The current state of anonymous file-sharing

Bachelor Thesis

Marc Seeger
Studiengang Medieninformatik
Hochschule der Medien Stuttgart

July 24, 2008

Abstract

This thesis will discuss the current situation of anonymous file-sharing. An overview of the currently most popular file-sharing protocols and their properties concerning anonymity and scalability will be discussed.
The new generation of "designed-for-anonymity" protocols, the patterns behind them and their current implementations will be covered as well as the usual attacks on anonymity in computer networks.

Declaration of originality

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Marc Seeger
Stuttgart, July 24, 2008

Contents

1 Different needs for anonymity

If John Doe hears the word "anonymity", it usually leaves the impression of a "you can't see me!" kind of situation. While this might be the common understanding of the word "anonymity", it has to be redefined when it comes to actions taking place on the internet.
In real life, the amount of work that has to be put into taking fingerprints, footprints, analyzing DNA and other ways of identifying who spend some time in a given location involves a lot more work than the equivalents in the IT world.
It is in the nature of anonymity, that it can be achieved in various places and using various techniques. The one thing they all have in common is that a person tries to hide something from a third party.
In general, and especially in the case of IT-networks, a person usually tries to hide:

- Who they are (Their identity)
- What information they acquire/share (The content they download/upload)
- Who they talk to (The people they communicate with)
- What they do (That they participate at all)

The third party they are trying to hide from differs from country to country and from society to society.
The main reason a person is trying to hide one of the things mentioned above is because the society as a whole, certain groups in society or the local legal system is considering the content or the informations they share/download either amoral or illegal.
Downloading e.g. Cat Stephen's Song "Peace train" could get you in trouble for different reasons.
In Germany or the USA, you'd probably be infringing copyright. In Iran, the ruling government simply wouldn't want you to listen to that kind of music.
There are also people who have the urge to protect their privacy because they don't see the need for any third party to collect any kind of information about them.
Over the past years there have been many attempts to keep third parties from gathering information on digital communication. To allow me to keep this thesis at a certain level of detail, I decided to focus on the attempts that allow the sharing of larger amounts of data in a reasonable and scaleable way. Having to explain all the different methods that allow general purpose anonymous communication would extend the scope way beyond the targeted size of this paper.

2 Basic concept of anonymity in file-sharing

This chapter will give a short overview about the motivation for an anonymous file-sharing network and anonymity in general.

2.1 What is there to know about me?

Especially when it comes to file-sharing, the type of videos you watch, the kind of texts you read and the persons you interact with reveal a lot about you as a person.
As sad as it may seem, but in the times of dragnet investigations and profiling, you may end up on one list or another just because of the movies you rented over the past months. Arvin Nayaranan and Vitaly Shmatikov of the University of Texas in Austin showed in their paper "Robust De-anonymization of Large Sparse Datasets" [1] how a supposedly anonymous dataset, which was released by the online movie-rental-company Netflix, could be connected to real-life persons just by combining data freely available on the internet.
This demonstrates how easy it is to use just fragments of data and connect them to a much bigger piece of knowledge about a person.

2.2 The wrong place at the wrong time

One of the *social* problems with current peer to peer networks is the way in which peer to peer copyright enforcement agencies are able "detect" violation of copyright laws. A recent paper by Michael Piatek, Tadayoshi Kohno and Arvind Krishnamurthy of the University of Washington's department of Computer Science & Engineering titled *Challenges and Directions for Monitoring P2P File Sharing Networks –or– Why My Printer Received a DMCA Takedown Notice*[2] shows how the mere presence on a non-anonymous file-sharing network (in this case: Bittorrent) can lead to accusations of copyright infringement. They even managed to receive DMCA Takedown Notices for the IPs of 3 laser printers and a wireless access point by simple experiments. All of the current popular file-sharing networks are based on the principle of direct connections between peers which makes it easy for any third party to identify the users which are participating in the network on a file-basis.

2.3 Motivation: Freedom of speech / Freedom of the press

The whole situation of free speech and a free press could be summarized by a famous quote of American journalist A. J. Liebling: *Freedom of the press is guaranteed only to those who own one.*
While free speech and a free press are known to be a cornerstone of most modern civilizations (nearly every country in the western hemisphere has freedom of speech/freedom of the press protected by its constitution), recent tendencies like the patriot act in the USA or the pressure on the media in Russia have shown that a lot of societies tend to trade freedom for a (supposed) feeling of security in times of terror or aren't able to defend themselves against their power-hungry leaders. With fear-mongering media and overzealous government agencies all over the world, it might be a good idea to have some form of censorship resistant network which allows publishing and receiving of papers and articles in an anonymous manner and still allow the public to access the information within this network. Moreover, it should be possible to allow sharing of textual content as well as video and audio. This tends to be difficult as file sizes of video and audio content are pretty big when compared to simple textual content.

3 Technical Networking Background

3.1 Namespaces

To really understand the patterns which allow accessing networks in an anonymous way, it is first necessary to understand the basic concepts of networks and the unique characteristics a PC possesses in any given network.
There basically are two things that identify a PC in a network:

- The NICs [1] MAC[2] address (e.g. "00:17:cb:a5:7c:4f")
- The IP address used by the PC to communicate (e.g. "80.131.43.29")

Although a MAC address should be unique to a NIC and an IP addresses should be unique on a network, both of them can be altered. Both of them can be spoofed to imitate other Clients or simply disguise the own identity. Spoofing the MAC address of another client that is connected to the same switch will usually lead to something called

[1]NIC: Network Interface Card
[2]MAC: Media Access Layer

"MAC flapping" and make proper communication impossible.
Also: MAC Addresses are only of concern in the same collision domain. As soon as a Network-Packet is being transfered over a gateway to another network, the MAC Address of the source node will be overwritten by the MAC address of the gateway. As this is the case on the internet, we can mainly concentrate on the IP address as a means to identify a Computer on the internet. There are other means of identifying a Computer within a group of peers, this will be covered in Chapter 3.3, but first, we have to take a look at how the data is being transfered over the Internet.

3.2 Routing

A main factor in the whole concept of anonymity is the way a file takes when traveling through the Internet. There are some patterns that allow an anonymous exchange of files when being on the same physical network, but that is usually not a "real-world" case and will not be discussed in this paper.
When sending a file from one edge of the Internet to another, you usually end up going through several different stations in between. Let's look at the usual "direct way" a TCP/IP packet will go by when traveling from the Stuttgart Media University to the MIT University to give a certain understanding of who is involved in the exchange of a simple IP packet: The tool (traceroute) visualizes the path a packet will usually take to

```
traceroute to mit.edu (18.7.22.69), 30 hops max, 40 byte packets
 1  192.168.1.250 (192.168.1.250)  0.767 ms  0.844 ms  0.944 ms
 2  192.168.100.1 (192.168.100.1)  1.054 ms  1.062 ms  1.049 ms
 3  ciscovlgw318.hdm-stuttgart.de (141.62.31.246)  1.048 ms  1.231 ms  1.688 ms
 4  firewall-h.hdm-stuttgart.de (141.62.1.1)  1.117 ms  1.155 ms  1.530 ms
 5  cisco.hdm-stuttgart.de (141.62.60.254)  1.834 ms  1.832 ms  2.044 ms
 6  Stuttgart1.belwue.de (129.143.101.201)  2.150 ms  1.062 ms  1.380 ms
 7  Stuttgart2.belwue.de (129.143.1.25)  2.194 ms  2.558 ms  2.809 ms
 8  xr-stu1-ge8-3.x-win.dfn.de (188.1.38.53)  2.818 ms  2.975 ms  2.964 ms
 9  xr-fzk1-te2-1.x-win.dfn.de (188.1.145.81)  4.459 ms  4.512 ms  4.564 ms
10  dfn-gw.rt1.fra.de.geant2.net (62.40.124.34)  8.838 ms  8.825 ms  8.825 ms
11  dfn.rt1.fra.de.geant2.net (62.40.124.33)  7.629 ms  7.717 ms  7.640 ms
12  abilene-wash-gw.rt1.fra.de.geant2.net (62.40.125.18)  331.319 ms  335.853 ms  334.037 ms
13  so-0-0-0.0.rtr.newy.net.internet2.edu (64.57.28.10)  114.697 ms  114.695 ms  114.673 ms
14  nox300gw1-Vl-110-NoX-INTERNET2.nox.org (192.5.89.221)  105.856 ms  105.393 ms  105.355 ms
15  nox1sumgw1-Vl-803-NoX.nox.org (192.5.89.237)  105.715 ms  105.452 ms  105.528 ms
16  nox1sumgw1-PEER-NoX-MIT-207-210-143-110.nox.org (207.210.143.110)  106.380 ms  105.793 ms  105.849 ms
17  W92-RTR-1-BACKBONE-2.MIT.EDU (18.168.1.25)  105.920 ms  105.788 ms  105.687 ms
18  WEB.MIT.EDU (18.7.22.69)  105.969 ms  105.947 ms  105.830 ms
```

Figure 1: IP hops from Germany to the US

reach its target. As you can see in the trace log, there are 18 different nodes involved in the transfer of a packet from stuttgart to MIT. While this is necessary to keep the internet stable in the event of outages, it also increases the numbers of possibilities of a

"man in the middle" attack. To keep this routing process scaleable and still hiding the receiver and sender information is one of the biggest tasks to accomplish when designing an anonymous file-sharing protocol.

3.3 Identification throughout the OSI layers

The previously mentioned basic namespaces (IP/MAC-addresses) aren't the only means of identifying a specific PC within a group of others. In the different layers of the Open Systems Interconnection Basic Reference Model (OSI Reference Model or OSI Model for short), you can find several means of identification. Figure 2 shows a small summary of

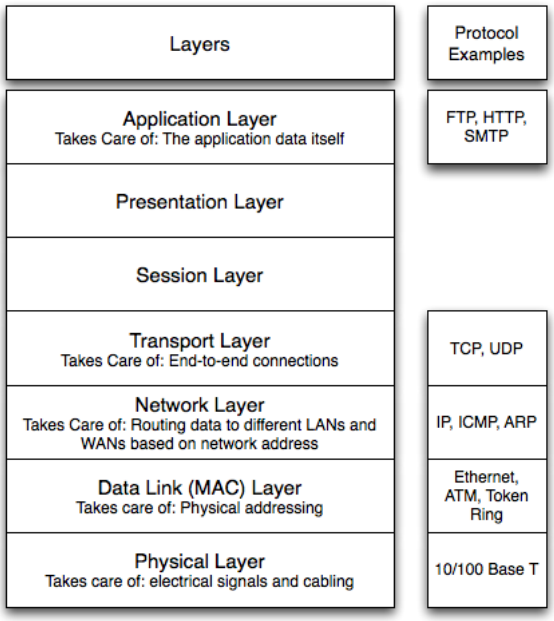

Figure 2: OSI Layer Model

the different Layers of the OSI Model together with a sample of often found protocols residing in these layers. It has to be noted that describing the actual layer concept in connection with implemented systems is not always easy to do as many of the protocols that are in use on the Internet today were designed as part of the TCP/IP model, and may not fit cleanly into the OSI model. As the whole networking community seems to be especially in distress about the tasks taking place inside the Session and Presentation layer and the matching protocols, I decided to treat them as one and only give a short overview of their possible contents in respect to anonymity.

Physical Link Layer: Anonymity in the Physical Link Layer can be undermined by an unauthorized 3rd party tapping into the physical medium itself. In the case of Wireless LAN or any other Radio-Networks, this is a pretty trivial task, tapping into ethernet can be a bit of a challenge and tapping into a fiberglass connection is pretty much impossible due to the precision with which the material has to be handled. The result is always the same, communication can be recorded and analyzed.

Data Link Layer (Ethernet): The Data Link Layer is the first thing when it comes to identifying a computer on a local network. Its frames usually carry the Network interface cards physical (MAC) address of the sender and the destination of a packet.

Network Layer (IP): When it comes to bigger networks, the communication is based on IP addresses which identify each computer and are embedded in the Network packets themselves to allow routing protocols to work.
These IP addresses are managed by central instances. At the moment, there are five organizations giving out IP addresses:

- AfriNIC Africa Region
- APNIC Asia/Pacific Region
- ARIN North America Region
- LACNIC Latin America and some Caribbean Islands
- RIPE NCC Europe, the Middle East, and Central Asia

This centralized IP distribution usually allows anybody to look up the company that acquired a certain address-space. If those companies are handing out IP addresses to people (e.g. your Internet Service Provider), they have to, at least in Germany, keep records which allow to identify who used a given IP address at a given point of time.
The only thing that might keep people from identifying a single computer on a larger private network is the fact, that doing Network Address Translation will only show the IP of the gateway to the outside world.

Transport Layer (TCP): In the Transport Layer, a port is the thing identifying the use of a certain application. While this usually doesn't give a third party that much information about the identity of a Computer as the Network Layer, it enriches this information with the Actions of the machine in question. People connecting

to e.g. Port 21 will most probably try to download or upload a file from or to an FTP Server.

Presentation+Session Layer: These layers might only help identifying a node if encryption based on public key cryptography is used. The certificate exchanged in e.g. TLS could give away a persons Identity, but it will keep the Data in the application layer under protection.

Application Layer (FTP/HTTP): The application layer usually adds the last piece of information to the actions of a node under surveillance. While Layers 1-4 will tell you who is doing something and what this node is doing in general, the application layer will tell what the node is doing in particular. This is where Instant Messengers carry their Text Messages or Mail-Clients transmit login information.

4 Current popular file-sharing-protocols

4.1 A small history of file-sharing

Genealogist Cornelia Floyd Nichols once said "You have to know where you came from to know where you're going". This turns out to be true especially for fast-moving fields such as Computer Science. Before one can talk about the current and future file-sharing protocols, it would be wise to take a look at where file-sharing came from.

4.1.1 The beginning: Server Based

Even before the creation of internet people were trading files using Bulletin Board Systems ("BBS"). You had to dial into a remote machine using telephone-infrastructure. After some time of running on their own, Those remote machines were brought together and synchronized with each other and formed a network called the "FIDO-NET" which was later on replaced by the internet.
In the first years of the Internet and until today, Usenet (based on the NNTP Protocol) allowed people to discuss topics ranging from politics to movies in different "Groups" with random strangers from all over the world. Usenet Providers synchronized each others servers so everybody could talk on the same logical network. Usenet also carries so called "Binary groups" starting with the prefix "alt.binaries." . In those groups, binary files were uploaded as text-messages using BASE64-encoding (the same encoding used

in email attachments until today). This leads to a 33% increase in size and isn't the best way to exchange files, especially since the usability of newsreader applications able to decode binary files never really was something for most endusers. On a smaller scale, servers using the File Transfer Protocol (FTP) are until today one of the main means of file exchange between peers.

4.1.2 The evolution: Centralized Peer-to-Peer

When Napster came up in 1998, it was the birth of the revolutionary idea of Peer-to-Peer (P2P) file-sharing. Napster searched each computer it was started on for MP3 audio files and reported the results to a central server. This central server received search requests by every other Napster application and answered with a list IP addresses of computers which carried the files in question and some metadata (filenames, usernames, ...). Using these IP addresses, the Napster client was able to directly connect to other clients on the network and transfer files.
Although this server-based approach allowed search results to be returned quickly without any overhead, it was what brought napster down in the end as it gave corporations something to sue. Napster had to install filters on its servers which should filter out requests for copyrighted files and lost its user base.
Other protocols fitting in this generation of P2P applications are Fasttrack ("Kazaa" as the usual client) and eDonkey. What separates Fasttrack and eDonkey from Napster is the possibility for everybody to set up a server and propagate its existence to the network.

4.1.3 The current state: Decentralized Peer-to-Peer

Right about the time when the first lawsuits against Napster rolled in, Justin Frankel released the first version of a decentralized P2P application he called *Gnutella*. His employer (AOL) forced him to take down the app a short time later, but it already had spread through the web. Although it only was released as a binary and no source code was available, the protocol used by the application was reverse-engineered and several other compatible clients appeared. Gnutella used a flooding-technique to be able to keep a network connected without the use of a central server (more information in chapter 4.4).
There are basically two ways of searching for data in a decentralized Peer-to-Peer network: flooding or random walk.
The flooding approach uses intermediate nodes to forward request to all of its neighbors

(or a selective amount of neighbors if there is more information available that can help diminish the possible targets of the search request). This approach is a form of breadth-search
The random walk approach is a depth-search based way of only forwarding messages to one neighbour. Most of the time, a node in the network knows its direct neighbors and is able to direct the message according to meta-information to a specific neighbor.
The flooding approach guarantees to be able to find a certain node on a given network, but has to deal with a lot of overhead in the process, the random walk approach can't guarantee that a certain node will be found and expects every node to store information about the surrounding nodes, but also has less overhead.

4.1.4 The new kid on the block: Structured Peer-to-Peer networks

A lot of the recent "big" file-sharing networks were unstructured, meaning that they relied more or less on some form of broadcasting for search-operations because they don't know much about the topology of the network they participate in. Joining an unstructured network means creating an arbitrary connection to other whichever node a client can get information about. In an unstructured P2P network, if a peer wants to find a unique piece of data, the query has to be flooded through the network in an attempt to reach as many peers as possible that might share the data. This can lead to search-queries without any ”hits”, although the data is in fact somewhere in the network. Popular content will show up at more than one peer a search query will result in many ”hits”. Search queries for very rare data on the other hand are unlikely to be able to receive any results.
Structured P2P networks use a protocol that ensures that search requests are automatically routed to a node serving the file that is searched for, even if it is a "one in a million" type of situation. To be able to do this and still scale, you can't go on and have peers randomly connect to other peers and broadcast their searches. A solution to give the network the needed structure lies in the use of a distributed hash table (DHT) that usually uses some form of consistent hashing to assign ownership of a file to a particular peer on the network. This results in every file having a logical position on a node in the network.
Currently there are two ways of implementing this way of content organization:

- structured search

- structured search and content distribution

In the case of e.g. eMule, the hash table-based Kademlia protocol has been adopted by the EMule file-sharing client (more about it in chapter 4.3) and is been used as an overlay protocol to use for **efficient searching**. The download is still being handled by the eDonkey protocol itself. The same goes for Bittorrent with the limitation of not being able to do a text-based search. Bittorrent clients only use the hash table to automatically allocate peers which are also downloading the same file (based on the file's hash value. more about Bittorrent in chapter 4.2).
The other way of implementing the hash table approach is to use the hash table not only for searching, but also for logical **content distribution** within the network.
A model that can be often found is to hash the content a node is sharing and transferring it over to the node with the networks equivalent to the hash of the file. This way, every Download request for a certain file hash can be directed directly to the node closest, according to the networks criteria, to the filehash itself. This means that every node in the network donates a bit of hard disc space and stores data according to the networks criteria. The specifics of each of the networks are different, but the main idea can be easily categorized this way. Examples of the the currently developing structured P2P-file-sharing Networks are *Chord*, *CAN*, *Tapestry* or *Pastry* or the more "public" Wua.la and OFF (details about OFF are discussed in chapter 6.1).

4.1.5 The amount of data

According to the ipoque p2p study 2007 [3], file-sharing applications are responsible for roughly 70% of the total Internet traffic in Germany: The current "big" file-sharing protocols that make up the most traffic on the internet are:

- Bittorrent
- eDonkey
- Gnutella

To understand the problems and weaknesses that allow 3rd parties to receive information on a nodes file transfers, we first have to take a short look at how these protocols handle file search and file distribution.
In the next sections we will take a more detailed look on the currently biggest peer to peer networks: Bittorrent, eDonkey and Gnutella.

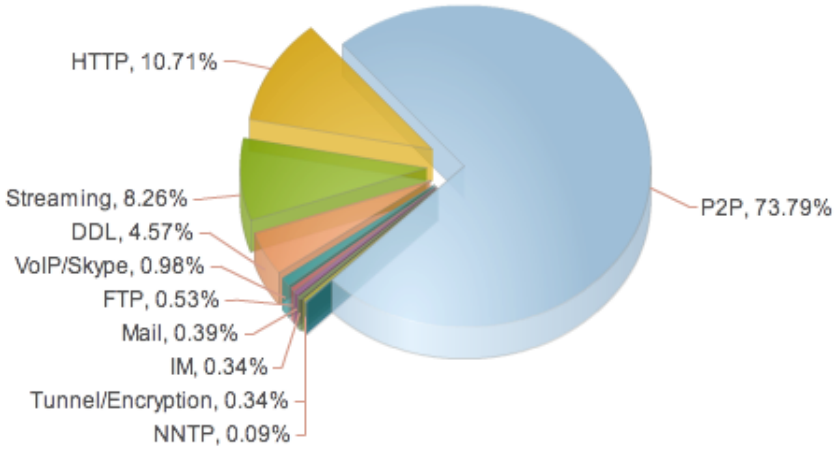

Figure 3: ipoque internet study 2007: total protocol distribution Germany

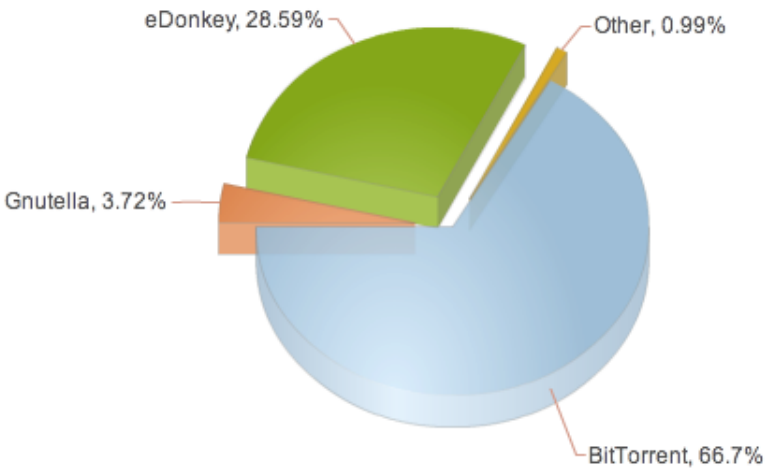

Figure 4: ipoque internet study 2007: p2p protocol distribution Germany

4.2 Bittorrent

The Bittorrent "network" isn't a network in the classical sense. Not every node downloading something in Bittorrent is on the same logical network. As in Bluetooth, there are scatternets being built for every file or group of files being shared. When talking

about a "network" in context with Bittorrent, it relates to the "network" that is being built for a specific torrent file. The protocol has no built-in search possibilities or other bells and whistles, it is a pretty slim protocol and there are 2 or 3 different types of nodes involved in a file transfer on the bittorrent network depending on the way the network is used.

4.2.1 Roles in a Bittorrent network

In a bittorrent network, you can find 3 different roles that clients can take:

- a "seeder": the node serving the file to the network
- a "leecher": a node downloading the file from the network

And usually there also is:

- a "tracker":
  A place where all the information about the participants of the network come together. The tracker is a node serving meta-information about the torrent file. According to the current protocol specs, a tracker can be replaced by a distributed hashtable which is being distributed throughout the users of the different scatternets, given that the clients on the network support it. The major Bittorrent applications and libraries all support the so called ”mainline DHT”.

4.2.2 Creating a network

A file (or a group of files) that is being distributed to the bittorrent network first has to undergo a process to create a matching ".torrent" file. The file(s) is/are being cut into chunks of 128 kb size and those chunks are being hashed using the SHA1 algorithm. The information about the name of the file(s), the amount of chunks, the chunks respective position in the original file(s) and the hash of each chunk is being saved in the torrent file. After the meta-information about the files has been added, an "announce URL" is being embedded in the torrent file. The announce URL contains the IP/DNS of the tracker and a hash identifying the torrent file itself. Usually, the resulting file is being transfered to (and therefor: registered by) the tracker. The seeder now uses a Bittorrent-client to announce to the tracker that it has 100% of the file and can be reached using its IP address

(e.g. 80.121.24.76).

4.2.3 Downloading/Uploading a file

To download this file, a client now has to gain access to the meta-information embedded into the torrent file. Trackers often have a web-interface that provides an HTTP-based download for the registered torrent files. After downloading the torrent file, the client now announces its presence on the file's network (which is identified by the hash of the torrent file) and starts "scraping" (asking) the tracker for the IP addresses of other nodes that also are on the files logical network. The tracker then sends a list of IP+port combinations to the client containing the other peers on the files network. The client starts to connect to the given IP addresses and asks the peers for chunks that itself is missing and sends a list of chunks that it has already acquired. To signal which pieces the client still misses and which ones are available for others to download, the Bittorrent protocol uses a special "bitfield message" with a bitmap-datastructure like this:

0000000000000000 nothing downloaded yet (it's optional to send this)
1000000000000100010 three blocks completed
1111111111111111 successfully downloaded the whole file

This way, every node on the network is able to upload/share what they have acquired so far and download the blocks they still miss at the same time. To distribute the load put on the tracker by every client asking about other peers, some additions like PEX[3] and the DHT[4] were integrated in the widespread clients.

[3]PEX: Peer Exchange
[4]DHT: Distributed Hashtable

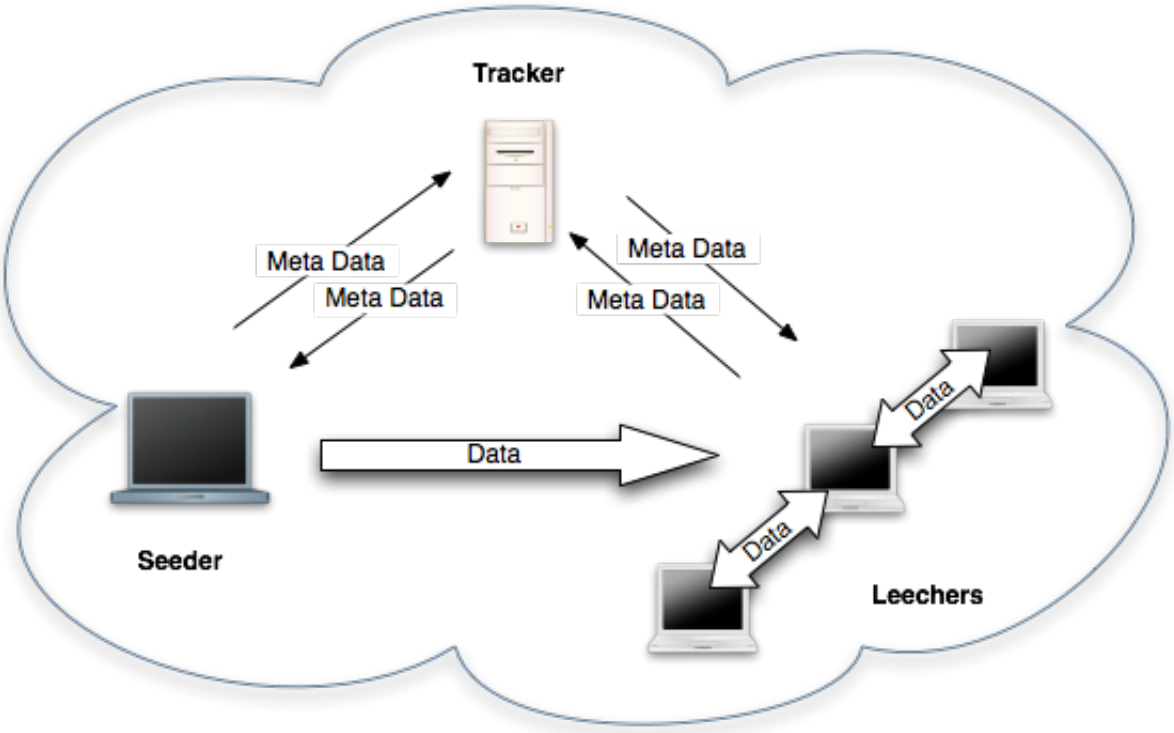

Figure 5: schematic view of a file's bittorrent network

4.2.4 Superseeding

One problem of Bittorrent is, that the original uploader of a given file will show up as the first node having a 100% file availability and therefor losing a bit of anonymity.
To keep nodes from showing up as "the original uploader", a mode called "superseeding" is used, which results in the original client disguising as yet another downloader. This is done is by simply having the uploading node lying about its own statistics and telling other nodes that it's also still missing files. To allow the distribution process to be completed, the client varies the supposedly "missing" pieces over time, making it look as if itself was still in the process of downloading.
It has to be noted that this might only be of help if the tracker itself isn't compromised and being monitored by a third party.

4.3 eDonkey

The eDonkey Network came into existence with the "eDonkey2000" client by a company called Metamachine. After some time there were several open-source Clients that had been created by reverse-engineering the network. Especially a client called "Emule" gained popularity and soon was the major client in the network with a thriving developer community creating modified versions of the main emule client that introduced lots of new features.
The eDonkey network, unlike the Bittorrent network, is a "real" network. It is fully searchable and information about files on the network can therefor be gathered without the need of any metadata besides the files name.

4.3.1 Joining the network

To join the edonkey network, there are 2 basic possibilities:

1. Connecting using an eDonkey Server
2. Connecting using the Kademlia DHT

The concept of eDonkey Servers was the "original" way to connect to the network and already implemented in the original eDonkey2000 client. There are a lot of public servers available which are interconnected and able forward search requests and answers to one another.
The Kademlia DHT was integrated in an application called ”Overnet” by the creators of the original eDonkey2000 client and later on by EMule clients. It has to be noted that although they both used the same DHT algorithm, these networks are incompatible with one another.
On an interesting side note: As of 16 October 2007 the Overnet Protocol was still being used by the Storm botnet for communication between infected machines. For more information on the ”Storm Worm”, I recommend reading *A Multi-perspective Analysis of the Storm (Peacomm) Worm* by Phillip Porras and Hassen Sa¨ıdi and Vinod Yegneswaran of the Computer Science Laboratory [25].

4.3.2 File identification

Files inside the eDonkey network are identified by a so called "ed2k hash" and usually are exchanged as a string looking like this:

`ed2k://|file|<filename>|<filesize>|<filehash>|/`
a real-world example would be:
`ed2k://|file|eMule0.49a.zip|2852296|ED6EC2CA480AFD4694129562D156B563|/`
The *file hash* part of the link is an MD4 root hash of an MD4 hash list, and gives a different result for files over 9,28 MB than simply MD4. If the file size is smaller or equal to 9728000 bytes, a simple MD4 of the file in question is being generated. The algorithm for computing ed2k hash of bigger files than 9728000 bytes is as follows:

1. split the file into blocks (also called *chunks*) of 9728000 bytes (9.28 MB) plus a zero-padded remainder chunk
2. compute MD4 hash digest of each file block separately
3. concatenate the list of block digests into one big byte array
4. compute MD4 of the array created in step 3.

The result of step 4 is called the *ed2k hash*.
It should be noted that it is undefined what to do when a file ends on a full chunk and there is no remainder chunk. The hash of a file which is a multiple of 9728000 bytes in size may be computed one of two ways:

1. using the existing MD4 chunk hashes
2. using the existing MD4 hashes plus the MD4 hash of an all-zeros chunk.

Some programs use one method some the other, and a few use both. This results to the situation where two different ed2k hashes may be produced for the same file in different programs if the file is a multiple of 9728000 bytes in size.
The information about all of this can be found in the emule source (method: CKnownFile::CreateFromFile) and the MD4 section of the aMule wiki [16].

4.3.3 Searching the network

When connecting to a server, the client sends a list of its shared files. Upon receiving this list, the server provides a list of other known servers and the source lists (IP addresses and ports) for the files currently in a downloading state. This provides an initial, automatic search-result for the clients currently active file transfers. When searching for new files, the server checks the search request against his connected client's file lists and returns a list matches and their corresponding ed2k hashes

4.3.4 Downloading

The downloading process itself is based on the Multisource File Transfer Protocol (MFTP) and has slight differences between the different clients on the network. I will not discuss the handshake procedure and other protocol specific details, but rather the things that make the protocol interesting when compared to its competitors.
The protocol has the concept of an upload queue which clients have to wait in when requesting a chunk of data and other transfers are already in progress.
Something else noteworthy is the possibility to share metadata about a given file within the network (such as: this file is good, this file is corrupted, this file isn't what the name may indicate); in this case, the files are identified with their ed2k hash numbers rather than their filenames (which might vary between nodes).
There are several unique forks of the protocol:

eDonkey2000 implemented a feature called "hording" of sources. This allows the user to define one download which has top priority. Several nodes which have set the same file as a top priority one will try to maximize the upload between each other and e.g. skip waiting queues.

eMule uses a credit system to reward users that maintained a positive upload/download ratio towards the client in question. This will shorten the time a client has to wait in the upload queue.

xMule has extended the credit system to help in the transfer of rare files

4.4 Gnutella

The Gnutella Protocol is one of the most widespread Peer to Peer file-sharing protocols. It is implemented in many clients such as Frostwire/Limewire, Bearshare, Shareaza, Phex, MLDonkey and others. The development history is kind of special. It was developed by Nullsoft (the makers of the popular MP3 Player Winamp) and acquisitioned by AOL. After the binary version of the application was available for only one day, AOL stopped the distribution due to legal concerns. These concerns also prohibited the planned release of the source code. Luckily for the open-source community, the binary version of the application had been downloaded thousands of times and allowed the protocol to be reverse engineered.
Gnutella is part of the second big generation of decentralized peer to peer applications. The gnutella network is based on a "Friend of a Friend" principle. This principle is

based on the property of societies, that you don't need to know every node on the whole network to reach a large audience (e.g. when sending out searches for filenames). It's enough to know 5-6 people who themselves know 5-6 other people.
If each node knew 5 contacts that haven't been reachable from the "root node" in the net before, this little sample illustrates how many nodes could be reached by a simple search request that is being relayed be the nodes the request is sent to:

- 1st hop: 5 nodes (the ones you know directly)
- 2nd hop: 25 nodes (the ones that your direct contacts know)
- 3rd hop: 125 nodes
- 4th hop: 625 nodes
- 5th hop: 3125 nodes
- 6th hop: 15625 nodes
- 7th hop: 78125 nodes
- ...

Before the discussion of the "how does my node get on the network" topic, I'd like to provide the "basic vocabulary" of the gnutella protocol as taken from the Gnutella Protocol Specification v0.4[5]:

Ping :

Used to actively discover hosts on the network.
A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.

Pong :

The response to a Ping.
Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.

Query :

The primary mechanism for searching the distributed network.
A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.

QueryHit :

The response to a Query.
This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.

Push :

A mechanism that allows a firewalled servent to contribute file-based data to the network.

4.4.1 Joining the network

Now that the meaning of the packets in the gnutella protocol are understood, we can address the topic of how to gain access to the network.
To gain access to the network, you need to know the IP address of a node already participating in the network. Once you have the IP of a node that is integrated in the network, you have basically two possibilities to get to know other nodes:

Pong-Caching :

"Pong Caching" means to simply ask the node you already know to introduce you to its "friends". This might lead to a segmentation of the net as certain "cliques" tend to form which are highly interconnected but lack links to the "outside" of this group of nodes.

Searching :

Another way of ”getting to know” more nodes is to simply search for a specific item. The search will be relayed by the node you know and the incoming results are containing the information about other nodes on the network, e.g. "JohnDoe" at the IP+Port 192.168.1.9:6347 has a files called "Linux.iso" in his share. Using this information, a network of known nodes that share the same interests (at least the search results reflect these interests by the filenames) is established. As most people tend to share different kinds of media those cliques that were built based upon interests will interconnect or at least "stay close" (hop-wise) to the other

cliques. A little example would be, that a node that has an interest in Linux ISO files and the music of Bob Dylan. This Node could have 2 connections to other nodes that gave an result on the search term "Linux iso" and 3 Nodes that gave an result on the search term "dylan mp3".

The two methods of pong-caching and searching are working fine as long as you already know at least one node in the network. This however leaves us standing there with a chicken or the egg causality. To gain information about other nodes we have to have information about other nodes. As not everybody wants to go through the hassle of exchanging their current IPs with others on IRC/Mail/Instant Message, something called *GWebCaches* have been implemented which automate that process.
That is why the current preferred way to gain information about the FIRST node to connect to if your own "address book of nodes" is empty, is to look up the information in a publicly available "web cache".
A web cache is usually a scripting file on a web server that is publicly available on the internet, which a gnutella clients queries for information. There are GWebcache implementations in PHP, Perl, ASP, C and may other programming languages.
The web cache itself basically does 3 things:

1. Save the IP+Port information about the client asking for other nodes
2. Give out information about the recently seen contacts that also asked for other nodes
3. Give out information about other public web caches

Please note that a web cache should only be used if the initial "address book" of nodes is empty or outdated. otherwise the other two possibilities should be used to gather information about nodes to connect to.
An alternative to the previously discussed ways to connect to nodes is called "UDP Host Caching" It's basically a mix between Pong-Caching and web caches. The communication to an UDP Host Cache relies, as the name suggests, on UDP instead of TCP which keeps the overhead to a minimum. The UDP Host Cache can be either built into a Gnutella Client or it can be installed on a web server.

4.4.2 Scalability problems in the Gnutella Network

The way that searches are being relayed though the network poses the risk of your directly connected nodes acting quite similar to "smurf amplifiers" in a smurf IP Denial-of-Service attack (more information: CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks[5]). In network technology, a smurf attack consists of sending spoofed ICMP echo (aka "ping") packets to a large number of different systems. The resulting answers will be directed to the spoofed IP address and put heavy load on the PC behind that IP. In Gnutella this can happen if a node starts searching for a term that will most likely result in lots of hits (e.g. just the file extension ".mp3").
As a solution to this problem, the equality of all nodes was divided into two classes with the introduction of the (not yet finished) Gnutella Specifications 0.6[6]:
Ultrapeers and **leaves**.
An Ultrapeer is a node with an high-bandwidth connection to the internet and enough resources to be able to cope with. It also acts as a pong cache for other nodes. The "leaves" (which are the "outer" Nodes, similar to the leaves on a tree) are connected to an ultrapeer and the ultrapeers themselves are connected to other ultrapeers. In the Gnutella Client Limewire, there usually there are 30 leaves per Ultrapeer, each leaf usually connected to 3-5 ultrapeers and the ultrapeers themselves maintain 32 intra-ultrapeer connections. Upon connecting, the leaves send the equivalent of a list of shared files (represented by cryptographic hashes of the shared filenames. See the "QRP" chapter) to their ultrapeer. After the exchange of information, search requests that probably won't result in a response usually don't reach the leaves anymore and are instead answered answered by their ultrapeers. Interestingly, the same situation took place in regular computer networks too. Gnutella transformed from a big broadcasting network (Hubs) to a "routed" network in which the ultrapeers act as routers between the network segments. this way, broadcasting gets limited to a minimum and the overall traffic is reduced. The current state of gnutella is that leaves don't accept Gnutella connections. They search for ultrapeers and initiate connections to them instead.
To sum this up, we can basically say that there are 3 kinds of connections in the Gnutella Network:

1. A leaf connected up to an ultrapeer (3 connection slots)
2. An ultrapeer connected to another ultrapeer (32 connection slots)

[5] `http://www.cert.org/advisories/CA-1998-01.html`

3. An ultrapeer connected down to one of its leaves (30 connection slots)

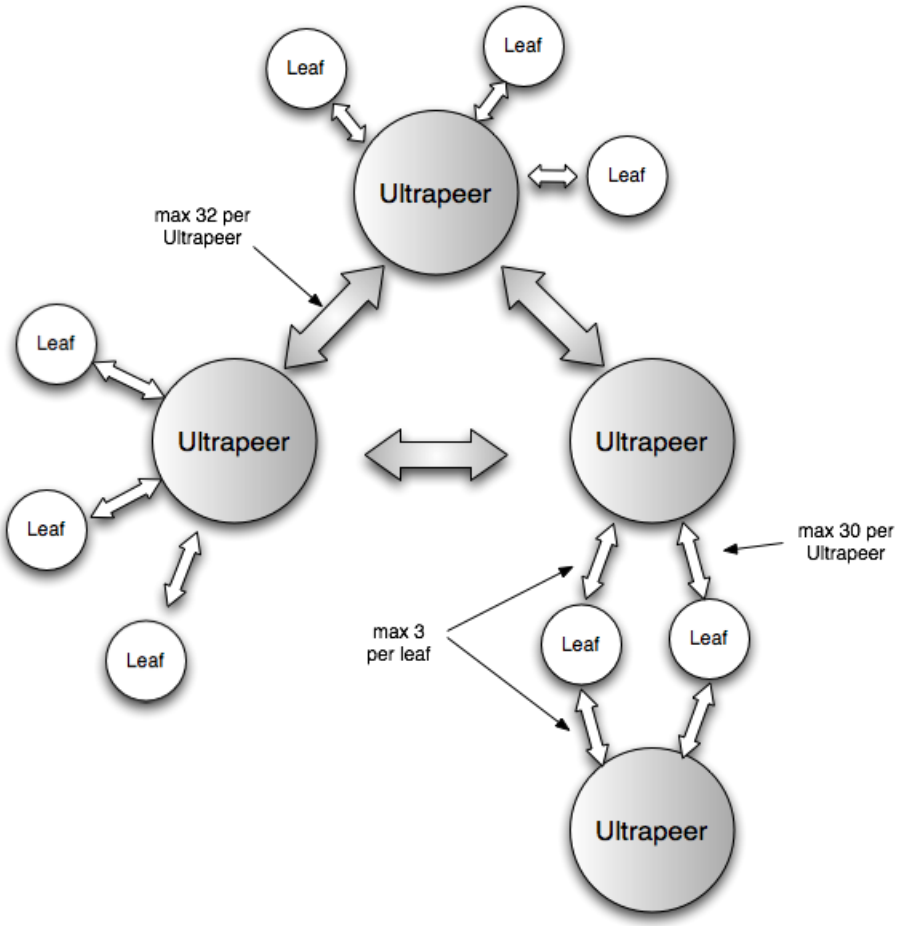

Figure 6: schematic view of the Gnutella network

4.4.3 QRP

As previously mentioned, a leaf is sending a list of its shared files to its ultrapeer. As sending every filename and a matching filehash would take up way too much bandwidth, Leafs create an QRP Table and send it instead. A QRP Table consists of an array of 65,536 bits (-> 8192 byte). Each filename the leaf has in its share is being cryptographically hashed to an 8 byte hash value and the the bits at the according position in the QRP array is being set to 1. After all filenames have been hashed, the resulting 8 kb table is being sent to the corresponding ultrapeers upon connection. The ultrapeers themselves hash the search querries and compare the resulting number with the position

of its leaves QRP Tables. If the bit in the leaves QRP table is set to 0, the search querry won't be forwarded as there won't probably be any result anyway. If the leaf has a 1 at the corresponding position of its QRP Table, the result is forwarded and the information about filename, filesize etc. are transmitted back to the ultrapeer that forwarded the the search query. To keep unnecessary querries between ultrapeers down to a minimum, each ultrapeer XORs the QRP-Tables of each of its leaves and generates a composite table of all the things his leaves are sharing. These composite tables are being exchanged between ultrapeers and help the a ultrapeer to decide for which one of its own ultrapeer connections the search queries won't generate any results anyway. This allows the ultrapeer to direct search queries with a TTL of 1 only to ultrapeers which will answer with a result. Because of the high number of intra-ultrapeer connections (32 in Limewire), each relayed search can produce up to 32 times the bandwidth of the previous hop. A TTL 3 query being sent to one ultrapeer will generate up to 32 query messages in its second hop, and up to 1024 (32*32) in its third hop. The final hop accounts for 97% of the total traffic (up to 32768 messages). This is exactly where ultrapeer QRP comes really into play.

4.4.4 Dynamic Querying

Another thing that is being used in Gnutella to keep the incoming results and network traffic down to a minimum is called "Dynamic Querying". Dynamic querying is a series of searches which are separated by a time interval. As an ultrapeer performing dynamic querying, it will do the first step, wait 2.4 seconds, and then do the second step. If the first or second search gets more than 150 hits, no more searches will be started because 150 hits should be enough and further searches should use more specific search terms to get the amount of results down. If the last search lasts for more than 200 seconds, the ultrapeer will give up and stop.

The first step:

> The first search is being directed only the leaves directly connected the the ultrapeer that received the search query (–> TTL 1: only one hop). The ultrapeer searches the QRP Tables and forwards the search to the leaves that are likely to have an result. If less than 150 results are returned, the second step is being started. If more than 150 results are returned, the search has officially ended and no more "deeper" searches will be started.

The second step:

This step is being called the "probe query". It's a broad search of the top layers of the network and doesn't put too much stress on it. If the number of hits returned is above 150, no more further searches will be started. If the number of results is below 150, it is used to decide how "deep" future searches will penetrate the gnutella network.
At first, a TTL 1 message is sent by the ultrapeer to it's connected ultrapeers. This will only search the connected ultrapeers and their nodes, but not forward the message to other ultrapeers than the directly connected ones. If a TTL1 search is being blocked by all of the connected ultrapeers QRP tables, the TTL will be raised to 2 for some of the ultrapeers (meaning that the search will be forwarded by the directly connected ultrapeer to another ultrapeer once and end there).
If after the "broad" search less then 150 results were returned, a search with a TTL of 2 or 3 is being sent to a single ultrapeer each 2.4 seconds. For a very rare search, the queries all have TTLs of 3 and are only 0.4 seconds apart. The time between searches and the depth has been decided by the results the probe query got.
According to the developers of Limewire, "a modern Gnutella program should never make a query message with a TTL bigger than 3."

If you feel like implementing your own client by now, you should take a look at the excellent *Gnutella for Users* [7] Documentation and the Limewire Documentation concerning *Dynamic Querrying* [8]

4.5 Anonymity in the current file-sharing protocols

4.5.1 Bittorrent

At first, the Bittorrent-protocol wasn't designed to have any anonymizing functions, but there are some properties and widespread additions that might add a certain layer of anonymity.
The first thing that can be identified as "anonymizing" is the way a person joins a Bittorrent network. To gain knowledge about the people participating in the distribution of a certain file, you first have to have the information about the network itself. This information is usually embedded within the torrent file itself in the form of the announce URL and identifying hash. This information can also be received by using PEX or the mainline DHT. Both of these additional ways can be disabled by placing a "private" flag

in the torrent file. Marking a torrent file as private should be respected by bittorrent-client-applications and should allow people to create private communities (similar to the Darknet pattern described in chapter 5.1).
Another way of hiding unnecessary information from third parties came up when ISPs [6] started to disturb Bittorrent traffic to keep the bandwidth requirements of their customers down (see: "ISPs who are bad for Bittorrent"[10] over at the Azureus Wiki). This resulted in two new features being implemented in the main clients:

- Lazy Bitfield:
  Wherever Bittorrent connects to another client, it sends a Bitfield Message to signal which parts of the file it has already got and which ones are still missing (compare Chapter 4.2.3). More detail is available in the official Bittorrent Specification 1.0 [11]:

  > bitfield: <len=0001+X><id=5><bitfield>
  > The bitfield message may only be sent immediately after the handshaking sequence is completed, and before any other messages are sent. It is optional, and need not be sent if a client has no pieces.
  > The bitfield message is variable length, where X is the length of the bitfield. The payload is a bitfield representing the pieces that have been successfully downloaded. The high bit in the first byte corresponds to piece index 0. Bits that are cleared indicated a missing piece, and set bits indicate a valid and available piece. Spare bits at the end are set to zero.
  > A bitfield of the wrong length is considered an error. Clients should drop the connection if they receive bitfields that are not of the correct size, or if the bitfield has any of the spare bits set.

  Some ISPs who were especially keen on keeping clients from uploading data (which is potentially more cost intensive for the ISP) installed third party appliances which filtered traffic for "complete bitfields" meaning that the peer has already downloaded the complete file and only wants to upload things from that point on. Upon detection of such a packet, usually forged TCP RST packets are sent by the ISP to both ends of the connection to forcefully close it or the connection was shaped down to a point where no more than 1-2 kb/s throughput are being transfered.

[6] ISP: Internet Service Provider

To keep the ISP from detecting this, Lazy Bitfield was invented. When enabling Lazy Bitfield, the Bittorrent client handshake is altered, so that even if a Bittorrent Client has all available Data already downloaded (–> complete bitfield), the client reports that it is still missing pieces. This will make a BitTorrent client that usually just wants to upload data initially appear to be a downloading peer (also referred to as "a leecher"). Once the handshake is done, the client notifies its peer that it now has the pieces that were originally indicated as missing.

- Message Stream Encryption[12]:
  Here's a little quote taken from the azureus wiki[12], that describes the use of Message Stream Encryption (or if only the header is encrypted but the payload left unchanged: "Protocol Header Encryption") pretty good:

  > The following encapsulation protocol is designed to provide a completely random-looking header and (optionally) payload to avoid passive protocol identification and traffic shaping. When it is used with the stronger encryption mode (RC4) it also provides reasonable security for the encapsulated content against passive eavesdroppers.

  It was developed when some ISP's started traffic-shaping Bittorrent traffic down to a point where no more communication was possible.

  Message Stream Encryption establishes an RC4 encryption key which is derived from a Diffie-Hellman Key Exchange combined with the use of the torrents info hash. Thanks to the Diffie-Hellman Key Exchange, eavesdroppers can't gather important information about the crypto process and the resulting RC4 encryption key. The use of the info hash of the torrent minimizes the risk of a man-in-the-middle-attack. The attacker would previously have to know the file the clients are trying to share in order to establish a connection as a man in the middle.
  This is also emphasized in the specifications[12]:

  > It is also designed to provide limited protection against active MITM attacks and port scanning by requiring a weak shared secret to complete the handshake. You should note that the major design goal was payload and protocol obfuscation, not peer authentication and data integrity verification. Thus it does not offer protection against adversaries which already know the necessary data to establish connections (that is IP/Port/Shared Secret/Payload protocol).

RC4 is used as a stream cipher (mainly because of its performance). To prevent the attack on RC4 demonstrated in 2001 by Fluhrer, Mantin and Shamir[13] the first kilobyte of the RC4 output is discarded.
The "Standard Cryptographic Algorithm Naming" database[14] has a "most conservative" recommendation of dropping the first 3072 bit produced by the RC4 algorithm, but also mentions that dropping 768 bit "appears to be just as reasonable".
According to these recommendations, the 1024 bit dropped in the MSE specification seem just fine.

According to the ipoque internet study 2007[3], the percentage of encrypted bittorrent traffic compared to the total amount of bittorrent traffic averages at about 19%. In

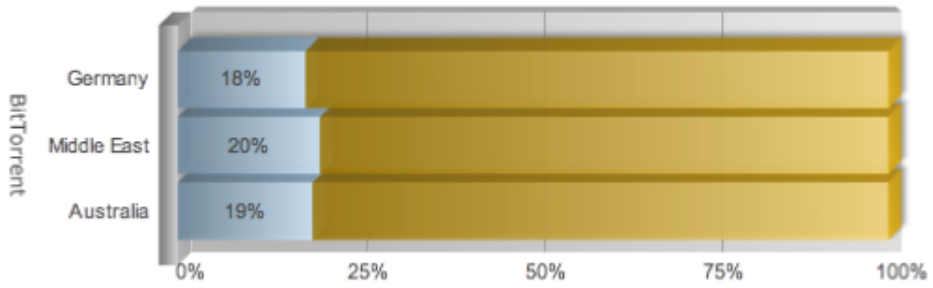

Figure 7: percentage of encrypted Bittorrent P2P Traffic

a press release [15] by Allot Communications dating back to the end of 2006, they state that their NetEnforcer appliance detects and is able to filter encrypted Bittorrent packages.

4.5.2 eDonkey

Emule, the client with the biggest amount of users in the eDonkey network, supports protocol obfuscation similar to the Bittorrent protocol header encryption starting from version 0.47c to keep ISPs from throttling the Emule file-sharing traffic. The keys to encrypt the protocol are generated by using an MD5 hash function with the user hash, a ”magic value” and a random key.
Encrypting the protocol (Handshakes, ...) tries to keep 3rd parties from detecting the usage of Emule. The number of concurrent connections, connections to certain ports

and the amount of traffic generated by them might be a hint, but the protocol itself can't be distinguished from other encrypted connections (VPN, HTTPS...), at least in theory. The only technical documentation about this feature that can be found without hours of research consists of copy & paste source code's comments in the Emule Web Wiki[17]

4.5.3 Gnutella

The way searches are forwarded over multiple hops doesn't allow a single node (besides the connected supernode) to determine who actually sent the search in the beginning. Although this wouldn't keep a 3rd party which acts as an ISP to detect who sends out searches, but it keeps a single malicious attacker from "sniffing around" unless he acts as a supernode.

5 Patterns supporting anonymity

So far, this thesis has discussed the currently "popular" file-sharing networks and their protocol design concerning scalability . It was also pointed out that although these protocols weren't designed with anonymity in mind, they tend to have properties or modes of usage which make them anonymize parts of a user's action in the network.
Now it is time to take a look at network-protocols that were designed with anonymity in mind. This thesis will first take a look at the basic patterns behind nearly all of the currently available implementations of anonymous file-sharing software and then discuss the implementations themselves with a focus on their scalability.

5.1 The Darknet pattern

The darknet pattern targets the social part of the whole anonymous file-sharing problem. As explained in the Chapter "Different needs for anonymity" (1), there are different reasons why someone might want to achieve a certain level of anonymity, but most of them deal with the problem of hiding information about the things you do/say from a third party. This third party most of the time isn't anyone you know personally and tend to socialize with. This fact is exactly where Darknets come to play. A Darknet exists between friends and well-known contacts which share a mutual level of trust.
By creating a darknet (also known as a "friend to friend" (F2F) network), you manually add the nodes your application is allowed to connect and exchange information with. In the end, the "pure" concept resembles an instant messenger with an added layer of file-sharing. Most of the major implementations have an "all or nothing" approach to sharing files, so having private information in your shared folder which is personally directed to only one of your peers usually isn't a good idea.
One of the major problems of darknets is, that they usually tend to model the social contacts of a person into a network. While this is good for protection against certain third parties, the amount of data shared and the number of contacts that are online at a single instance is tiny compared to the popular file-sharing protocols. The reason behind that is the fact that a person can only add a limited number of friends without compromising security by adding peers they don't really know (The usual "seen at a party once" kind of contact). In some implementations (e.g. Alliance P2P[7]), it is possible to allow friends to interconnect even if they might not really know each other.

[7] AllianceP2P: `http://alliancep2p.sf.net`

This might help to solve the problem of having too few nodes to share files with, but it destroys the idea behind darknets as a protection against unknown third parties.
A good overview of darknets can be found in the paper ”How to Disappear Completely: A survey of Private Peer-to-Peer Networks”[20] by Michael Rogers (University College London) and Saleem Bhatti (University of St. Andrews).

5.1.1 Extending Darknets - Turtle Hopping

As previously mentioned in Section 5.1, the main problem with Darknets is, that the number of peers available at a single instance is usually only a fraction of the number of total peers in the users ”contact list”. Different time-zones, different schedules at work/university/school tend to diminish the number of shared files in a persons darknet to the point where it loses its attractiveness to run the software. This problem is being solved by the "Turtle Hopping” pattern, described on turtle4privacy homepage [18]. The ”Turtle Hopping” pattern combines proxy chaining and Darknets by using your direct connections as ”gateways” to connect to your friend’s friends. While this solution poses

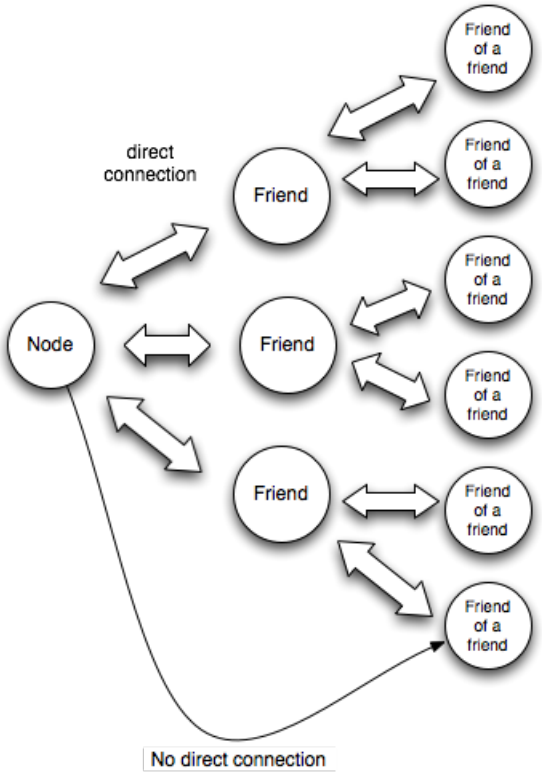

Figure 8: Turtle Hopping

a certain throughput/security tradeoff, it extremely extends the amount and availability of files. Both, search queries and results are only forwarded between trusted peers. The "official" implementation of turtle hopping, which was developed at the Vrije Universiteit in Amsterdam also uses encryption between hops to keep eavesdroppers from listening to the searches.
A great source of further information concerning the darknet pattern is "The Darknet and the Future of Content Distribution" by Microsoft [9]

5.2 The Brightnet pattern

The word Brightnet makes one thing pretty clear: it's the opposite of a Darknet. To freshen up you memories: In the Darknet pattern, you only control WHO you share your data with, but once you decide to share with a peer, you usually don't have any secrets from that node about WHAT you share.
In the Brightnet pattern it is exactly the opposite: you usually don't care that much about who you trade data with, but the data itself is a "secret".
This might seem a little bit confusing, how can data that is openly shared with everyone be a secret?
To understand this, one has to look at the typical bits and bytes that are shared in a file-sharing application: Usually, every piece of data does not only consists of arbitrary numbers, but it has a certain meaning. It is part of the movie "xyz", the song "abc" or the book "123". When uploading even a small portion of this data, you might be infringing copyright or, depending where you live, might get yourself into all sorts of trouble for the content you downloaded. This could lead to the situation that every government agency or corporation could tell if you're sharing/acquiring a file by either simply receiving a request from or sending a request to you (given that you're on an "open" network) or looking at the transfered data on an ISP level.
The main idea behind the brightnet pattern is to strip the meaning from the data you transfer. Without any distinct meaning, the data itself isn't protected by any copyright law or deemed questionable in any way. It would be totally fine to share and download this meaningless data. To end up with "meaningless" data, the current brightnet implementations use a method called "multi-use encoding" as described in the paper "On copyrightable numbers with an application to the Gesetzklageproblem[19] by "Cracker Jack" (The title is a play on Turing's "On computable numbers").
The simplified idea is to rip apart two or more files and use an XOR algorithm to recombine them to a new blob of binary data which, depending on the way you look at

it, has either no specific meaning or any arbitrary meaning you want it to. The whole meaning of the block lies in the way you would have to combine it with other blocks to end up with either one of the "original" files used in the process.
The main idea of the people behind the brightnet ideas is the fact that the current copyright-laws originate from a time where a "digital copy" without any cost wasn't possible and that those laws simply don't reflect the current possibilities. They claim that digital copyright is not mathematically self consistent. Therefore logically/legally speaking the laws are inconsistent as well.

5.2.1 the legal situation of brightnets

In most western societies, the interesting thing is the way copyrighted original blocks would influence the whole situation concerning the possible copyright of the resulting block.
The specific details of this can be found in the great paper by *Hummel* called "Würdigung der verschiedenen Benutzungshandlungen im OFF System Network (ON) aus urheberrechtlicher Sicht" [24].
To summarize it, there are basically 3 different possibilities the resulting block could be treated as according to German law:

a new work As the process of creation was fully automated when generating the resulting block, the new block doesn't qualify as a real "geistige Schöpfung" according to §2 II UrhG in connection with §7 UrhG.

a derivation of the old work In this case, the original copyright holder would still have a copyright claim on the newly generate block.
To be able to argue this case, the operation used to create the resulting block would have to be reversible and a single person would need to be able to claim copyright.
As the resulting and original blocks which are needed to reverse the transformation find themselves on a peer to peer network, the reversibility is only given when all of the nodes carrying the blocks in question are actually on the network.
As the resulting block could contain traces of more than one copyrighted file (in theory: as many as there are blocks in the network), everybody could start claiming copyright on that block.

shared copyright In german law, according to §8 UrhG, a shared copyright only exists when there was a combined creation of the work ("gemeinsame Schöpfung

des Werkes"). As the generation of the block was fully automated, there is no "innovative process" to constitute a shared copyright.

This leads to the conclusion that everyone could legally download arbitrary blocks.

5.3 Proxy chaining

When it comes to the pattern of proxy chaining, we have to look at two things:

- What is a proxy server?
- How does this improve anonymity? (aka: the proxy chaining pattern)
- What does being a proxy mean in the legal sense?

5.3.1 What is a proxy server?

The first question can be easily answered: A proxy server is a computer system or an application, which takes the requests of its clients and forwards these requests to other servers.
The thing differentiating proxies is their visibility. Sometimes a proxy has to be knowingly configured in the client's application. Companies often use SOCKS Proxies to allow clients behind a firewall to access external servers by connecting to a SOCKS proxy server instead.
These server allow companies to filter out questionable content or manage weather a user has the proper authorization to access the external server. If everything that is checked has been deemed fine, the SOCKS Server passes the request on to the target server and. SOCKS can also be used in the other direction, allowing the clients outside the firewall zone to connect to servers inside the firewall zone. Another use of socks proxies can be caching of data to improve latency and keep the outbound network traffic down to a minimum.
Transparent proxies can do pretty much the same thing, the difference to the usual SOCKS proxies is, that the client doesn't need to tell his applications to use any form of proxy because they are built as part of the network architecture. One could compare transparent proxies to a positive version of a "man in the middle".

5.3.2 How can I create a network out of this? (aka: the proxy chaining pattern)

The main feature of a proxy chaining network is, that it is a logical network being built upon the internets TCP/IP infrastructure.

This means that the packets being send into the network don't have specific IP addresses as a source or destination on a logical level. They usually use some sort of generated Identifier (e.g. a random 32 bit string). This can be compared to the way Layer 3 switches /routers manage IP addresses. A switch itself doesn't know where in the world a certain IP address is stationed. The switch itself only knows on which port someone is listening who knows a way to transfer a package to the destination IP. While receiving new packets, the switch makes a list (a so called "routing table") of the different senders IP addresses in conjunction with the MAC Address and the (hardware-) port the packet came from. this way, the switch learns to identify a certain port with a certain MAC/IP address.
When joining a proxy-chaining network, the client tries to connect to a certain number of other clients on the network. These initial connections can be compared to the different ports a switch usually has. After connecting, the client notes the Source IDs of packets it receives from one of those initial connections and generates a routing table. This way, the client begins to draw its own little map of the network which helps forwarding packets to the proper connection.
The way a client in a proxy chaining network reacts when receiving a packet with an unknown target ID can also be compared to the way a switch works. This time it's comparable to an unknown target MAC address. In case a switch doesn't know a target MAC address, it simply floods the ethernet-frame out of every port. This way, the packet should arrive at the destination if one of the machines behind one of the (hardware-) ports knows a way leading to the destination address. To keep packets from floating though the network forever, some sort of "hop-to-live/time-to-live" counter is usually introduced which gets decreased by 1 every time someone forwards the packet. If there is no reasonable short way to get the packet to its destination.

5.3.3 How does this improve anonymity?

As there is no central registration that gives out the IDs used to identify nodes on the logical network, it is virtually impossible to gather knowledge about the identity of a person operating a node by looking at the ID of e.g. a search request. In current networks such as Gnutella (Chapter 4.4) or eDonkey (Chapter 4.3), the identity of a client is based on its IP address. As IP addresses are managed globally, an IP can be tracked down to the company issuing it (in most cases: you internet Service Provider) and, at least in germany, they are supposed to keep records when a customer was issued

an IP.
The only thing that can be said with certainty is that a node forwarded a packet, but as I will explain in Chapter 5.3.4, this shouldn't be a problem in most modern societies.

5.3.4 legal implications of being a proxy

Before I start talking about the legal implications, I'd first like to clarify that I am *not* a lawyer and can only rely on talks with people who have actually studied law.
As this thesis is written in Germany, I'd like to give a short overview of the current german laws regarding the situation of a proxy in a network. This is important to know as some of the implementations of anonymous file-sharing protocols like the ones implementing a form of proxy chaining will put data on your node and make it the origin of material that might be copyright infringing or pornographic without the your node specifically asking for it.
While communication is encrypted in most of the cases, it might happen that the node directly in front or directly behind you in a proxy chain will try to sue you for forwarding (in a simpler view: "sending out") or receiving a file. That could be because the file your node forwarded is being protected by copyright or the content itself is illegal (e.g. child pornography). Especially in the case of severely illegal things like child pornography, it's important to know how the current legal situation might affect the owner of a node who might not even know that kind of data was forwarded.
As a simple "proxy", the user should be covered by §9 TDG:

> *§9 TDG: Durchleitung von Informationen*
>
> 1. Dienstanbieter sind für fremde Informationen, die sie in einem Kommunikationsnetz übermitteln, nicht verantwortlich, sofern sie:
>   - 1. die Übermittlung nicht veranlasst,
>   - 2. den Adressaten der übermittelten Informationen nicht ausgewählt und
>   - 3. die übermittelten Informationen nicht ausgewählt oder verändert haben.
>
>   Satz 1 findet keine Anwendung, wenn der Dienstanbieter absichtlich mit einem Nutzer seines Dienstes zusammenarbeitet, um eine rechtswiedrige Handlung zu begehen

2. Die Übermittlung von Informationen nach Absatz 1 und die Vermittlung des Zugangs zu ihnen umfasst auch die automatische kurzzeitige Zwischenspeicherung dieser Informationen, soweit dies nur zur Durchführung der übermittlung im Kommunikationsnetz geschieht und die Informationen nicht länger gespeichert werden, als für die Übermittlung üblicherweise erforderlich ist.

You have to take in concern that most of the time, there is hop-to-hop encryption between the nodes in anonymous file-sharing network, but it's not very likely that this could be interpreted as "changing" the transported information according to §9 (1) Nr.3 While the general situation for nodes acting as a proxy server seems to be perfectly fine, there could be a differentiation between nodes concerning the content they forward or the network such a node is positioned in.
I will discuss two of the different worst-case scenarios a node-owner could face when operating a client in a file-sharing network in a proxy-like manner. These scenarios would be a lawsuit for:

1. Distribution of illegal content
2. Distribution of copyright protected content

5.3.5 illegal content stored on a node (child pornography as an example)

§184b StGB clearly states that the distribution, ownership or acquisition of child pornography is illegal. The paragraph doesn't explicitly make negligence illegal.
It's a bit hard to grasp the term "negligence" when it comes to child pornography, but keep in mind that in some of the networks, your PC only acts as a proxy for file-transfers started by other nodes. A user's node might even store some data to serve it to the rest of the network without the user's knowledge. In this case, the user couldn't expect this to happen and therefor might have acted negligent and it could be deducted from §15 StGB that this isn't punishable:

> *§15 StGB: Vorsätzliches und fahrlässiges Handeln*
> Strafbar ist nur vorsätzliches Handeln, wenn nicht das Gesetz fahrlässiges Handeln ausdrücklich mit Strafe bedroht.

The burden of proof in this case is yet another problem which could fill a thesis on its on and will therefor not be discussed in this paper, but usually the legal term "Ei incumbit

probatio qui dict, non que negat" comes into play (Translates to: "The burden of proof rests on who asserts, not on who denies").
The latin legal maxim of "in dubio pro reo" is in effect in most modern societies and "local" versions of it can be found

- in English: "the presumption of innocence"
- in French: "La principe de la présomption d'innocence"
- in German: "Im Zweifel für den Angeklagten"

Unluckily for the situation of anonymous file-sharing, it is always a question of the point of view if there are any doubts concerning the legality of a person's actions.
In case of file-sharing networks, it is always of interest what the network in mainly used for. If the network doesn't degenerate to a meeting point for child-molesters to swap pornography, and the existence of illegal content stays an exception in the network, a node that acts as a proxy and forwards it shouldn't be facing any punishment.
In case of the network degenerating or being covered badly in the media (usually the yellow press is pretty interested in "big headlines"...), it could be difficult to establish the "negligence" part when running a proxy node in the network.
In this case it could be seen as a potentially premeditated distribution ('eventualvorsätzliche Verbreitung') which would be judged as a premeditated offense ('Vorsatzdelikt').
At the moment, the current german legal system seems to be under the impression that a node forwarding possibly illegal data (acting as a proxy) in an anonymous file-sharing network shouldn't face any punishment for doing so.
For people more interested in the corresponding german laws, I'd recommend reading Art 103 II GG, Art 6 EMRK and Art 261 StPO

5.3.6 copyright infringing content

When it comes to data that is subject to copyright, the situation is basically the same as in Chapter 5.3.4. The main difference is, that in this case the files are protected by 106 UrhG:

> *§106 UrhG: Unerlaubte Verwertung urheberrechtlich geschützter Werke*
>
> 1. Wer in anderen als den gesetzlich zugelassenen Fällen ohne Einwilligung des Berechtigten ein Werk oder eine Bearbeitung oder Umgestaltung eines Werkes vervielfältigt, verbreitet oder öffentlich wiedergibt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

2. Der Versuch ist strafbar

Another interesting Paragraph concerning copyright is §109 UrhG, which shouldn't go unmentioned in this chapter:

> *§109 UrhG: Strafantrag*
> In den Fällen der §106 bis §108 und des §108b wird die Tat nur auf Antrag verfolgt, es sei denn, dass die Strafverfolgungsbehörden ein Einschreiten von Amts wegen für geboten halten.

The main difference to 5.3.4 is, that the border between negligence and probable intent is very small. Especially in the case of some of the bigger anonymous file-sharing networks which are at the moment almost exclusively used to swap copyrighted movies/music/software, it would be pretty unbelievable to argue that a user wouldn't tolerate his node forwarding copyright protected files (according to §106 UrhG).

5.4 Hop to Hop / End to End Encyrption

Encryption can be used as a means to keep a passive third party from eavesdropping on the content of your file-transfers. They will still allow third parties to monitor who you share files with, but besides a little hit on performance, it won't harm using it.

5.4.1 Hop to Hop

Hop to Hop encryption can also be described as "Link encryption", because every intra node communication is secured by the use of cryptography. Using a Hop to Hop encryption only secures connections *between* nodes and keeps 3rd parties which are *not* involved in the process from spying on the data which is exchanged. The nodes which are forwarding the data and are *active* components of the file-sharing-network (not just routing instances on the "normal" internet) will be able to read the data in cleartext. Every node that receives a message will decrypt, read, re-encrypt and forward this message to the next node on the route. It will also put extra stress on the network because the encryption has to occur on every hop again. Usually the speeds of internet connections aren't that fast, so recent PCs don't have a problem encrypting the data fast enough.

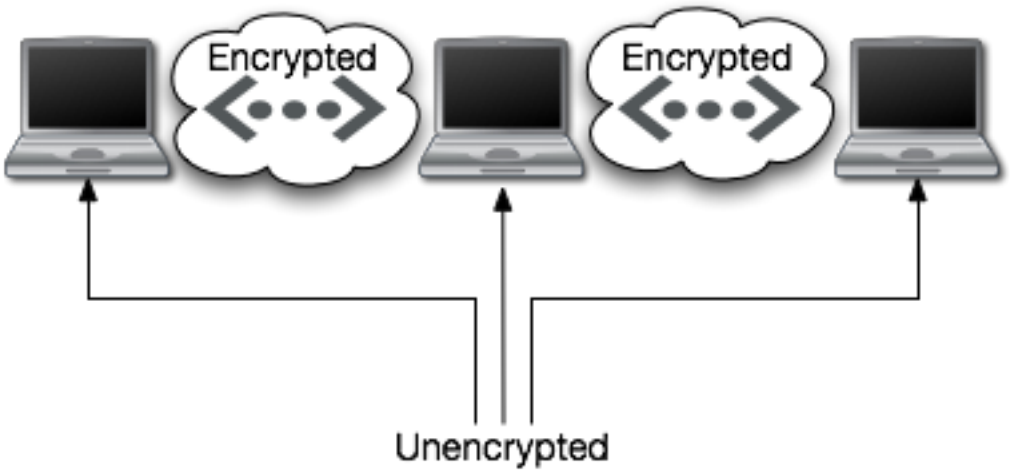

Figure 9: Hop to Hop encryption

5.4.2 End to End

End to end encryption adds encryption from the source to the end. This keeps every node besides the source and the destination from spying on the content of the data.

5.4.3 Problems with end to end encryption in anonymous networks

As Jason Rohrer, the inventor of the anonymous file-sharing software "Mute"[8] points out in his paper dubbed "End-to-end encryption (and the Person-in-the-middle attacks)" [21], using end-to-end encryption in an anonymous network poses a technical problem: How can the sender get the receivers key to start the encrypted transfer in the first place? In the majority of anonymous requests are routed over several nodes acting as proxies (compare: Chapter 5.3). Each of those nodes could act as a "man in the middle" and manipulate the exchanged keys. This would allow an arbitrary node to put in its cryptographic keys and to read and re-encrypt the traffic for each of the endpoints of the file-transfer.
On the internet, encrypted connections are usually established by using a trusted third parties keys to verify the integrity of the person on the other end of your logical connection. While this works perfectly fine on the internet, using this "trusted third party" scheme on an anonymous network would compromise your anonymity by the necessity to reveal your identity to the trusted third party. Even if that was somehow possible, it

[8] `http://mute-net.sf.net`

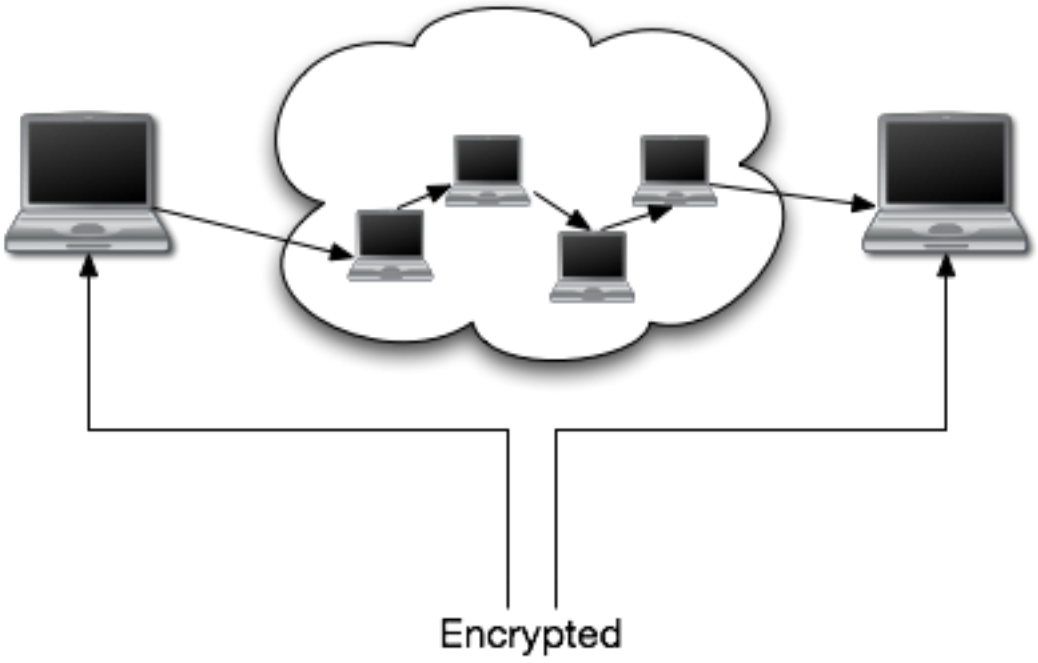

Figure 10: End to End encryption

is pretty impossible to request the key of the person you're trying to communicate with if you don't know who it is (remember: the network is anonymous).
It basically comes down to this main problem:
To establish a secured/encrypted connection with another node on the network, you somehow have to exchange keys (or other cryptographic information). As long as you're routing this information through the network, intermediary nodes can interfere and compromise the security of the resulting end-to-end encryption.
There were attempts to make use of end-to-end encryption by the creator of AntsP2P[9], but his paper *Why End 2 End Encryption is useful for an AD-HOC anonymous NET* wasn't received that well by the community and it seems to be offline (although it is still linked to) for some time now.

[9] `http://antsp2p.sourceforge.net`

6 Current software-implementations

This section will go into detail about the current leading implementations of the different anonymity patterns discussed in Chapter 5 and what makes them special (e.g. in terms of scalability). Implementations like OFF (chapter 6.1) will be discussed in more detail because of the importance of their implementation in concern to the way the network works.

6.1 OFF- the owner free filesystem - a brightnet

As the brightnet pattern was a pretty high level approach to the whole subject of copyright, the focus on the implementation of the client will go a bit more into detail.

6.1.1 Structure of the network

As discussed in chapter 4.1, there are different generations of file-sharing protocols, OFF itself is using a state of the art DHT based approach to structure the whole network and the data in it. The only purpose of the DHT structure in OFF is to store and locate blocks. According to its developers, OFF doesn't represent the "traditional" DHT approach when it comes to the initial formed connections when joining the network. OFF does not maintain a regular node structure and there are no deterministic circles or chords. This means that there is no "structured" way to create connections inside the off network, the network becomes connected arbitrarily as nodes search for blocks. This mechanism bares resemblance to the way Gnutella connects to other nodes (compare chapter 4.4). This leads to OFF being more highly connected than most other DHT based networks.
The nodes usually have a large number of other nodes they "know about", but they only are connected to a few of them. A node can adaptively add and close connections as far as it's necessary for an efficient search in the DHT. The management of node connections is actually done in two separate lists, one for the inner-circle of nodes which return blocks consistently and quickly, and one for the outer-circle of nodes close to it (according to the DHT's distance metric). This allows arbitrary nodes to be located by asking the closest node you know for the closest nodes it knows.
When looking at the network as a whole, one could say that the network basically consists of three main building blocks: nodes,blocks and URLs.

Nodes Every node on the network has a unique private/public key pair. Every node

identifies itself to other nodes sending an ID, generated by using the SHA-1 hash function on its public key. This ID is the logical identifier which will help structure the network. The IP address of the node is only the "physical" identifier which is needed to have connectivity on TCP/IP networks.
Example: c4c3b73899f753a92aaf77b8e6c507497c34ded5

Blocks Every piece of data *AND* metadata in the network is stored as a 128 kb block, padding is applied to fill up the 128 kb. There are two categories of blocks: stored blocks and derived blocks. They differ only in their access mechanism but look completely alike to a third party.

Stored Blocks

Stored blocks are accessed associatively using their ID as an access key. Because these keys are always 160 bit SHA-1 hashes. There can be a maximum of 2^{160} accessible unique stored blocks in any given OFF Network.

Derived Blocks

These blocks are never physically stored in the OFF network. However, they can be computed using a subset of stored blocks called a tuple. Tuples can be made up of 3, 4 or 5 stored block. A requested block is then derived by XORing the tuple blocks together.

There are three main types of derived blocks (see Figure 11):

Source Blocks Contain the data itself

Hash Map Blocks Contain the hashes of Source Blocks or other Hash blocks (a layer of indirection). This allows to check for the integrity of reconstructed source blocks

Descriptor Blocks These are the first blocks that the OFF Client downloads. They contain hash tuples of either Hash Map Blocks or Source Blocks

Every block uses a 40 character hash value of its data as its identifier.
Example: eee6a6b9c6dcc6ba6d9ba1859a8ed3a2926d53f4

URLs an URL is the metadata describing the way blocks have to be combined to recreate a file that was inserted into the network. The URL simply contains the identifiers of certain blocks

Example: `http://<host>:<port>/offsystem/<urlversion='v2'>/<mime>/<type>/<maphash>/<filesize>/<filehash><deschash>/<dh1>/<...>/<dhn>/<offset>:<stream_len>/<filename>`

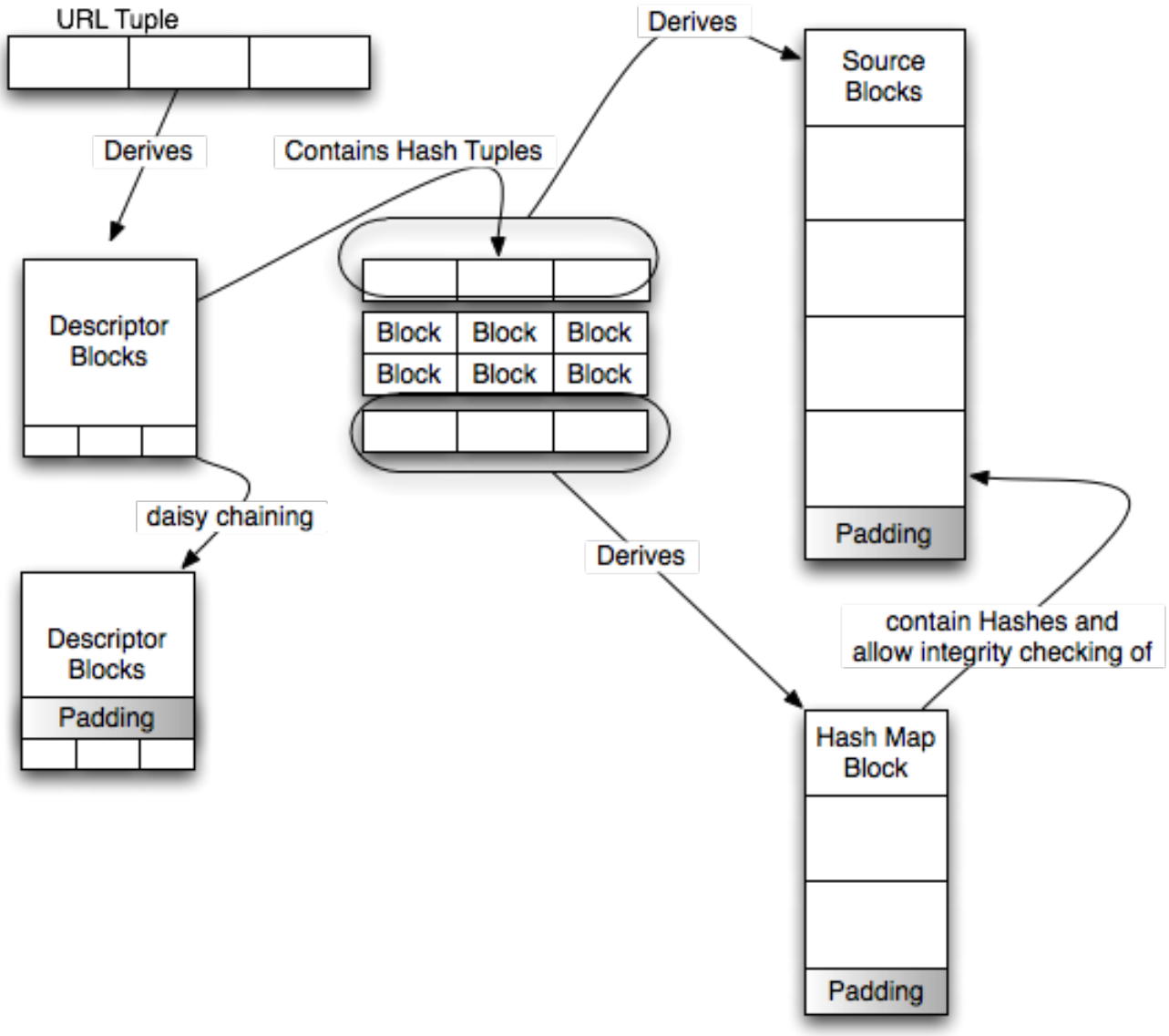

Figure 11: Different types of Blocks in OFF and their relation

This will tell the client the following things:

<host>:<port> Upon clicking, send a HTTP get request to the application listening on the given port on the given PC (it is possible (disabled by default) to control a machine other than localhost using this mechanism)

<urlversion='v2'> This is a link in version 2 link format (the current link format, differences to v1 will not be discussed due to relevance)

<mime>/<type> The MIME Type of the download, e.g. /application/pdf/ for a PDF document

<maphash> The hash of the first hashmap block

<filesize> The total filesize of the download

<filehash> The hash of the final download

<deschash> The hash of the result of the descriptor tuple

<dh1>/<...>/<dhn> those are the hashes of the descriptor blocks which contain the information about all the other blocks needed

<offset>:<stream_len> used for the splitting information of a concatenated store. The details on the concatenated store are described in the *Scaling OFF under load* chapter (chapter 6.1.7)

<filename> The filename the resulting download will be saved as

6.1.2 Cache and Bucket

Before going into detail, a short explanation of the term *Hamming Distance* is necessary, this will be of use in the next pages:
The **Hamming distance** between two strings of equal length (in our case: the client- and blockhashes) is the number of positions for which the corresponding symbols are different.
As an example:
Client ID: abcdef12345
Block ID: abcdef99345
–> Hamming distance of 2 (2 characters are different)

In OFF, the hamming distance is computed on a bit level (not a character level as shown in the example).

The OFF client has two different areas where it stores its block on the harddisc:

cache every block that is downloaded or inserted is being placed in the blockcache. it grows according to the need for more space (e.g. when starting new downloads) and needs to be "trimmed" by moving blocks with the biggest Hamming distance to the own Node ID to other nodes on the network with a closer Hamming distance.

bucket the bucket is the place in which a client stores blocks that have a close Hamming distance to the client. the bucket won't grow or shrink unless it is specifically told to do so. It will however be in a state of constant optimization by trading the blocks with the largest Hamming distance to the clients ID with blocks it happens to run into (e.g. when downloading things or actively searching for blocks) that have a smaller Hamming distance to its node ID.

6.1.3 Bootstrapping

As previously mentioned, the OFF network isn't very DHT centric when it comes to connecting to other nodes.
On the first start, OFF will pay attention to a file called *local_ URLs.txt*. This file contains URLs to public domain files (e.g. the copyrightable numbers paper on which off is based) which are tagged with IP addresses of the bootstrapping nodes. These nodes either use a service like dyndns.org or have a static IP. Once off starts it will connect to these bootstrapping nodes and ask for other nodes. This isn't a very elegant solution and without these nodes, it would be fairly impossible to join the network without coming to the projects IRC channel and asking for a push. In my testing period of more than a year, the bootstrap nodes had an uptime of >99%

6.1.4 Search

The search function in OFF is basically a search for URLs (the "recipe" for combining blocks so that a particular file will be the result). It is possible to disable the sending of search results or to forward searches that contain certain words.
The current search implementation is a heavy work in progress and has not been documented all that well besides in the source code. The algorithm itself is a depth-search based approach with a maximum of six tries, doubling the search interval each time. There is *no* anonymization taking place at the moment, so search results should only be

passed out to trusted nodes (you can define which nodes you trust inside the application) To quote the developer: *ATM it's just a simple system to get you some damned URLs.*

6.1.5 Downloading

Once a client has received an URL, it will parse it for the hashes of the descriptor blocks and start looking for them on the network. These blocks combined (XOR) will result in the first descriptor block which holds the hash tuples of blocks that, when combined, result in either source blocks or hash map blocks.
After downloading and recombining the blocks (the details on this follow in the next paragraph), the client can check the integrity of the recombined source-blocks using the hash-information found in the hashmap blocks.
The advantage of having a DHT to organize the data is, that every block has a *logical position* inside the network. As every client has a client ID (SHA1 hash of the clients public key) that is as long as the block hashes, the client is able to calculate the Hamming distance between the two.

6.1.6 Sharing

Sharing a file consists of *inserting* it into the network and *dispersing* it to other nodes.

inserting means that the file a node wants to share is combined with available blocks according to the main idea behind OFF. There are different ways of choosing the other blocks to combine with, as these are only used to optimize network performance, they will be discussed in the *Scaling OFF under load* section, chapter 6.1.7. Please also note that inserting a file will only create the blocks and an URL for the file, it hasn't bee distributed ('dispersed') to the network yet.

dispersing The term *dispersing* describes the process of sending blocks which were created in the *inserting* phase to the network. They way the blocks are dispersed is currently under heavy development and will be discussed in the *Scaling OFF under load* section, chapter 6.1.7

6.1.7 Scaling OFF under load

Scalability can't be achieved by simply doing some compiler tweaks, it has to be in the protocol and the way it handles the usual and unusual occurrences in the network. To

give a short overview over the different techniques which are each adding a bit of better network performance or robustness, we have to look at the parts the protocol consists of:

inserting to be able to handle inserts of thousands of small files as well as on big chunk of data, there are different modes when it comes to inserts:

targeted store if you plan to insert a series of documents but want to keep the overhead of randomly selected blocks to mix them with down, you would chose a targeted insert. A targeted insert allows the user to chose which files he wants to include in the block XORing process. This could be used to e.g. insert a series of podcasts by simply selecting the previous episode on each new insert. People that download Podcast episode 43 are likely to still have episode 42 in their blockcache. It basically allows the user to control which ”arbitrary” blocks are used in the block-creation-process. This allows for block-reuse which helps keeping the overhead down.

concatenated store if you plan to insert many small files, the usual thing off would do is to XOR each file and add padding until the 128 kb of a block are filled up. When e.g. dispersing a source code archive, you'll end up wasting 50% of the space for padding. Using a concatenated store will allow the user to select the files he or she wants to insert and those files will be joined together to fill up the 128kb of a block. They will be split when being downloaded again using metadata from the URL (the optional "offset" and "length" part)

dispersing When dispersing a block, the OFF client usually looks for the node ID with the smallest Hamming distance to the blocks. This will make it very easy to find the blocks later on. The problem that occurred during the first testing of the network was clients simply leaving the net and taking the files with them. The original node that sent out the block still has it, but it will take a very long time until the downloading node starts asking other nodes for the file. To solve this issue, there first was the idea of recovery blocks using erasure encoding, this however could lead to a problem for anonymity of data according to the developers.
The developer said this about the erasure coding situation:
annoyingly, it came down to this:
EITHER, the RS blocks were trivially identifiable as resulting from a given insert,

and thus subject to legal attack, OR, the insert+RSblock system was not fully recoverable from a completed download, and thus subject to cumulative failure. neither situation was tolerable, and none of us could think of a solution. so I called a halt to dev on the RS system, pending some brainwave or other.

The current solution is to send the block out 3 times: one time to the node with the closest Hamming distance, one time to the fastest known node, one time to the node with the longest known uptime.

Another thing which is beneficial to the networks scalability is the way the network *cleans itself*. Once a client has been asked for a certain block often enough, it realizes that there is a block that seems to have a close hamming distance to itself and therefor should belong to him. This will lead the client to look for the block itself and download it into his bucket.

downloading When a node is downloading a file from the network, we have to take the whole structure of the network in concern in order to understand why it is efficient the way it is.

- Every node has a node ID.
- Every block has a block ID.
- Two nodes can be compared using their node ID's Hamming distances (which is the number of bits between the two that are different. This results in a number from 0-160)

Each node is responsible for keeping the blocks "closest" (using the Hamming distance.) to its node ID. Every node advertises a "maximum" hamming distance that it will try to retain blocks for, that distance is called the "bucket radius".

Other nodes are allowed to ask a node for any block ID that is within the node's own bucket radius. Without this limitation, it would be easy for an adversary to know if a node has been downloading particular URLs by simply asking this node for every block that is described by that URL. The "theoretical" way to find each block is to ask the 'closest' node to that block, and it *SHOULD* have the blocks.

As the network grows it won't be possible to "know" all of the nodes online at a given moment, but thanks to the DHT foundation of OFF, it is very likely that within 3 messages, a node will find the block in question. The way this works is that a request for a block can also be answered with a list of nodes that are close to the block from a Hamming distance point of view. This leads to the situation that with every failed block-request, a node gets closer to the blocks position within

the network.
Over time, blocks will automatically move to their closest nodes and the download/routing speeds in the network should get faster since a node can find each block in fewer requests.
This is one of the ways that allow OFF to keep performance good even if the network starts growing.

6.2 Retroshare - a friend to friend network

Retroshare implements both, the original darknet idea (described in Section 5.1) and the darknet-expansion pattern called "Turtle Hopping" (discussed in Section 5.1.1). As there is no real open network, scalability and implementational details are pretty trivial and only two of the more important parts (bootstrapping and encryption) that are vital to the function of the client will be dealt with.

6.2.1 Bootstrapping

Especially in Germany, ISPs tend to disconnect the user every 24 hours and reassign the user a new IP address. This poses the problem of not being able to locate previously added friends after both of the nodes have had an IP change while being not part of the network. Retroshare uses the openly available Distributed Hashtable "opendht"[10] to be able to map identities to IP addresses. Every client updates its current IP address on startup and collects the current IPs of its already added friends.

6.2.2 Encryption

Retroshare uses a certificate based authentication and encryption using the openssl library to secure traffic between peers from third parties. This allows authentication and circumvents spoofing attempts.

6.3 Stealthnet - Proxychaining

Stealthnet is one of the most current representatives of the file-sharing applications using proxy-chaining to establish a certain kind of anonymity. Stealthnet itself is the community-fork/successor of and application called "RShare", which was the reference-implementation for a protocol designed by Lars Regensburger.

[10] `http://www.opendht.org`

The application was designed to be usable for the average user, optimize file transfer speeds and keep anonymity at the same time.
The basic idea behind Stealthnet is the use of a so called "RShare-ID" (96 characters) to route packets within the network instead of IP addresses. The ID is created on application-startup and is used as the clients identifier for that session. The anonymity in the network relies on the fact that no one should be able to connect RShare-ID's to IP addresses.
As the RShare network is an open network, the main focus of the implementational aspects will be on the way searching and downloading is performed.

6.3.1 Bootstrapping

In both, RShare and Stealthnet, web caches are the usual way to connect to the network. As in other file-sharing networks, web caches save the IP+port combination of members of the network and share this information with new nodes to allow them to join the network. The uptime of the web caches is vital for new clients trying to join the network.

6.3.2 Encryption

All of the connections that occur in the network are point-to-point encrypted with RSA + Rijndael (AES).

6.3.3 Search

When searching for files in the network, a node basically floods the search request to its surrounding peers. The search request contains the RShare ID of the node sending it. The peers themselves forward the search request with their RShare ID as source. That way, a node can't tell if the search request it just received did originate or was only forwarded by the node sending it to them.
This behavior however would lead to something equivalent to broadcast storms inside usual ethernet networks. In the ethernet world, those broadcast storms occur when improperly configured networks have loops in their topology and ARP requests are passed around in a circle. In the LAN[11] world, protocols like "Spanning Tree" keep those storms from happening by organizing the network in a certain way which keeps loops from occurring. Another way of keeping broadcasts down to a minimum is to

[11]Local Area Network

ensure that broadcast domains[12] don't grow too large by using routers and other higher-layer devices and configuring VLAN instances to separate a network into smaller pieces. As organizing an anonymous, fully decentralized network tends to be complicated task, the RShare protocol has included a "HTL" (hops to live) counter, similar to the TTL (time to live) counter of IP packets. Every node receiving a search request will increase the HTL counter by 1 and starts dropping the request when a certain maximum (35 in Stealthnet v 0.81) has been reached.
There are slight modification to that scheme, as a simple HTL counter would allow nodes that receive a request with the HTL of 1 to deduce that the node sending this request to them is actually the creator of the request. The same goes for nodes that would receive a maximum HTL counter and would by answering to the request debunk the fact that they themselves have the files they answered with.
The modification to solve those problems is a pretty simple one:
A client starting a request will NOT include a HTL counter and a client receiving a request without a HTL counter will only add one with a possibility of 20%.On the other extreme, a client receiving a search request with a HTL counter of 35 will only mark it as "HTL exceeded" and pass it on. Clients receiving an exceeded HTL counter will only drop it with a possibility of 50%. That way, a 3rd party controlling a node couldn't tell if the person sending a request or answering one is the source of either the request or the answer.

6.3.4 "Stealthnet decloaked"

Nathan Evans, Nils Durner and Christian Grothoff of the Colorado Research Institute for Security and Privacy did a security analysis on Stealthnet called "StealthNet decloaked"[22]. Although the flaws they found had already been fixed in the development branch of the application but hadn't arrived in the current "stable" release at that time. The way that allowed them do de-anonymize is interesting never the less.
Please keep in mind that the main security feature of stealthnet is that an RShare-ID (found in searches, search results, download requests, ...) can't be connected to the matching IP. The main flaws they found and which allowed them to connect ID to IP were:

- They way the 20% possibility of a node increasing a search request's HTL counter came to be

[12] A broadcast domain is a part of a network in which all nodes can reach each other by broadcast at the data link layer.

- The fact that the client used the same IDs for search requests and results.
- The way a client forwards searches with an exceeded HTL counter

The first flaw consisted of the way the 20% HTL counter increase was created. A client receiving a search request without an HTL counter used the so called "Flooding-Hash-Value" of the received packet and did a modulo 256 operation on it. If the result of the modulo 265 operation was smaller than 51 (which is the fact in 20% of the cases), the node forwarded the request with a HTL of 1. The problem was that the forwarded package includes a new flooding hash value.
This lead to the situation that a node receiving a search request WITHOUT an HTL counter but WITH a flooding hash that is bigger than 51 when put into a mod(256) operation could deduce that the node sending the request was the origin of it. This was fixed by forcing origins of a search request to produce flooding hashes that are greater than 50 when doing a mod(256).
The second flaw simply allowed to connect the results of searches with an IP address by looking at searches that were started by that IP.

7 Attack-Patterns on anonymity

Once there is a network, there is somebody who wants to bring it down. TV Soaps seem to rely on this principle for their daily social drama. Ever since file-sharing came up, single people, organizations and countries were trying to protect their copyright/power by influencing the network in a negative way.
The main ways of disturbing a network are similar to both, anonymous and "usual" file-sharing applications, with the typical ones being:

- Attacks on communication channel: weaken communication between peers
- Attacks on Anonymity of peers: try to reveal the identity of peers that are sharing information in the network.
- Attacks on individual files: polluting a file

A good overview about the different kinds of attacks can be found in Dharma Reddy Manda's master thesis "A study on anonymous P2P Networks and their vulnerabilities" [23]

7.1 Attacks on communication channel

7.1.1 Denial of Service

The most popular way of manipulating a network today seems to be a DoS[13] attack on its participants. The principle is, that either central points in the architecture of a network (servers/supernodes/...) are rendered unable to do their job, therefor disabling proper communication within the network itself, or the links between clients are deliberately jammed, therefor disturbing proper routing and sometimes even splitting whole networks. Two of the bigger examples are the "Great Firewall of China" and the US ISP Comcast. Both of them posses the power to control all of the network traffic in their part of the internet. They use this power to break TCP/IP connections they deem harmful in one way or another. Comcast tries to get down the amount of data being shared over their network, China isn't too keen on having its citizens know about certain things that happened in the chinese past.
In the case of china, special search terms (e.g. the Tiananmen Square protests of 1989 aka the "Tiananmen Square massacre"[14]) on wikipedia or popular search engines are

[13] Denial of Service
[14] `http://en.wikipedia.org/wiki/Tiananmen_Square_protests_of_1989`

filtered by their networking equipment, in the case of Comcast, the Bittorrent protocol (among others) is detected and attacked.
In both cases, faked TCP RST packets are send both, to the sender an the recipient, effectively terminating the connection.
One of the less elegant ways to bring down communication between hosts is to overload one of the participants with work by e.g. permanently sending requests from a fast internet connection.
Today, distributed Denial of Service attacks are en vogue because of how easy it has become to acquire a botnet and the risen connection speed of "home connections". They are basically the same as the "original" version, but instead of taking one really fast connection, hundreds of smaller home connections are utilized. This also makes it harder to block using conventional firewall rules.
Basically, attacks on the infrastructure of a network can only be prevented by using proper protocol design or (if possible) a modification of firewall rules. It has been shown in the paper "Ignoring the great Firewall of China"[26] by Clayton,Murdoch and Watson, that simply dropping RST packets on both sides of the connection allows for proper communication over the great firewall of china, as the original packets aren't dropped, but still arrive at their destination.

7.2 Attacks on Anonymity of peers

One of the main ways to influence an anonymous file-sharing network (besides DOS attacks) are attacks that destroy the anonymity of peers. Using these techniques, presence in the network or downloads/distribution of files could be censored by a third party by simply gaining knowledge of a nodes IP address and then tracking them down in "real life".

7.2.1 Passive Logging Attacks

Passive Logging Attacks are a family of different long-term attack techniques against anonymous communications systems. They all have in common, that the attacker usually doesn't act as a participant within the network itself, but is eavesdropping on the network traffic of a single peer or a group of peers.

7.2.2 Passive Logging: The Intersection Attack

By using e.g the *intersection attack*, an attacker would take a collection of a timely disjoint set of nodes and the matching lists of IP addresses in the network at the given time. Intersecting those sets will decrease list of possible IPs for a given node. A simple example is shown is Figure 12. A detailed look at passive logging attacks can be

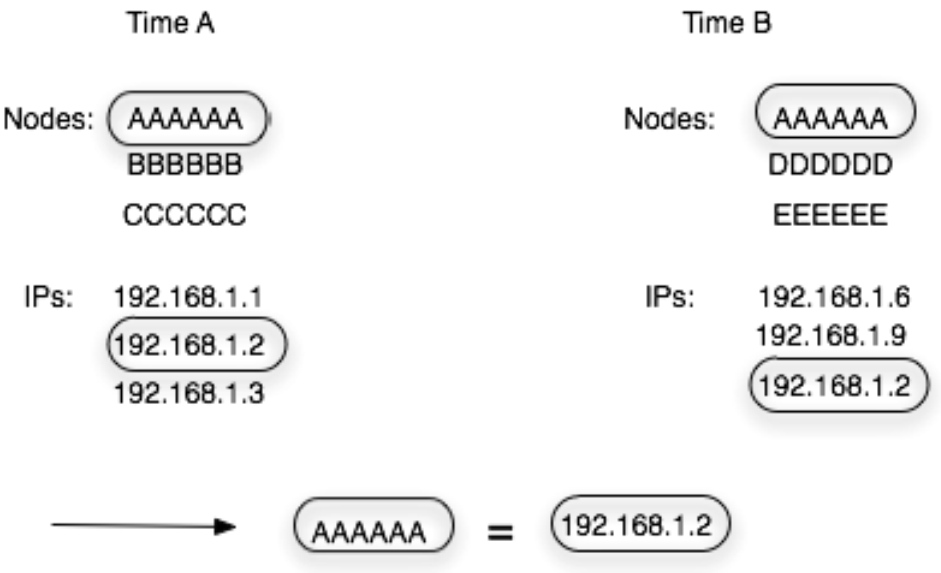

Figure 12: The intersect attack - a passive logging attack

found in Matthew K Wright's dissertation "Passive logging attacks against anonymous communications systems"[27] or in Andrei Serjantov's and Peter Sewell's paper *Passive Attack Analysis for Connection-Based Anonymity Systems* [28].

7.2.3 Passive Logging: Flow Correlation Attacks

A flow correlation attack can be done by a 3rd party by basically analyzing similarities in the senders outbound flow of data and the receivers inbound flow of data. The result of this attack is a table of probabilities which connect the IP addresses in a network with a clients identity within the network. Several different ways are described in the paper *Timing analysis in low-latency mix networks: attacks and defenses* [4] by Shmatikov and Wang of the University of Texas at Austin.

7.2.4 Active Logging: Surrounding a node

While being related to the passive logging attacks, the ”surrounding a node” attack is an active form of logging. Especially in the proxy-chaining based networks, a node could be attacked by simply surrounding it with nodes controlled by a third party. If the node under attack only connects to nodes controlled by a malicious third party, the traffic going to and coming from the node could be analyzed. If the node receives a package without forwarding it or sends a package without receiving it first, the third party could detect that the node under attack is the origin/destination for the search request/download.

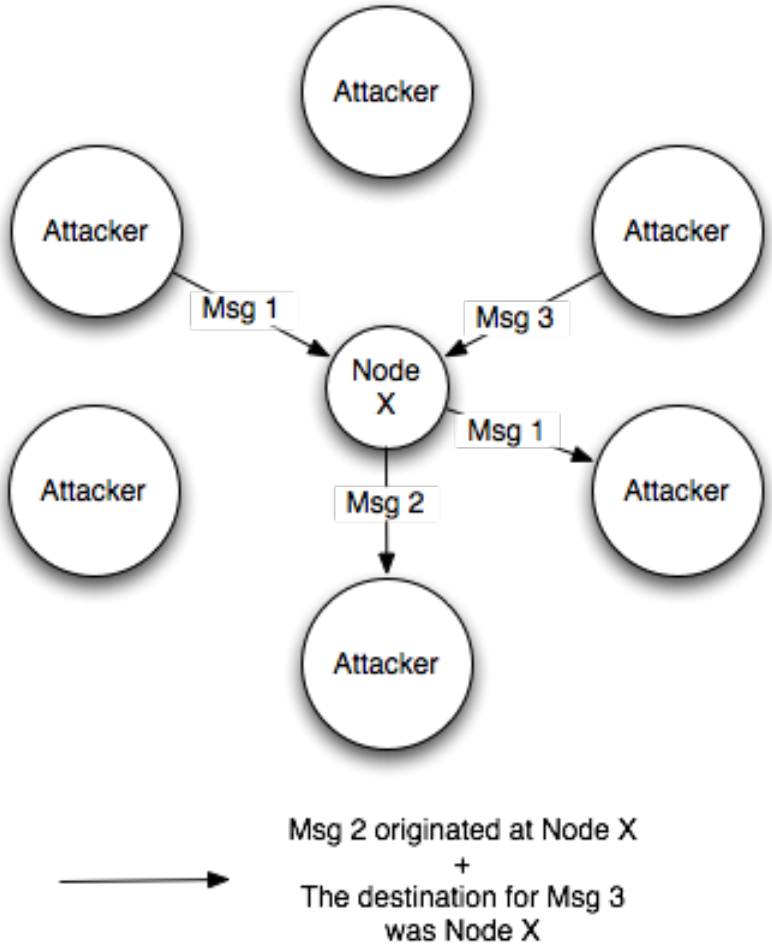

Figure 13: Surrounding a node

List of Figures

References

[1] Arvin Nayaranan and Vitaly Shmatikov of the University of Texas in Austin: "Robust De-anonymization of Large Sparse Datasets"

[2] Michael Piatek, Tadayoshi Kohno, Arvind Krishnamurthy, University of Washington, Department of Computer Science & Engineering: Challenges and Directions for Monitoring P2P File Sharing Networks –or– Why My Printer Received a DMCA Takedown Notice
http://dmca.cs.washington.edu/uwcse_dmca_tr.pdf

[3] the IPoque Internet Study 2007
http://www.ipoque.com/userfiles/file/p2p_study_2007_abstract_de.pdf

[4] Shmatikov and Wang, University of Texas at Austin: Timing analysis in low-latency mix networks: attacks and defenses
http://www.cs.utexas.edu/~shmat/shmat/esorics06.pdf

[5] Gnutella Specification v0.4
http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[6] Gnutella Specification 0.6
http://rfc-gnutella.sourceforge.net/developer/testing/index.html

[7] Gnutella for Users
http://rakjar.de/gnufu/index.php/GnuFU_en

[8] Limewire Documentation concerning Dynamic Querrying
http://wiki.limewire.org/index.php?title=Dynamic_Querying

[9] The Darknet and the Future of Content Distribution by Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman of the Microsoft Corporation.
http://crypto.stanford.edu/DRM2002/darknet5.doc

[10] Azureus Wiki: ISPs that are bad for Bittorrent
http://www.azureuswiki.com/index.php/ISPs_that_are_bad_for_BT

[11] The Bittorrent Specification 1.0
http://wiki.theory.org/BitTorrentSpecification

[12] Azureus Wiki: Message Stream Encryption
http://www.azureuswiki.com/index.php/Message_Stream_Encryption

[13] Scott R. Fluhrer, Itsik Mantin and Adi Shamir, Weaknesses in the Key Scheduling Algorithm of RC4. Selected Areas in Cryptography 2001 pp1 24

[14] Standard Cryptographic Algorithm Naming database
http://www.users.zetnet.co.uk/hopwood/crypto/scan/

[15] Allot Telecommunications "NetEnforcer appliance" Press Release
http://www.allot.com/index.php?option=com_content&task=view&id=369&Itemid=18

[16] MD4 Section of the Amule Wiki: http://www.amule.org/wiki/index.php/MD4_hash

[17] Emule Wiki: Protocol Obfuscation
http://wiki.emule-web.de/index.phpProtocol_obfuscation

[18] Turtle Hopping Pattern
http://www.turtle4privacy.org

[19] Cracker Jack: On copyrightable numbers with an application to the Gesetzklageproblem
http://offsystem.sourceforge.net/CopyNumbCJ.pdf

[20] Michael Rogers (University College London), Saleem Bhatti (University of St. Andrews): How to Disappear Completeley: A survey of Private Peer-to-Peer Networks
www.cs.ucl.ac.uk/staff/M.Rogers/private-p2p.pdf

[21] Jason Rohrer: End-to-end encryption (and the Person-in-the-middle attacks)
http://mute-net.sf.net/personInTheMiddle.shtml

[22] Nathan Evans, Nils Durner and Christian Grothoff of the Colorado Research Institute for Security and Privacy: Stealthnet Decloaked
http://crisp.cs.du.edu/youRShare/

[23] Dharma Reddy Manda: A study on anonymous P2P Networks and their vulnerabilities

http://www.medialab.tfe.umu.se/publications/master_thesis/2007/a_study_on_anonymous_p2p_networks_and_their_vulnerabilities.pdf

[24] Hummel: Würdigung der verschiedenen Benutzungshandlungen im OFF System Network (ON) aus urheberrechtlicher Sicht
http://offcorps.org/files/copyright-off%20(v.1).pdf

[25] Phillip Porras and Hassen Saidi and Vinod Yegneswaran, Computer Science Laborator: A Multi-perspective Analysis of the Storm (Peacomm) Worm (2007)
http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf

[26] Clayton,Murdoch and Watson: Ignoring the Great Firewall of China
http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf

[27] Matthew K. Wright University of Texas at Arlington and Micah Adler and Brian Neil Levine University of Massachusetts Amherst and Clay Shields Georgetown University: Passive logging attacks against anonymous communications systems
http://ranger.uta.edu/~mwright/papers/wright-tissec07.pdf

[28] Andrei Serjantov, Peter Sewell: Passive Attack Analysis for Connection-Based Anonymity Systems (2003)
http://www.cl.cam.ac.uk/~pes20/conn_sys.ps