. In conclusion, MSE prevents throttling by obfuscating the used protocol, unless ISPs apply more evolved and expensive methods for traffic identification, such as analysis of packet sizes, used ports, and time interval between send messages.

Single point of failure As a BitTorrent tracker stores and provides the location of all participating peers in a network, it represents a single point of failure. Consequently, the coordination between the peers will eventually stop working, when the server is down due to technical or legal issues. There are two approaches to improve the availability of the tracking services, which can even be combined. First, multiple trackers can be introduced, and, second, the tracking can be decentralised by using Distributed Hash Tables [20].

In the Multitracker Extension [21] the announce field in the metainfo file maps to a list of lists, each containing tracker URLs instead to a single URL. Trackers in the same list have the purpose of load balancing and constantly synchronise their lists of available peers, whereas trackers in different lists are backup if all the trackers from the previous lists are unreachable. Multiple trackers increase the likelihood of at least one online tracker, while the improvement mainly results from a single highly available tracker. However, they may also significantly reduce the connectivity of the peers in the overlay network, because in reality even the trackers in the same list know a different subset of clients due to joining and disconnecting peers, tracker failure, and the time intervals between tracker updates [22].

The DHT Extension [20] removes the central entity completely and stores the key-value pairs, which consist of the torrent's info hash as key and the corresponding list of IP addresses and ports as values, in a structured peer-to-peer network, as depicted in section 2.2. Each node participating in the network has a unique node ID, which is chosen from the same 160-bit address space as the BitTorrent info hashes. The result of the XOR function interpreted as an unsigned integer is used as a metric to determine the distance between two nodes. When a node wants to know about the peers sharing a specific torrent, it contacts the node from its routing table whose node ID is closest to the torrent's info hash value. The contacted node either replies with the requested peer list or sends the address of a node whose ID is closer to the info hash value. The original node iteratively queries the provided nodes, until it cannot find a node that is closer to the info hash. At the creation of a new torrent the metainfo file is filled with a set of nodes' locations, so that clients are able to initialize their routing table. As a result of the iterative queries, trackerless BitTorrent has a higher response latency for peer requests than a centralised solution, but largely improves the availability of the tracking service.

Attacks on leechers In an effort to stop the distribution of copyright protected material in BitTorrent networks, the film and music industry has filed lawsuits against the providers of tracking services, most recently, in 2009, against thepiratebay.org [23]. However, there too many BitTorrent trackers and metafile hosters scattered around the globe to succeed via legal attacks. Moreover, a lot of clients including the official BitTorrent client now support trackerless filesharing. As a result, the content industry has hired anti-P2P companies to impede the spreading of certain movies or audio tracks. The fake-block attack and the uncooperative-peer attack are most frequently used for this purpose [24].

To understand the fake-block attack, we need to recall that in BitTorrent content is partitioned into fixed-sized pieces, where one piece is typically 256 KBytes. Each piece is further divided into blocks with normally 16 blocks per piece. While downloading a piece, a client usually obtains different blocks from different peers. The goal of the attacker is to prolong the download of a file and to waste the victim peer's bandwidth by sending fake blocks. It joins the swarm by registering to the responsible tracker and then announces that it has a large number of pieces of the content. When a client receives this information, it will send an upload request for a block to the attack peer. But instead of sending an authentic block the attacker provides a fake one. The victim will not notice this until it has downloaded the complete piece from both the attacker and other probably benevolent peers and has performed a hash check. Since the mismatch of hash value indicates only that a least one block is compromised, but not which one, the whole piece has to be downloaded again. Even worse, the client may chooses to request blocks from the attacker because it cannot be identified among the other 15 peers, which will further delay the download. To block the IP addresses of all 16 peers is not viable, because it will quickly result in a self-caused isolation. Furthermore, the attack is efficient, as an attacker can waste 256 KBytes of bandwidth with only transmitting 16 KBytes (if the standard partition size is used).

An attacker performing the uncooperative-peer attack also joins the targeted BitTorrent swarm and establishes TCP connections with many victim peers. However, it never provides any data, neither fake nor authentic. A common version of this attack is the so called chatty peer attack. The attacker exchanges messages in the BitTorrent protocol with each victim peer and initiates the handshake followed by a bitmap advertising that it possesses several pieces ready to upload. Whenever a peer requests any of its blocks, the attacker will not only provide no data, but will also resend the handshake and the announcement messages over and over again. Thus, the attacker persists as a neighbour of its victim, who waste a considerable amount of time dealing with the chatty peer instead of downloading pieces. The effectiveness of this attack increases if a significant fraction of the victim's neighbours are uncooperative. In contrast to the fake-block attack, in this form of attack a malicious peer can be detected more easily and its IP be blacklisted.

IP blacklisting is the only currently deployed countermeasure against both attacks, but it is not sufficient because the attackers mostly use changing addresses, so that some of them are not listed, while other marked IPs may already belong to a benevolent user. The blocking of IP addresses is not part of the protocol, but implemented by many BitTorrent clients. In order to filter out attackers more effectively, sophisticated online algorithms identifying different types of attacks are needed. Nevertheless, measurements have shown that both the fake-block and the chatty-peer attack cannot delay the distribution for more than 50% [24].

4 Discussion and conclusion

All presented concepts of peer-to-peer based networks are applicable to manage the communication between clients in the overlay. In general, one has to choose between a simple centralised approach and a self-organising, decentralised coordination service. First of all, the usage of central servers is not scalable regarding processing power and bandwidth, and introduces a single point of failure and a bottleneck. Second, it does not follow the peer-to-peer paradigm of people sharing their content autonomously, because somebody is required to provide and maintain the server infrastructure. As real deployments demonstrate, this is not a serious problem, because there are companies making such services available and mostly sponsor them by advertisement. Then these companies have to care for the availability, and the users have a very efficient network in terms of stress and stretch, but completely depend on them. Pure unstructured peer-to-peer networks avoid this dependence by flooding messages in the overlay, and thus produce a high signaling overhead. Furthermore, it is possible that available content cannot be found, as the offering and the requesting peer are too many network hops away from each other. However, these decentralised networks are self-organising, scalable, and do not contain a single point of failure. Hybrid systems inherit all the advantages of pure peer-to-peer networks, and add a hierarchical layer to reduce the size of the network, in which state maintenance message are flooded, and, consequently, the induced overhead. Although such networks consist of both centralised and decentralised parts, they are scalable, because the ultrapeers are chosen dynamically depending on the current situation. The costs to construct and maintain a hybrid structure are higher than in pure decentralised networks, but they pay off in sufficiently large networks due to the lower signaling traffic. In contrast to the previous concepts, structured peer-to-peer systems neither contain a dedicated bottleneck nor do they rely on flooding to forward messages. Moreover, they are self-organising, scalable, and resilient, and enable content location in $O(log\, n)$ [12]. Nevertheless, they cannot compete with centralised networks in terms of produced signaling overhead, especially because they need to exchange many more messages in order to sustain their structure.

None of the presented peer-to-peer approaches consider the geographic or typological distance between peers except from Pastry [11]. This has no effect on centralised overlays, but can cause articulate delay due to zig-zag routes in other networks. However, there is a geo-sensitive enhancement to Gnutella [25], which tries to map the structure of physical network to the overlay, for example. This is not an easy task though, because there are two main problems each geo-sensitive approach has to face. On the one hand, a client has to know its geographical position in an easily comparable unit and on the other hand, nearby nodes have to be found at low costs. Both are not that easy to solve, as geographical proximity does not necessarily mean good connectivity. Moreover, the Internet's structure does not perfectly resemble the physical one, so that the topological and the geographic distance may differ [26]. As for the routing, the discovery of nearby peers is expensive, i.e. a lot of messages have to be exchanged. As a result, one has to take care, that the used algorithms do not cause more overhead than the now averted zig-zag routes.

As an example for a semi-centralised system, the BitTorrent protocol has been discussed, which supports a fast content distribution, while the upload is split among all peers, even the ones that only possess parts of the files. Consequently, even clients with low bandwidth, like mobile devices, can contribute without inevitably resulting in a slow connection, because one can download from several sources at once. This feature is also useful to balance the load among computers, when files are distributed to a large number of peers, for example new releases of Linux distributions are shared over BitTorrent. In contrast, the usage of Content Providers such as Akamai costs money, and, more importantly, both the end user and the original distributer depend on their technology. For example, if the Content Provider refuses to support IPv6, the files cannot be shared via this protocol. Even other peer-to-peer systems are less suitable for this task than BitTorrent, because clients first have to download the whole file, before they can support the distribution. BitTorrent is resilient and has been extended to improve availability and performance. For example, it can apply Distributed Hash Tables to form a structured overlay and, hence, eliminate the tracker as a single point of failure. Moreover, MSE can be used to obfuscate BitTorrent's traffic, so that throttling is a least very expensive for Internet Service Provider. Even well-directed attacks on leechers, such as the fake block or chatty peer attack, can only delay the download but not avoid it.

It is worth noting that a tendency back to the traditional client-server architecture can be discerned in last years. Many people prefer uploading their data to so called file hosters, like rapidshare.com or megaupload.com, instead of creating a torrent, for example. The major reason for this is also the main disadvantage of peer-to-peer networks. People have to provide some of their bandwidth and be reachable, which can be quite tedious. File hosting service enable asynchronous upload and download, and are said to more secure concerning prosecution, which is only relevant to the sharing of copyright protected material, of course. Nevertheless, peer-to-peer networks still dominate the Internet traffic statistics and will keep their important position due to the initially mentioned disadvantages of client-server systems. Furthermore, new application areas are tried, e.g. [27] proposes a way to deliver Internet services such as Video-on-Demand based on peer-to-peer concepts. Therefore ISPs could use gateways located at the consumers to reduce energy consumption and consequently costs.

References

1. Eberspächer, J., Schollmeier, R.: Past and Future. In Steinmetz, R., Wehrle, K., eds.: P2P Systems and Applications. Volume 3485 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg (2005) 17–23
2. Ipoque: Internetstudie 2008/2009. `http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009`
3. TPB: Worlds most resiliant tracking. `http://thepiratebay.org/blog/175` (November 2009)
4. Eberspächer, J., Schollmeier, R.: First and Second Generation of Peer-to-Peer Systems. In Steinmetz, R., Wehrle, K., eds.: P2P Systems and Applications. Volume

3485 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg (2005) 35–56

5. The Gnutella Developer Forum: The Annotated Gnutella Protocol Specification v0.4. Annotated Standard 1.6, Gnutella `http://rfc-gnutella.sourceforge.net/developer/stable/index.html`.
6. Schollmeier, R., Hermann, F.: Topology-Analysis of Pure Peer-to-Peer Networks. In Irmscher, K., Fähnrich, K.P., eds.: Fachtagung Kommunikation in Verteilten Systemen (KiVS 2003). Informatik Aktuell, Springer-Verlag Berlin Heidelberg (February 2003) 359–370
7. Klingberg, T., Manfredi, R.: Gnutella 0.6. Rfc, Gnutella (2002) `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`.
8. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: Proceedings of ACM SIGCOMM 2001, ACM New York, NY, USA (2001)
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM New York, NY, USA (2001)
10. Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. ACM SIGCOMM Computer Communication Review **32**(1) (April 2002)
11. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Guerraoui, R., ed.: Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Volume 2218 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg (November 2001) 329–350
12. Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., Stoica, I.: Towards a Common API for Structured Peer-to-Peer Overlays. In Kaashoek, M.F., Stoica, I., eds.: Second International Workshop, IPTPS 2003 Berkeley, CA, USA. Volume 2735 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg (2003)
13. Cohen, B.: The BitTorrent Protocol Specification. Standard 11031, BitTorrent, Inc. (February 2008) `http://www.bittorrent.org/beps/bep_0003.html`.
14. National Institute of Standards and Technology: Secure hash standard. Federal Information Processing Standards Publication 180-1, NSA (April 1995) `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.
15. Liogkas, N., Nelson, R., Kohler, E., Zhang, L.: Exploiting BitTorrent For Fun (But Not Profit). In: Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS '06). (2006) `http://www.iptps.org/papers-2006/Liogkas-BitTorrent06.pdf`.
16. Dischinger, M., Mislove, A., Haeberlen, A., Gummadi, K.: Detecting bittorrent blocking. In: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, ACM New York, NY, USA (2008) 3–8
17. Ludde, uau, The_8472, Parg, Nolar: Message Stream Encryption (aka PHE) format specification . Specification 1.0, Azureus, Inc. (February 2006) `http://www.azureuswiki.com/index.php/Message_Stream_Encryption#Message_Stream_Encryption_.28aka_PHE.29_format_specification`.
18. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6) (1976) 644–654
19. Brumley, B.B., Valkonen, J.: Attacks on Message Stream Encryption. In Nielson, H.R., Probst, C.W., eds.: Proceedings of the 13th Nordic Workshop on Se-

cure IT Systems (NordSec '08). (October 2008) 163–173 http://www.tcs.hut.fi/Publications/javalkon/nordsec08_brumley_valkonen.pdf.

20. Loewenstern, A.: DHT Protocol. BEP – Draft 11031, BitTorrent, Inc. (February 2008) http://www.bittorrent.org/beps/bep_0005.html.
21. Hoffman, J.: Multitracker Metadata Extension. BEP – Draft 10759, BitTorrent, Inc. (February 2008) http://www.bittorrent.org/beps/bep_0012.html.
22. Neglia, G., Reina, G., Zhang, H., Towsley, D., Venkataramani, A., Danaher, J.: Availability in bittorrent systems. In: Proceedings of 26th Annual IEEE Conference on Computer Communications (IEEE INFOCOM 2007), IEEE Press Piscataway, NJ, USA (2007) 2216–2224
23. BBC: Court jails Pirate Bay founders. http://news.bbc.co.uk/2/hi/8003799.stm (2009)
24. Dhungel, P., Wu, D., Schonhorst, B., Ross, K.: A measurement study of attacks on bittorrent leechers. In: Proceedings 7th International Workshop on Peer-to-Peer Systems (IPTPS '08). (2008) http://www.iptps.org/papers-2008/47.pdf.
25. Schollmeier, R., Kunzmann, G.: GnuViz - Mapping the Gnutella Networks to its Geographical Locations. Praxis der Informationsverarbeitung und Kommunikation (PIK) (26) (2003) 74–79
26. Padmanabhan, V., Subramanian, L.: An investigation of geographic mapping techniques for Internet hosts. In: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM New York, NY, USA (2001) 173–185
27. Valancius, V., Laotaris, N., Massoulie, L., Diot, C., Rodriguez, P.: Greening the Internet with Nano Data Centers. In: Proceedings of the 5th international conference on Emerging networking experiments and technologies, ACM New York, NY, USA (December 2009) 37–48