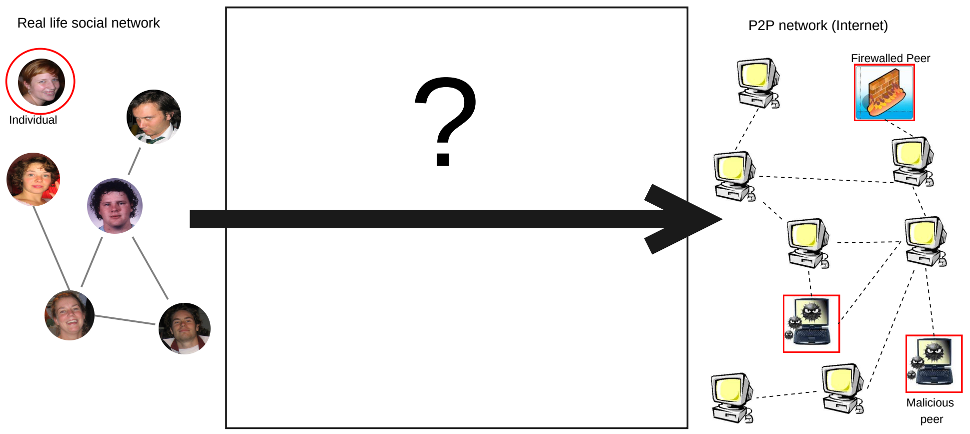
Creating and Maintaining Relationships in Social Peer-to-Peer Networks

S.D. Koolen



Delft University of Technology

Creating and Maintaining Relationships in Social Peer-to-Peer Networks

Master’s Thesis in Computer Science

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

S.D. Koolen

4th January 2007

Author
Steven Daniël Koolen
Title
Creating and Maintaining Relationships in Social Peer-to-Peer Networks
MSc presentation
12th January, 2007

Graduation Committee

Prof. dr. ir. H. J. Sips (chair)	Delft University of Technology
Dr. ir. D. H. J. Epema	Delft University of Technology
Dr. ir. J. A. Pouwelse	Delft University of Technology
Dr. P.H. Westendorp	Delft University of Technology, Faculty of Industrial Design Engineering

Abstract

Peer-to-Peer (P2P) systems are a good alternative to conventional client-server systems for distribution of content. Many different approaches exist towards the design of P2P networks, which should take into account scalability, efficiency, availability and integrity. The approach discussed in this thesis creates a network based on social relations between users, called a social P2P overlay network. In order to create a social P2P network, users must be able to create and maintain connections to friends.

Two main problems in creating a social P2P network are identified and discussed in this thesis. The first is the social network discovery problem. We have designed and implemented a protocol to spread and search user identities through the social network using an epidemic protocol and superpeers. The use of epidemic protocols in social networks is a scalable and efficient way of spreading the identities of users. The second problem is the peer IP address discovery problem. We have performed an analysis of the dynamics of IP addresses of peers in a BitTorrent community as well as in the Tribler network. From this analysis we conclude that IP dynamics are relatively low. Therefore, changes in IP addresses can be propagated through the network and the chance is high that IP changes can be discovered through the social network.

Preface

This thesis is part of the Master in Computer Science at Delft University of Technology. The research was done at the Parallel and Distributed Systems Group, as part of the I-Share research project on sharing resources in virtual communities for storage, communications, and processing of multimedia data. This thesis describes my research towards the creation of a Peer-to-Peer network based on social networks with focus on building the social network in a P2P network and maintaining connections with peers in the social surrounding of a peer.

I would like to thank a lot of people who supported and inspired me, or just were around during my research. First of all I would like to thank dr. ir. Johan Pouwelse for his never ending enthusiasm and advice and ir. dr. Dick Epema for his advice and guidance when writing this scientific report. Furthermore, I would like to thank prof. dr. ir. H. J. Sips for chairing the examination committee, and dr. P.H. Westendorp for being a member of the examination committee. My thanks also go to everybody in the research group who discussed issues with me or gave me inspiring insights. During my spare time, which was scarce in the last few months, my friends and family were always available to relieve times of pressure and had trust in me. Finally, my thanks go to my parents, who are partly responsible for my enthusiasm towards technology, and have always supported me.

Steven Koolen

Delft, The Netherlands
4th January 2007

Contents

Chapter 1

Introduction

In the last decade, the use of the Internet has increased extremely fast. The Internet has developed from a text and email-based service to a rich multimedia platform. Multimedia uses a lot of bandwidth, putting pressure on servers. An alternative for the usual client-server model of distributing data is the use of Peer-to-Peer (P2P) networks. P2P networks are already widely used around the Internet, mainly for file sharing. The massive size of some P2P networks contain huge numbers of all kinds of content. Different approaches to P2P network design exist, each having advantages and disadvantages. Many system have scalability, performance and integrity issues as the network size increases.

The research as part of this thesis takes social networks as a basis for the P2P structure. Social network have implicit trust and are quite scalable. This thesis gives challenges in P2P overlay network creation and a social network based solution. Two problems in creating and maintaining a social P2P network are discussed. The first is the building of the social network. It is vital for the quick adaption of social system that the network of friends and friends of friends can be build quick and easy. A solution is proposed and implemented based on spreading identities through the social network. The second issue is keeping up connections in the social network. Analysis of two P2P networks was done as part of the research. The results show a relative low dynamics of Internet Protocol (IP) addresses.

This chapter first gives an introduction into P2P systems in Section 1.1. Some examples of P2P systems using social features are discussed in Section 1.2. Section 1.3 gives a background overview of the P2P system which is being used as part of the research project. Finally Section 1.4 gives the structure of this thesis.

number of downloads	100 Million per day
average movie size	8.3MB
total data transfered	24 PB per month

Table 1.1: An estimation of the bandwidth usage of Youtube, based on 5 samples of the top 100 downloads of all time.

1.1 Peer-to-Peer Systems

The number of users who use the Internet to get large content such as video is increasing [13]. Media, telecommunication, and broadcasting companies are introducing ways to watch content through the use of data connections, and the digital distribution of television is growing. At the same time interactive television [19] and possibilities to create and publish own content are increasing. For example sites like `youtube.com` and `video.google.com`, which allow users to share their own video content, are very popular.
Today, most content providers still store content on central servers, which must be able to handle huge amounts of data. For example, the Dutch public broadcasting company has a service (`www.uitzendinggemist.nl`) where recent television shows can be watched. They have now reached their maximum of 4 Gbit per second of data and will expand to 22 Gbit per second in the near future[1]. Another examples is the bandwidth usage of youtube. Although no official figures are available an estimate[2] [3] has been made in Table 1.1.
The bandwidth costs are very high, while not even taking into account the cost of the servers. Clearly the servers are the bottlenecks in the distribution system. More users will be using these video-on-demand systems and in the future more and more content will be displayed on televisions directly from the Internet. Most of the content online at this moment is of low quality because of bandwidth constrains. For example the recommended size of content on YouTube is 320x240 while standard television already has far superior resolutions. As quality will rice so will the bandwidth consumption. Another disadvantage of servers is that they are single points of failure. If the central server fails the content is no longer available.
P2P systems are an alternative to the use of central servers for content distribution. P2P system use distributed methods to share and distribute content. Each participant in the systems is both client and server. Individual clients can connect directly to each other rather than through a computer designated as a central server. The main function of current P2P systems is efficient sharing and distribution of con-

[1]Source: Webwereld, 22-05-2006 (in Dutch)
[2]Measurement by http://willy.boerland.com/
[3]Based on Youtube Fact Sheet: http://www.youtube.com/t/fact_sheet

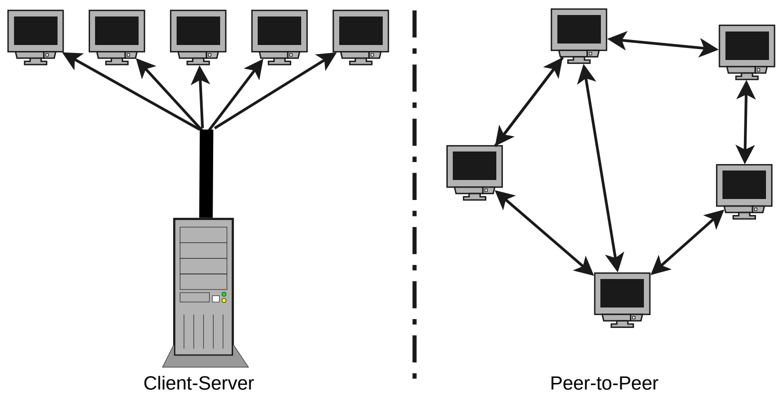


Figure 1.1: The Client-Server model versus the Peer-to-Peer model.

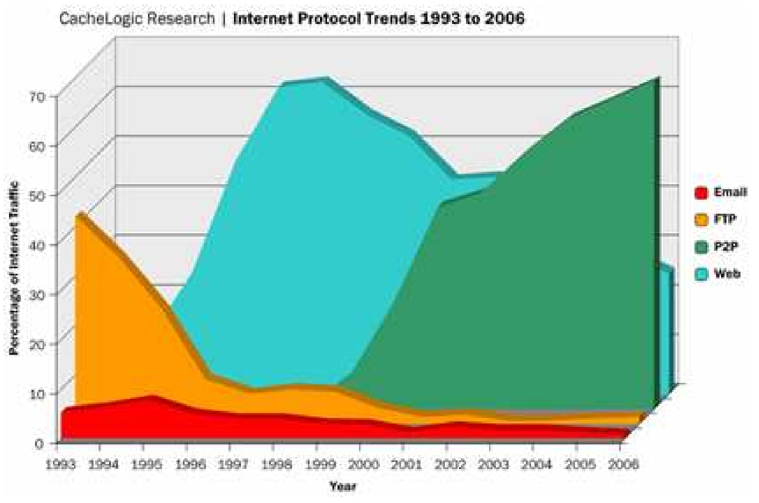


Figure 1.2: Data usage trends of the Internet (source: Cachelogic, 2006).

tent and sharing of resources. If one participant, a *peer*, in the network fails the network as a whole should continue to function. Figure 1.1 shows the difference between a client-server model and a P2P model. The content in the P2P model is both send and received by other peers. In the client-server model the content is provided by the server and received by the clients. In the client-server model the bandwidth required for the server equals the sum of the bandwidths required by the requesting peers. In a P2P system that pressure is no longer on one server. Instead the load is spread among peers in the network.

P2P systems are increasingly popular. At the end of 2004 already 60% [36] of all Internet traffic was used by P2P systems. Figure 1.2 shows the upward trend. Most existing P2P systems are based on anonymous connections between peers.

Some of these networks are tainted by malicous use [28]. A major problem in current systems is the lack of incentive to participate in sharing content and bandwith. A lot of poeple use the system without contributing, called freeriding. The performance of most systems decreased after being adopted by general users who tend to be egoistic in their behavior. Only 10% of the P2P community shares 99% of the bandwidth [45]. Also, current P2P systems are subject to attacks or misuse of the network. Most systems that work well nowadays have some form of central moderation to prevent malicious use. However, these central components must be avoided for scalability [8], availability [4], and stability [39] reasons.

1.2 Social Peer-to-Peer Systems

In P2P networks control and trust is partly distributed among the users of the network. As discussed in the previous section some systems fail to function because people try to maximize their profit and tend to freeride. Closed user groups tend to work better due to the social control. Examples are DirectConnect [14] and BitTorrent communities such as filelist.org. These systems require relationships and social control between users. Social P2P aims to create a P2P system based on social networks and thus having social control implicit in the structure.
A first advantage of social P2P networks is that peers are connected to known peers instead of anonymous peers as in present systems. This way a history can be build up and misbehaving peers can be identified and punished. Also levels of trust can be introduced by adding friends. Friends of friends (FoFs) are also probably more trusted than anonymous users. People who know the people they communicate with are less likely to misbehave and more likely to be altruistic.
A second advantage is that content favored by friends is more likely to be interesting content than content favored by random peers. The people in a social P2P network could also be people with the same interest, interest buddies. Content from these groups is even more likely to be of interest. Therefore content clustering is likely to occur. The concept of social P2P is discussed in depth in Chapter 3.

1.3 Tribler and BitTorrent

The work of this thesis was performed as part of the Freeband research project called I-share [23]. As part of I-Share, Delft University of Technology and Vrije Universiteit Amsterdam research social P2P. As part of the project an academic research vehicle has been developed called Tribler [40]. Tribler is a P2P client based on the BitTorrent protocol, and specifically on the ABC Client [10]. Tribler allows the user to add and remove friends. Improved authentication based on permanent identifiers(PermID) is added to provide identifiable peers. The social network can be used to find paths between peers through the social network. The social network is used by extra features of Tribler such as a content recommendation and discovery algorithm, called BuddyCast. Which uses a epidemic protocol to exchange preference lists with taste buddies. Tribler will also include distributed swarm discovery to overcome scalability issues with a central tracker. Cooperative downloading is an other social feature, allowing users to request friends for download assistance. Friends donate spare bandwidth. Many other features are in development in the Tribler project, such as video streaming, sharing-ratio enforcement and NAT and firewall traversal. This thesis investigates a mechanism to build and maintain the social network in the Tribler system. The results of this thesis will be partly tested in Tribler.

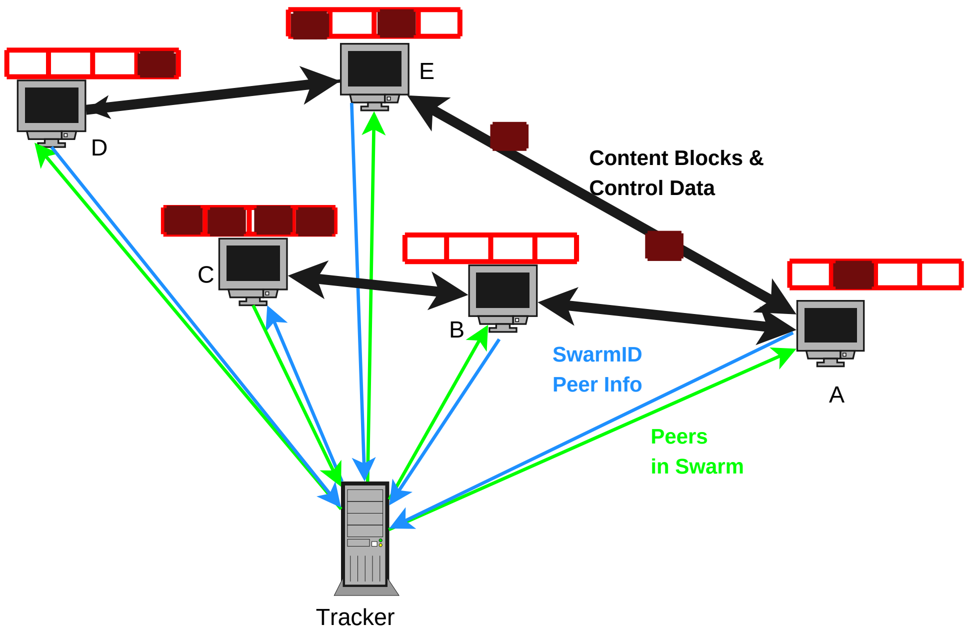


Figure 1.3: Information exchange in the BitTorrent system.

The BitTorrent system [11] uses central servers, called trackers, which coordinate single downloads (swarms). The tracker can also do some administrative task and put some limitations on the user base, for example by only allowing registered users. Users need a meta file called a *torrent* which describes the content in terms of size and hash values. This torrent also contains the address of the tracker(s). Clients request the tracker for other peers that have the content. After receiving the addresses of other peers from the tracker connections are set up to individual peers. The user can be in multiple swarms at the same time which are independent from each other. The system splits content in pieces which are exchanged between the peers. Because pieces can be downloaded from multiple peers at the same time the system is fast and efficient. The clients are also able to share the pieces that have already been acquired while still downloading others. To decrease freeriding users store past exchange data for a swarm. Peers that give much resources will be served faster. This mechanism, called tit-for-tat, is an incentive to share and forces fairness. In Figure 1.3 a simple example is given of the BitTorrent system. In the example data blocks of the swarm are distributed among the peers. Only peer C has the complete download, it is called a *seeder*. The other peers, called *leechers*, can exchange blocks with each other and peer C in order to acquire all the blocks. For example, peer A can acquire two blocks from E and the last part from either B or E, as soon as they acquired it.

1.4 Structure of the Thesis

The previous sections have given an introduction to the context of this thesis. In the next chapters we will discuss the goals and results of this research. Chapter 2 identifies problems in P2P overlay networks and the problems being studied in this research aimed at creating a social P2P network. Chapter 3 introduces a social P2P network solution and covers the research problems first introduced in Chapter 2. The two problems, Social Network Discovery and IP Discovery, are discussed in depth in Chapters 4 and 5, respectively. Finally, Chapter 6 covers the conclusions of the research, and also gives recommendations for further research.

Chapter 2

P2P Overlay Network Approaches and Challenges

P2P systems form a virtual topology of computers, called *peers*, and their connections. The network can be represented as a graph with the peers as the nodes and the network connections as the edges. In order to create a P2P network, an interconnecting network has to be created, which is called the *overlay network*. The overlay network facilitates the self-organizing system of the nodes (peers) and the edges (connections). The responsibility of the overlay network is to facilitate the efficient and working creation and maintenance of the virtual topology and facilitate the communication and connections between the peers. Designing an overlay is quite difficult. Some current designs fail to deliver good services. In this chapter we will discuss overlay networks and some problems that occur in such networks.
Section 2.1 gives an overview of the ideas behind an overlay network. Some of the most widely used P2P overlay techniques are in Section 2.2. General issues in designing an overlay network are given in Section 2.3. Section 2.4 discusses some problems in current P2P overlay networks that are treated in this thesis.

2.1 Peer-to-Peer Overlay Networks

Most current P2P networks have the goal to create a large distributed storage space to share content and share bandwidth in order to get high combined transfer speeds. In this thesis we will focus on these networks, it is however trivial to extend the ideas to the sharing of other resources such as processing power. The sharing of content is based upon replication. After initial insertion of the content in the network it will be distributed among other peers. As each of these peers also shares the content the speed at which this content can be acquired is the com-

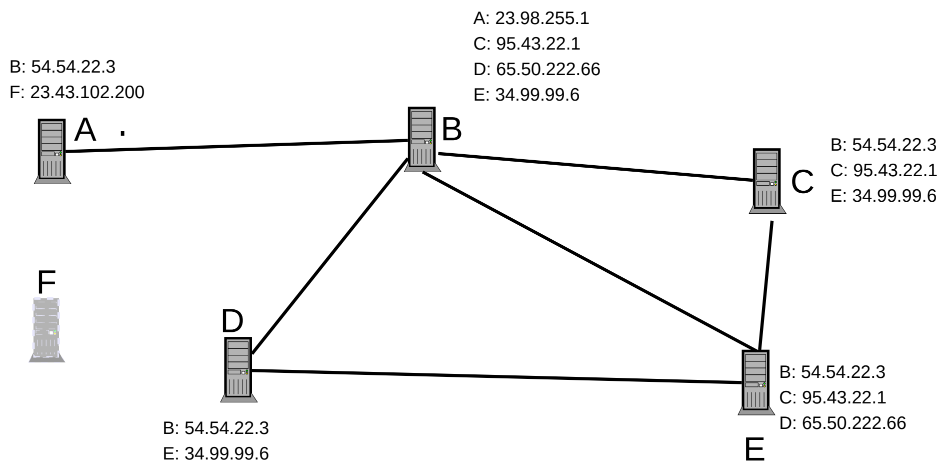


Figure 2.1: Example of a P2P overlay network.

bined available upload bandwidth of the peers sharing the content. As more and more replications of the content are available the total bandwidth available for the content will grow. Acquiring the content as well as the lookup or search for content is also facilitated by the overlay network. This may require routing messages through the network to other peers.

The idea of a P2P overlay network is too create a network of computers which are interconnected instead of using a client server model. Computers can only be connected to a relatively small number of other computers. Connect to a huge number of other peers is neither efficient nor feasible with the resources available per peer. P2P systems can however grow very large. Therefore not every peer in the network can be connected to all other peers. Because of the large size of current P2P networks, peers are only connected to a very small percentage of the P2P network. Overlay networks offer services for the communication and connection handling between the peers in the network, including peers that are not directly connected to the peer. The overlay network is a layer on top of the network connections, which creates transparent services to higher layers of the software to handle the virtual topology of the network.

Figure 2.1 shows an basic overlay network. Peers keep track of a subset of other peers and can be connected to them. In the example each peer keeps track of some other peers and their IP address. A peer may be in the list but not available, as peer F is in the example. The overlay will use algorithms and knowledge of the network to offer services to other peers. The overlay network facilitates the joining and leaving of a peers. For example the placement among other peers. After initial joining the overlay network will keep communicating with the network in order to keep the network stable and handle network maintenance, depending on the design. The overlay network will maintain a virtual topology and route messages among nodes of the topology.

2.2 Current Overlay Approaches

P2P networks have evolved from the first simple systems like Napster [44] to more advanced system. Different methods to control the structure and distribute content and metadata have been developed.
P2P overlay networks are mostly differentiated in the way the virtual topology looks, is created and maintained, and the way messages are routed between peers. In this section the designs of most important current overlay network solutions are shown. Each of these approaches have different ways of dissimenating data over the network. These forms show concepts from the extreme sides. Mixtures of the given approaches are often used to combine advantages of different approaches. An example of a mixture is the use of Distributed Hash Tables in the central point based system BitTorrent [31, 6] in order to overcome the problem of offline servers.

2.2.1 P2P Systems with Central Point(s)

The simplest way of structuring a P2P system is to have one or more central points. This central point keeps track of the properties, address and content of a number of peers. Upon connecting to the system a peer announces its (new) address and content shared. The complexity of the system is low because the server is the fixed place of return for peers. Another advantage is that the central points can discover peers disappearing which keeps the system stable. Also searching for content is easy. To fetch or search for content the user sends a request to the centralized server. The server then sends back a list of the peers having the requested resources and facilitates the connection and download. The central point is even more useful if it is ran by a trusted source. BitTorrent is an example of a systems using (multiple) central servers as shown in Figure 1.3.
A major disadvantage is that the central server is a bottleneck. The central server needs to be powerful enough to serve requests from all users in the network. In extremely large P2P systems such server requires huge storage, processing and bandwidth. This puts a limit on the scalability of the network. An other problem is the single point of failure of the server; once it disappears the network no longer works. To overcome the single point of failure not one server but multiple servers can be used. These systems are still easy to implement but the problem of scalability still exists. Although the single point of failure is partly overcome, having

centralized servers as the heart of a system is always a threat to the continuity of the system. Each of the servers could disappear and the peers connected to it will also stop functioning. The responsibility for the central servers lies with who runs it, which costs money and can make the medium unavailable to some. Also the owner can put sensor on the content which might be undesirable for some users. It is better to have as few central components in the system as possible. Examples of dominating systems that use (multiple) servers are the eMule network [26], DirectConnect [14] and BitTorrent [11].

2.2.2 Purely Decentralized P2P Systems

The first decentralized P2P systems were fully decentralized. All peers are threated completely equal. Peers connect to a first known peer in the network. They build links with other peers as they encounter them during a session. Connections are created ad-hoc with almost no rules. The peers are unordered as the example in Figure 2.2 shows. When content or information is required most of these system flood the network with request and messages. These messages are forward through the network a number of times. This method creates a lot of overhead. Most unstructured decentralized P2P systems do not scale [8] well because of the low efficiency of the flooding mechanism, although some improvements can be made using smarter routing of messages. One approach is using (multiple) random walks of the network [29]. Another approach is semantic routing [50]; in which queries are only send toward peers which are thought to have interest in the information or answers to the queries. The interesting peer selection is based on previous knowledge of the peers.

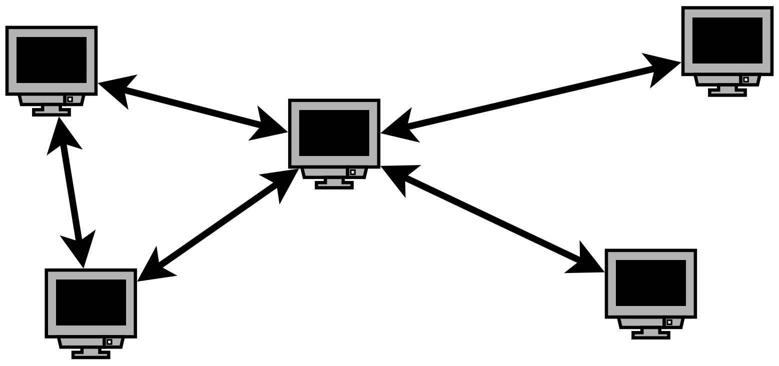


Figure 2.2: An overview of a decentralized P2P overlay network.

Epidemic protocols [5] are useful as an alternative for flooding of information that is not changing very fast. This technique slowly spreads the information across

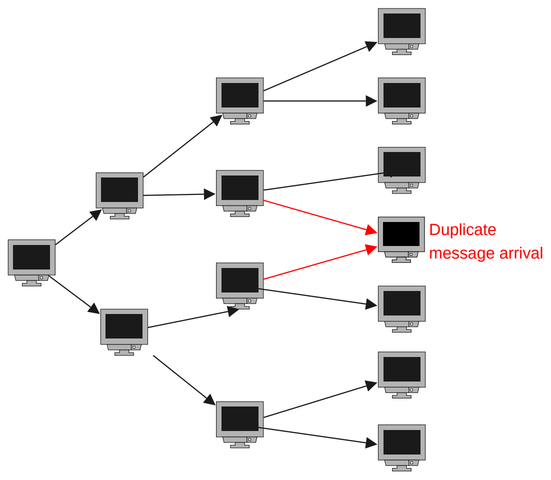


Figure 2.3: Example of epidemic gossiping.

the network using regular small random gossips of knowledge. It is also used in the spreading of information in distributed databases [2]. It is not a search technology but rather a way to dissimulate information across the network. Because the information is forwarded through the network each period it spreads quite fast around the network and duplication is high. The spread is in fact exponential, which makes the system robust against failures. If for example each period is five minute and peers send a gossip to two other peers. The spread an hour after insertion is already at $2^{12} = 4096$ peers, not counting duplicates. An example of a gossip is in Figure 2.3. It shows three forwards. One of the peers receives the message twice in the third period. Although this redundancy wastes resources it makes the method more robust.
The opinions on the scalability of gossiping are divided [12, 24], but is believed to be good, especially when combined with semantic routing [1] and clustering of peers [34].

2.2.3 Distributed Hash Tables

To become more scalable and deterministically find content, networks can be mathematically structured [20] using rules on how to organize the topology. Examples are Chord [46], Content Addressable Network (CAN) [41], and Kademlia [31]. The system maps a key onto a peer, making that peer responsible for the information stored at the key. The keys are spread mathematically equally around all users, so that all user in the structure can potentially be used as a small server for information. This method in essence implements a distributed database. Peers can update or receive information stored at a certain key. A peer requiring to

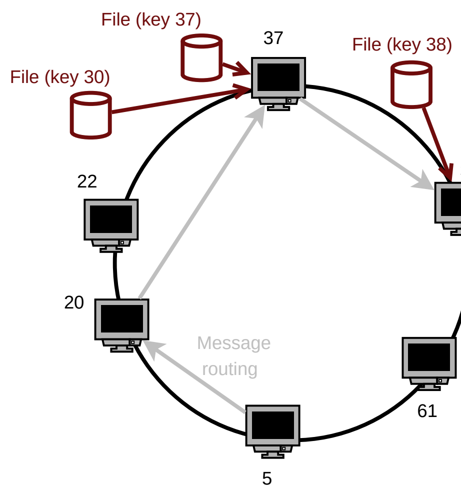


Figure 2.4: An example of a structured P2P overlay network. Simplified overview of Chord [46].

access information at a key must be able to calculate the key. The key can for example be a unique file hash. By routing messages through the structure the peer responsible can be determined. Each peer receiving a request either has the information and replies or it forwards the message to the closest peer it knows near the target. Most DHT systems can route a message to a peer in $O(\log n)$, n being the number of nodes in the system.
In the example of Figure 2.4, the peers are aligned across a virtual circle based on a mathematical distribution, which will evenly spread the peers across the circle. Each peer has an identifier and takes responsibility for the key with the same number. If a peer of a certain number does not exist in the network the first following peer will assume responsibility for the key. For example key 30 will be the responsibility of peer 37. If peer 1 wants to access the information stored at the key 38 it sends a message to peer responsible for key 38, which is forwarded through the network. Because no peer with number 38 exist peer 39 will get the message and handle the request or update.
A major problem of the structured approach is that it only works in stable systems. Upon failing, joining or leaving of peers the structure must be updated or repaired. Maintaining the structure is often difficult and resource consuming. When users are transient the system will be unstable and hard to maintain resulting in an unstable structure and a lot of misrouted messages. Studies have shown that most networks are highly transient [42]. The churn of the network makes the network unstable. Related is that an attack on the network can be very effective. Because attackers know the structure it is easier to attack essential peers. These attacks are called rational attacks [35]. In the example a malicious peer 20 could forward a

message to none existing peers so that peer 1 would not find its target.
Another problem is that load can be high on one or more peers. If a peer is responsible for popular content the requests can overload this peer. This overloading is negative for the function of the peer and the network. The load can be spread by using multiple ways to create keys and thus use multiple peers, but this technique is extremely complex and requires a lot of overhead.

2.2.4 Superpeers

Each peer has different connectivity capabilities, available bandwidth, CPU power and up time. Peers can have different responsibilities in the systems. Some peers can therefore have some special tasks which are handled by a central server in centralized approach. These *superpeers* can act as a servers for a number of ordinary peers. The superpeer technique [33] is a good way of overcoming the problems with transient users and makes the lookup and routing of message easier since a limited number of superpeers can handle these functions. Connections between superpeers can create a network of superpeers which in essence models the functionality of centralized server. These superpeers take over some of the responsibilities of the servers in the centralized P2P system. However once a superpeer fails, its functions and responsibilities will quickly be taken over by other peers. This technique is quite robust, but it puts a high load on a small fraction of peers. Figure 2.5 gives an overview of a network with four superpeers. Each of the superpeers is responsible for a number of peers. An example of a system using superpeers to lookup meta data is Skype [3]. Skype lets users create a list of contacts. By connecting to a superpeer the addresses of peers are determined and a search for users can be sent to the superpeers.

2.2.5 Social Features in P2P Networks

Social networking on the Internet is growing. Social networking websites such as `myspaces.com`, `orkut.com` and `hyves.nl` have millions of users. P2P systems also have been using social features for some time. In most systems groups are created to which a user has to register or meet certain criteria before being able to access the system. These restrictions are often created to fight freeriding and insertion of malicious content. An example is DC++ [14], which runs central configurable servers. Administrators of these servers can define minimal sharing features or even IP address ranges to assure high bandwidth connection and ban freeriding. An other adaption of a system to create social control is that some BitTorrent site require registration. Users who do not obey the rules of the community are banned. These closed communities have better social control than other open parts of the same network. Malicious users can be banned easier and

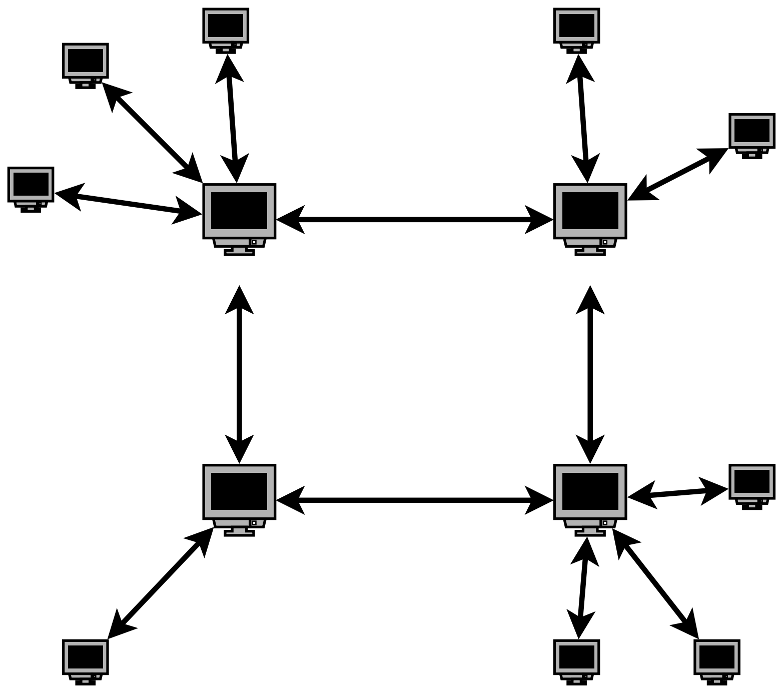


Figure 2.5: An overview of a superpeer P2P overlay network.

the totally utility of the system is increased. These P2P system are not designed for social control, but using some extra mechanisms social control is enforced, which is almost always done by registering IP address and control access to the system by these IP addresses.

Recently P2P systems with explicit social features are introduced. The P2P project Maze [9] is designed, implemented and maintained by an academic research team. The Maze system uses encryption and a central server to control the network and authenticate users. The maze system allows users to add and identify friends. Besides chatting and direct friend sharing it does not exploit other social features such as friends of friends. An other example is Skype [3], a voice-over-IP program using friends and superpeers to communicate. A pure decentralized example is thesocialized.net [7] which bases trust and social relationships on the usefulness of peers in routing messages and sending queries. Only thesocialized.net now explicitly uses social properties to evaluate other peers and determine how to route messages. It also uses similarities in interest to determine possible interesting links between peers.

2.3 General Challenges in P2P Overlay Design

The previous sections mentioned that a lot of pitfalls and problems exist in current overlay networks. Overlay networks can not function well unless they provide scalability, efficiency, availability and integrity. These properties are essential for a P2P overlay network. These are general problems and should be considered with every P2P overlay design. Some of these properties are partly contradictory. A central server for example can increase integrity and efficiency but is not scalable.

2.3.1 Scalability and Efficiency

P2P systems can grow extremely large over time. Current P2P networks connect thousands to millions of users. As the size grows so does the total amount of resources in the system. However the system becomes far more difficult to control. Even though a central server may be able to control such large networks these systems themselves do not scale. Scalability means that a growing P2P network should not have a significant decrease in performance. Adding more users increases the total available resources but may also require more overhead in maintenance and network messages. This problem puts restrains to the possible network size. This overhead must not become too big and result in decreased performance.
In a decentralized network each peer only knows a small portion of the total system but must be able to find content and communicate across the network without having too much increase in latency and bandwidth consumption. As the network grows the traffic to route messages across the network will increase. The protocol of the system must be efficient in order to minimize the overhead which does not directly contribute to the actual content sharing. The design of a P2P system must make sure that the overhead is minimized and the system is scalable [8].

2.3.2 Availability

The total number of users of a P2P system may be very large but not all peers are available at the same time. Most system have a highly transient user base. A study [21] showed that only 3% of the user of a BitTorrent system had an uptime of over 10 hours. Also constantly a lot of users leave the system forever as well as join for the first time. Transient users require peers to connect to different users and reroute traffic. The network structure has to be maintained constantly and a part of the connects or messages will not succeed.
Content is also transient. Most users [39] only share content for a short time and BitTorrent studies have shown that 99% of the P2P bandwidth is provided by only 10% of the P2P community [45]. P2P systems must be designed to overcome the

problems of transient users and content. The transient nature of the users must have the least possible impact on the system as a whole. Misusage of the system must be difficult and altruistic behavior must be encouraged by creating incentives to do so.

2.3.3 Trust and Integrity

Even if a peer is connectable and available the functionality of the overlay network can decrease if some of these peers misbehave. Some users misbehave within the design boundaries of the system. Users of networks tend to be unwilling to share contents or bandwidth. Incentives to share are required for most users. The system must be designed with implicit or explicit incentives to share resources. Some networks for example depend the download speed on the upload speed.
Some peers may also attack the system or disobey the rules. This can make the network unstable. Especially P2P system with a tight structure are vulnerable to these attacks. Repairing and stabilizing the network is difficult and requires a lot of overhead. Systems that require forwarding of messages are vulnerable to these attacks since peers are able to decrease the performance of the network by not forwarding messages or taking out essential points in the network.
A possible solution is evaluation of peers. Behavior of peers can be evaluated and actions can be taken to diminish the impact of bad behavior. Identifiable peers are easier evaluated because past behavior can be taken into account. Peers can earn trust and or have credentials to show its trustworthiness. For trust to work the system needs to be able to correctly identify peers. Peers should not be able to forge their identity or worse take over the identity of other peers. This problem is know as the peerspoofing problem. Due to the relatively anonymous environment of the Internet and the massive size of current P2P system peers can easily take a new clean identity. The whitewashing of history by peers is very difficult and it is therefore easier to evaluate peers on the positive behavior than on negative behavior.

2.4 Specific Research Challenges

The solution to the design of an overlay network discussed in this thesis uses social networks of people as a basis for the topology. In order to create such a topology the system must be able to link real life people to peers. Figure 2.6 gives an example of a social network on the left and a P2P network on the right. This figure shows the main goal of our research. The challenge is to create an overlay network which is able to map the social network onto the P2P network. The system must handle the construction and maintenance. This problem can be divided

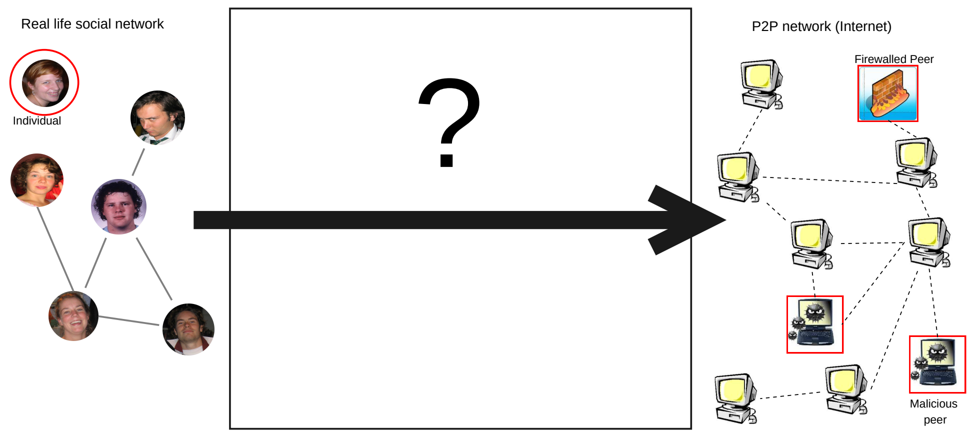


Figure 2.6: From people and their social network to Internet P2P networks.

into two subproblems. First the system must be able to determine to connect to which peers, the social network discovery problem. Second a connection must be set up and maintained to these peers. To be able to make a connection to friends their current address must be found among the possible millions of users. Since the address of peer may change, this is a major challenge. This problem is the IP discovery problem. In this research we would like to research the social network discovery problem and the IP discovery problem. The results should make it possible for the Tribler system to create a social network.

2.4.1 Social Network Discovery Problem

The overlay network is used to determine which peers to connect to. The overlay network may need to connect to certain peers in order to fetch content from that peer. This requires that the overlay knows which peers have the content. The determination of content of peers is not part of the main function of the overlay network and not discussed here. Content and meta information distribution will be handled in higher levels in the software hierarchy.
The overlay network is used to create and maintain the virtual topology. This requires the overlay network to determine autonomously to which peers to connect. A computer can only make a maximum number of connection and each connection creates overhead. It is important that the overlay network creates and maintains connections to a relatively small number of peers so that the operation of the entire P2P network works well. The choice of peers is often determined by the chosen structure of the topology of the network. For example in a pure centralized environment all the peers will only connect to one other peer, the server. In a social network the connections would be to peers in the social nearness of a

peer.
Given that the system would know the people to connect to in most current system peers are anonymous and it is difficult to determine the identity of connected peers. People can counterfeit their identity and make use of privileges of the owner of the identity, a sybil attack [16]. A problem is how to identify peers. Using a trusted third party to enforce identifiers is the easiest and most safe way, but this requires a central component and thus can not be used in distributed P2P environment. In a distributed environment other certification is necessary. Examples are using the IP address and using hardware embedded keys. Both are not very useful nowadays. IP address based is useless because often multiple peers are behind one IP. The absence of hardware embedded keys renders it useless. A distributed solutions is required to identify peers. In order to use social networks a peer must be identifiable so that returning connections can be recognized and buddies can be recognized. If peers are identifiable the system must be able to differentiate between friends and other peers. Given that peers are identifiable methods are required to find friends and allies and build the social network, the social network discovery problem.

2.4.2 IP discovery problem

Once the target peers are clear, connections must be setup and maintained. The system must be able to find the Internet address (IP address) of the peer. Even if a connection has been made before or an IP address is known connecting to a peer is not always easy. Peers are unreliable and can change address. A P2P overlay network must handle the dynamics of the behavior of these peers and be able to (re)connect to peers.

Unreliable Peers

Studies have shown that most peers are not available all the time. Only 3% is available 10 hours or more a day [21]. Also connections can be unstable. Therefore given an internet adress of a peer a connect will not always succeed. Peers are unreliable.

Dynamic IP

Even if a certain peer is online the IP address may change. Some Internet service providers assign IP addresses randomly from a pool of IP address numbers resulting in changing IP addresses. Also the physical location of peers may change. For example in case of mobile clients. In most cases changing the physical location will result in a change of IP address. Discovering the current address of a peer is

important so a connection to transient peers can be made. A lot of solutions exist, for example Mobile IP (IETF RFC 3344 [38]),IPv6 [49] and MobileAgent [27]. But most required significant change of the underlying network.
Once a peer starts a session it should (re)discover the address of other peers in order to connect to them. If peers are anonymous this may require rerunning the algorithm to determine neighbor peers. If the P2P system uses identifiable peers the possible changes of IP addresses must be looked up in order to connect to a known peer.

Unconnectable Peers

Even though the current address of a peer is known and the peer is online the peer may be unconnectable. Many computers are protected by firewalls or are behind NAT routers to share the IP address [18]. If these black boxes in the connection are not properly set up, our connection attempt may be blocked. This problem is however outside the scope of this report.

Chapter 3

A Social P2P Overlay Solution

The approach to the problem of creating a P2P overlay network further discussed in this thesis is to base the topology of the overlay network on a social network. The people in the social network are the peers in the P2P network and the connections are the social links. In order to do so a P2P program must build and maintain the social network. This chapter gives an overview of my solution from real life social network to physical Internet network. In the chapters 4 and 5 two parts of the solution are more elaborately discussed.
Before discussing the building and maintaining of a social network, Section 3.1 discusses useful features of a social network. For a social P2P network to be useful it must first be populated by friends. Peers must add their friends or buddies. The building of a network and maintaining it is discussed in Section 3.2.

3.1 Social Network

Each person has ties and relationships with many people through their social network. The social network is a network of friends, friends of friends (FoFs), friends of friends of friends, and so on. A social network has features that are useful to exploit creating an overlay network. An example overview of a social network is in Figure 3.1. The figure shows the social network of user A, which has four friends B,C,D and E and ten FoFs (F - O). Some of the friends are mutual friends, such as C and B. It also shows that our some of the FoFs can be reached through multiple friends. For example peer M can be reached through peer D and L, B and N or through E.
If the social network is used as a bases for a P2P overlay network friends will be connected to friends. Users are less selfish in their behavior towards friends [37]. They may even be altruistic. Friends also have friends. You trust your friends and your friends trust their friends. These FoFs are therefore probably

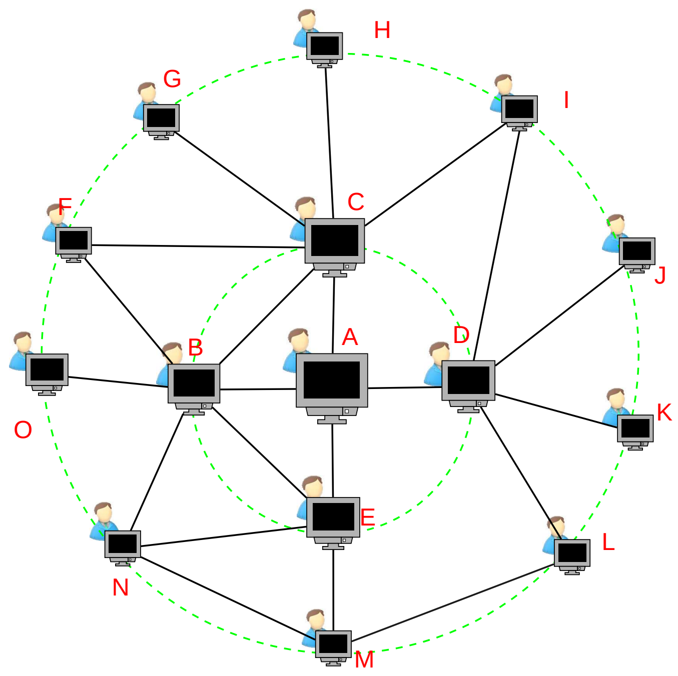


Figure 3.1: A social (P2P) network. The black lines are relationships, the dotted circles are friend levels in the social network of peer A.

more trusted than random other peers. It is suggested [30] that the trust of two peers is dependent on the distance of those peers in the social network. Therefore FoFs are also more likely to behave well. The further the distance between two peers the lower the trust. If misbehaving peers are identified they will probably be removed as friends in the social network and therefore move to the edges of the social network. The creation of the knowledge of the social network around a peer is not part of the scope of this report but is essential in a social P2P network. The implicit trust in the system helps to keep content in the social network correct and keep malicious peers on the edge of a social network. Also people tend to be more willing to share resource towards friends because they expect the favor back or are altruistic in the future which can greatly increase availability.
Social networks are also clustered. People tend to have a lot of friends in common [43]. This clustering is useful because friends are clustered and therefore friends can be used to connect to other friends [47]. Also people with the same interest tend to be connected to a lot of common friends. This clustering of interest can be used since useful content will probably be around users of the same interest [48]. This increases the chance of correct semantic routing. Peers can also expand the social network with peers with similar interests of taste in content and cluster with them in the same way. This would increase the reach of the social network.
Throughout the social network most peers are within a relatively close to each other. Research has shown that two random people anywhere in the world are on average separated by only by a small number of intermediate social acquaintances. The average number of hops in the USA is six [32]. On the Internet, the average number of hops is five to seven [15]. This social nearness makes it possible to connect to almost any peer in the world using only a small number of hops around the social network. Each peer should know its surrounding social network with a number of layers of FoFs. By using the route in the social network almost any peer can be connected. Due to the small number of hops the scalability of such a system can be very good.

3.2 Building and Maintaining the Social Network

The social P2P network building can be seen as three steps:

1. Finding real life networks. From the real life social network to computer social networks.

2. Finding the PermIDs of peers. From computer social networks to identifiable peers.

3. Determining IP addresses. From identifiable peers to individual computers.

These steps are shown in Figure 3.2 based on the social network of the person in the middle of the real-life social network. This figure also shows the potentially connected malicious and unconnectable users.

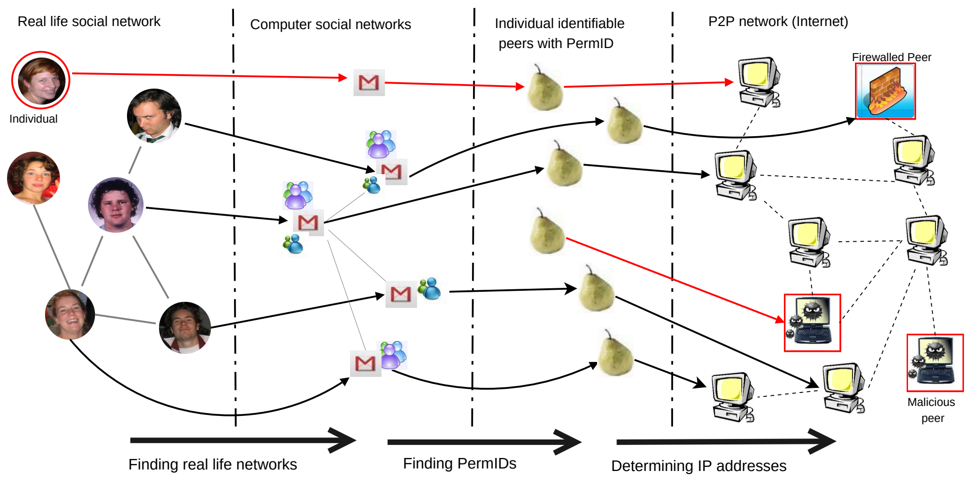


Figure 3.2: The three steps from real life social networks to a P2P computer network with identifiable computer peers.

In order to use a social network in a P2P system the system creates a virtual social network. The challenge is to find a person which is classified as a friend and be able to connect to that peer. This requires identifiable peers and peer must not be able to spoof their identity. So a way to authenticate a user is required. In [25] we introduced permanent identifiers (PermID) for each peer, a strong authentication scheme using a challenge response system. This PermID represents the identity of a peer on the network and peers can verify that identity.
To build the social P2P network users are required to add people to build their social network. The current Tribler system allows users to copy their PermID and give or send it to their friends. These friends can manually add the PermID of this friend to their list of friends. This process is too difficult and ineffective for most users and prohibits a quick adaption of the system by users. Upon first use of the system a user must be able to quickly build up a social network in the Tribler network. This is the bootstrap phase of the social network. Given that peers are identifiable by their PermID the question is how to find the PermID of a person. This enables us to find the PermIDs of friends so a social network can be build. Second we should be able to connect to a peer given the PermID. The Internet Protocol (IP) address and Port number at which a peer is connectable must be found. Our approach uses the social networks to find the IP address of a PermID. As shown in figure 3.2 the building of the P2P social network requires steps from a real life social network to the physical network.

3.2.1 Finding Real Life Social Networks

Although this step can be skipped and people could enter the real life social identities them self automating this step can decrease the time of building a social network. As a lot of current social computer networks already exist with social network information this information can be extracted and used in quickly building a social P2P network. In real life people are member of all kinds of social networks. The people in these networks have unique identities represented by strings and numbers. Some of these are unique for the person, for example email address or phone number. A mechanism that extracts current social networks, from address books for example, can be useful. We designed and implemented modules to extract such information from GMail[1] and MSN Messenger[2], which are discussed in Chapter 4.

3.2.2 Finding PermIDs

The next problem is how to map current real life social networks to the social P2P system. Given a unique value representing our identity we must be able to store information representing our identity on the network and search for identities we know. This bootstrap phase is not frequently used and information change rate is relatively slow. Therefore it is not necessary to have extremely smart and efficient protocols. We propose to spread mappings of identity information to allow the search for the corresponding permanent identifier. It is necessary to be able to send information and to request it. To ensure privacy the information should not be directly revealed. Hash values can be used to hide the private data. The spreading and updating of information can be relatively slow and thus epidemic gossip protocols are a good choice.
We use the social network to store identities and search for ones. The social clustering increases the chance of finding friends as soon as we already have friends. This way the social network can grow incrementally. We create an implementation for the Tribler system, which is discussed in Chapter 4.

3.2.3 Determining IP addresses

Once a peer gets online it should be able to reconnect to its friends. Some peers will be offline and not be connectable. Other peers may have changed addresses. The first problem ca not be resolved, although friends may be altruistic to each other and stay online longer. Thus the social incentive to stay online for friends.

[1]see http://www.gmail.com
[2]see http://messenger.msn.com

The problem of finding changed addresses should and can be resolved. This requires finding an IP address for a PermID. This can be resolved using different technologies such as superpeers, flooding or central servers. Using the bootstrap phase of building a social network we get initial IP addresses of friends and the system can store the IP address if an other peer initiates the connection. The system can always try the last known IP address. If this fails the system must resolve the new IP address. We propose to use the social network. Our solution is to use common friends and friends of friends to store and find the current IP address. A peer coming online should announce its identity and address to all peers in its social network to which it can connect. By exchanging lists of PermIDs and addresses with friends and FoFs the local IP address database of friends and FoFs can be kept up to date. As only one peer in our social network stays online to captured changes we will be able to reconnect to the social P2P network and acquire changes.
The question is whether this will succeed. In order to answer this it is necessary to know the chance a user is offline or the statistical distributing of online users and the the chance and frequency of IP address changes and the time users keep their IP address. We measured the dynamics of IP address, which is discussed in Chapter 5.

Chapter 4

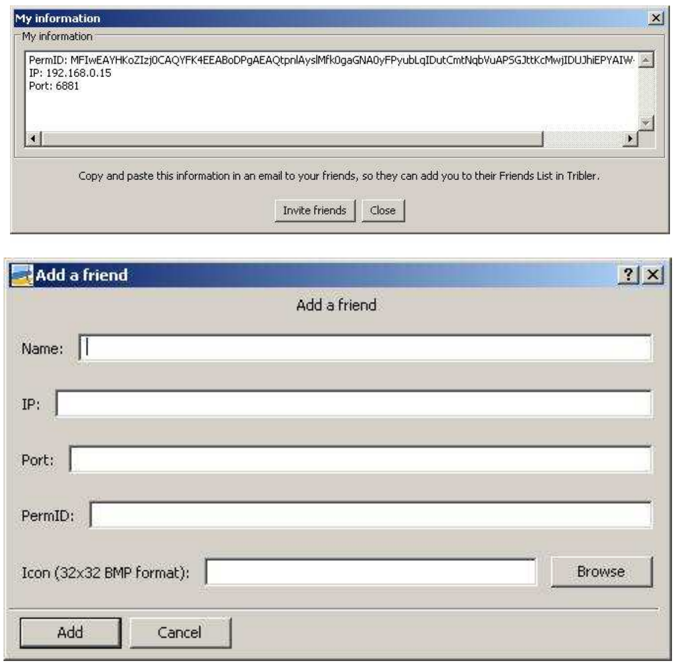
Social Network Discovery

In the previous chapter we outlined the concept of a social P2P network, which consists of relationships between people. In order to use such a P2P network it is required to add friends and a social P2P network with more friends can function better. The social network can be build manually, by adding user permanent identifiers and other data. Figure 4.1 shows the screen shots of the current method in Tribler. But this method is a slow method and difficult for the average user, blocking the quick adaptation of a system.

It is essential for adaptation of a social P2P system that a social network is build quick and easy. Users of a social P2P system must be able to find the P2P identity of their friends, represented by a permanent identifier. A fast and easy way is to enable users to search for their friends by known values, such as values of real life identities. User should be able to insert their real life identity in the network and attach it to their online P2P identity. Social network discovery must be included in every social P2P network.

This chapter discusses an approach based on the social network, which is used to spread information among the peers in the social nearness of a peer by using an epidemic protocol. In case no social network exists superpeer technology is used as a backup. A module has been designed and implemented in Tribler for spreading and searching of identities. Also, example modules have been developed for the extraction of identities of friends from GMail and MSN Messenger in Tribler. These identities can be input for a search on the network.

Section 4.1 gives an overview of the identity search mechanism for the social network discovery problem. The prototype implementation is discussed in Section 4.2. Results of some tests are in Section 4.3. Section 4.4 discusses the solution in this chapter.


My information

PermID: MFIwEAYHKoZIzj0CAQYFK4EEABoDPgAEAQtpnlAyslMfk0gaGNA0yFPyubLqIDutCmtNqbVuAPSGJttKcMwjIDUJhiEPYAIW-
IP: 192.168.0.15
Port: 6881

Copy and paste this information in an email to your friends, so they can add you to their Friends List in Tribler.

Figure 4.1: Current method to add friends in Tribler.

4.1 Design of Identity Search

The goal of the social network discovery is to enable users to find identities of people. This requires users to be able to add their own identity and search for others. The spreading and searching must be quick and easy. The mechanism should also consider that malicious peers could fake their identity, using a spoofing attack, in order to get certain privileges.

As discussed in Section 2.2, different ways exists of spreading and searching information across a network. A social P2P network can exploit social features for this goal. Social networks have social nearness and clustering of peers, therefore user can spread and save mappings around friends. Social clustering means that people have a lot of common friends and thus users can find each other through their friends. The reach of a social network is also very huge. For example, a study has shown that humans have on average a direct social network of 124 people [22] and on average everybody in the world is connect through 5 to 7 intermediate friends[15]. If each of these people would also have a social network of 124 people and only half of the people overlap, two levels in the social network would already consist of $124^2 * 0.5 = 7688$ people. More levels will exponentially grow the reach. Peers can reach each other through the people in their social network. The concept is shown in Figure 4.2. In this figure two layers of the social networks of peer A and E are shown. Peers A and E are said to have four degrees of separation. Peer C is in both and thus can store information from both A and E. It is possible for A to reach for information of E and find the information through C and vice versa. Thus spreading and searching information among two levels of a social network creates the possibility for the system to reach four levels of users. In a crowded network this would mean many users.

We propose exploiting of the social P2P network with an epidemic gossip protocol. Since the social network can become enormous, but is relatively clustered this technology is very efficient and scalable. The social network also has implicit trust among peers. Since friends are more trusted than random anonymous peers the impact and possibility of a spoofing attack is reduced. Identities are spread in the social network and searches are executed in the social network. If a peer already has friends in his network these are requested first in a search. Also friends of friends can be requested because they are also more likely to have shared known contacts. The clustering of friends increases the chance the mapping is found if peers have a common friend which also uses the network. Because not all peers are clustered or have a social network build, superpeers can be used as a rendezvous point. If the search among friends and FoFs does not result in a positive response superpeers are queried. Superpeers store all information they encounter, but since this information is not verified mapping information from superpeers must be considered less trusted.

This solution spreads personal mappings from an identity to a permanent identifier, to allow the search for the corresponding permanent identifier. An identity can be seen as the tuple $(service, value)$, which should point to the correct PermID. To ensure privacy the information should not be directly revealed, instead of the actual values the (sha [17]) hash values are used. The service parameter is a string description of the already existing identity, for example *email*. The inclusion of a service parameter adds to possibility to use the system for different types of real life identities. To spread and request information we propose two new messages to the BitTorrent protocol: `BOOTSTRAP_GET` to request and `BOOTSTRAP_SET` to send identity information. The exact specification of these messages is in Appendix A.

A boost in the speed of building a social P2P network can be accomplished by using existing social networks and extract the relationships as input for the search on the network. Nowadays, a lot of systems exist which imply relationships between persons. Extracting information from these system can give us friends of a person. Examples of these systems are MSN Messenger, Frienster and GMail contact list.

The message size in a test (see Section 4.3) on average consisted for 66% of useful data and the rest is overhead created by the bencoding and dictionary format. Dropping the dictionary values and using an array and a strict order raised that

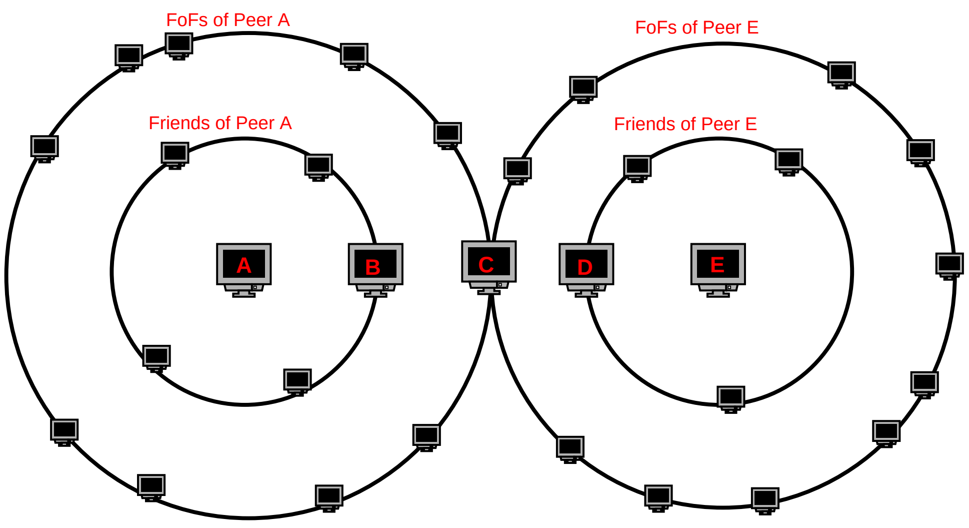


Figure 4.2: Social Network of two Peers.

86%. However, to allow the message specification to be extended in the future and to allow flexibility the dictiories are used in the final design.

4.1.1 Message Exchange

The identity search mechanism continuously runs in the background allowing other peers to request or update information. Every transaction in the system is based on one peer sending a `BOOTSTRAP_SET` or `BOOTSTRAP_GET` message. The system uses the Tribler secure overlay module [40], which makes it possible to send message to specific PermIDs and register to receive messages. Usage of the bootstrap system is done in the following cases:

- User insertion of identity mapping
- User request of PermID based on a real life identity
- Continuous spreading of information. Done by gossiping
- Receiving of information

Each of these cases is clarified in the following subsections.

Insertion of identity data

If a user wants to add information to attach to its PermID at least the name of the service and the value must be provided. The system will attach signatures and

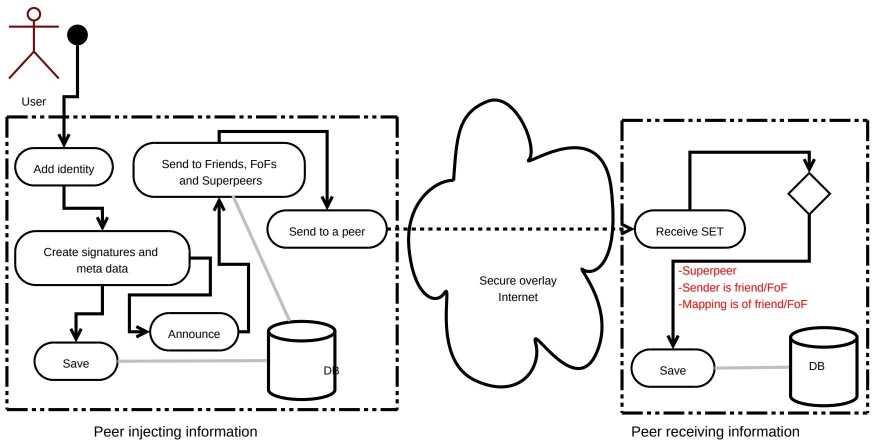

User
Add identity
Send to Friends, FoFs and Superpeers
Send to a peer
Create signatures and meta data
Announce
Save
DB
Peer injecting information
Secure overlay Internet
Receive SET
-Superpeer
-Sender is friend/FoF
-Mapping is of friend/FoF
Save
DB
Peer receiving information

Figure 4.3: Scheme for adding of an identity.

other meta data. After the data is created and stored in a local database the new information is announced to friends, friends of friends (FoFs) and superpeers using the `BOOTSTRAP_SET` message. Figure 4.3 shows this scenario in a diagram.
A peer should only insert mappings that represent his own identity. His P2P identity is unique by his PermID and corresponding private key. To avoid false insertions pointing to a PermID, and thus making distributed denial of service attacks possible, a created identity mapping must be signed by the inserting peer. This signature includes the service and value and because information can be updated the time of insertion is included in the signature. The signature avoids that peers can create false mappings pointing to other peers.
Peers can however insert false identities. They can claim to be the owner of a certain identity while they are not. Without a trusted source it is impossible to avoid this. In order to minimize the impact peers should not store multiple values of the same service for each PermID. Although a identity value can be claimed by multiple peers it is not possible for a peer to claim multiple identities of the same service. For example both Bob and Alice can insert the `email:alice@mail.com` tuple and a request should return both. But if Bob first inserts `email:alice@mail.com` and later `email:bob@mail.com` peers receiving both should discard the oldest. This way a peer can not claim a lot of identities in order to redirect every search to him. Peers should however verify mappings before using them as friends as discussed in Section 4.4.

Request for PermID of real life identity

Users should be able to search for identities of friends. A request or search is done using the $(service, hashedvalue)$ tuple. The value (for example an email

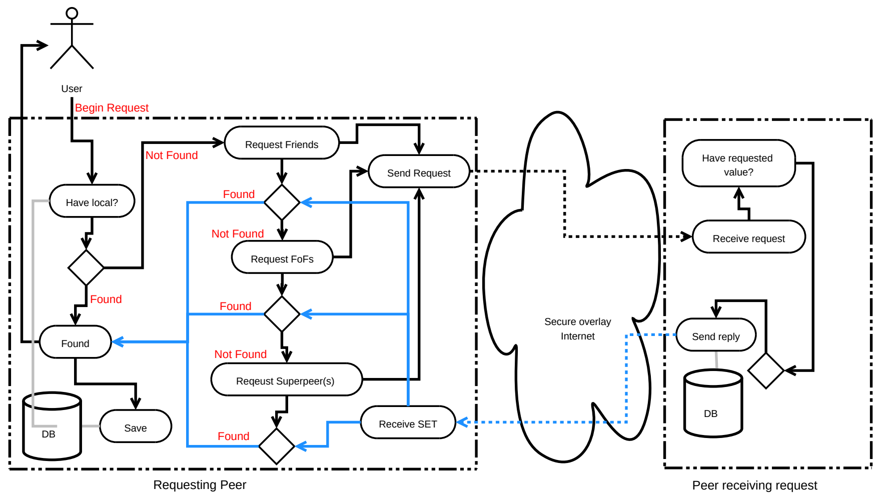


Figure 4.4: Scheme for requesting other peers for an identity.

address) is hashed for privacy reasons. This tuple is used to search friends, FoFs and superpeers using the BOOTSTRAP_GET message. To increase the efficiency each request may consist of multiple tuples. Friends are queried immediately. The load on superpeers should not be too big and a peer can have a huge number of FoFs. To limit the bandwidth usage and peer load FoF and Superpeer searches are queued and at a certain interval a number of these searches is send until all searches are done or a positive result is returned. The scheme for a request is shown as a diagram in Figure 4.4.

The use of friends and FoFs has the advantage of using social clustering of friends as a means to find peers. Also friends are likely to be more trusted than random peers which decreases the impact of malicious clients. Once a friend is found the system can send previous unsuccessful requests to this peer. This way an incremental search is done which includes the growing social network. The system can reach up to four levels in the social network as shown in Figure 4.2.

Upon receiving a request the system looks in its database. If a PermID is found that matches the service and hash, all known mappings of that PermID are returned to the requester.

Gossiping

Information is announced during insertion. The information should however also spread among peers which were not connectable or in the social network at the moment of insertion. The concept of gossiping is used to slowly spread mappings around. At a specified regular intervals gossips are send around the social network. A gossip consists of a number of mappings of friends and FoFs from the

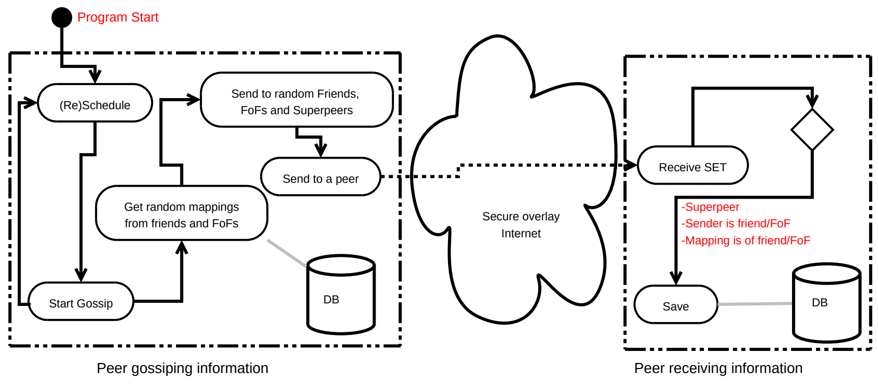

Program Start
(Re)Schedule
Send to random Friends, FoFs and Superpeers
Send to a peer
Get random mappings from friends and FoFs
Start Gossip
DB
Peer gossiping information
Secure overlay Internet
Receive SET
-Superpeer
-Sender is friend/FoF
-Mapping is of friend/FoF
Save
DB
Peer receiving information

Figure 4.5: Scheme for gossiping social mappings.

local database. This information is gossiped to random friends, FoFs and superpeers. Gossiping is an excellent way to distributed slowly changing data such as mappings as discussed in Section 2.2.2. The social surrounding will know everything from the nearby social layers in time. Figure 4.5 shows the diagram of this gossiping.

Receiving of mapping data

Upon receiving of a BOOSTRAP SET message containing information the systems first checks the integrity of the message and the signature. Second the system looks whether request were send for this mapping and notifies the user if that is the case. The system saves the message to the local database in the following cases:

1. The peer is a superpeer. Superpeers store all information
2. Social network:
   (a) The sender of the mapping message is a friend or FoF. The peer stores all information that friends and FoFs are storing.
   (b) The message contains mapping information from friends or FoFs. So that the peers stores all information from the nearby social network.

In each of these cases the information is only saved if no newer information from the same PermID and service exist. It is possible that the peer saves information of the same hash value but this cannot be from the same service-PermID. It is possible that two identical hash values with different PermIDs exist because people can insert false information. If this is the case at least one of the peers is try to fake its identity. Without verification it is impossible to determine the correct mapping.

4.1.2 Rate Control

Bandwidth consumption and the load on peers must be low. The mechanism should create little overhead. In order to control the bandwidth usage peers may only send messages to other peers once every number of seconds. Also the message may contain a maximum number of mappings or request. If the thresholds at the receiving peer are exceeded messages are dropped. If a request is launched by a user and the number of mappings is larger than the threshold the message is split up and queued.

4.2 Prototype

The basic bootstrap protocol can be used for different real life identities. As part of this thesis a prototype is implemented in Tribler[1]. The bootstrap protocol is implemented as a python module called BOOTSTRAPMANAGER. The modules for extracting the existing social networks are created as similar black boxes. Each of these modules should have a similar function which accepts the username and password for an existing account and returns a list of people represented by service, value and optional a name field. As part of this thesis black boxes are created for the extraction of friends based on contact lists from MSN Messenger Contact List and Gmail Address book.

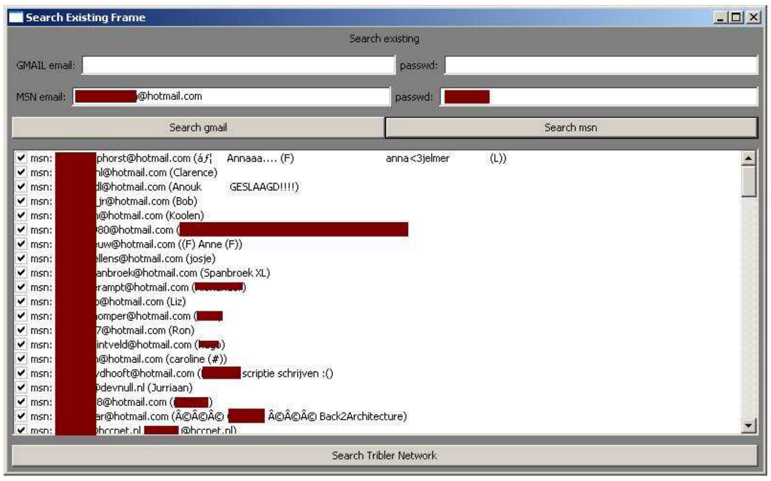
4.2.1 MSN

An adapted version of the MSNP package[2] is used to connect to an existing MSN Messenger account and extract contact information. The package connects to the MSN network and acts as a client. Once the contact list is acquired from the server, it is returned. In the prototype this is shown as a list which can be searched for on the Tribler network. Figure 4.6 shows an example of the prototype interface. The service is called `msn` since these are msn friends.

4.2.2 GMail

To extract the contact list from webmail accounts of gmail we use the libgmail module [3]. This module acts as a interface to connect to the website of gmail. It allows the extraction of contacts from the contact list. The module had to be slightly adapted in order to function some new functions of gmail as well. As with

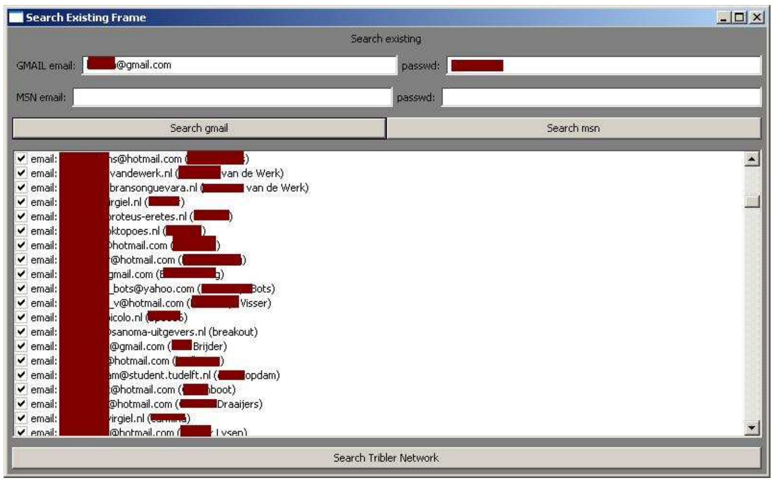
[1]Tribler branch of version February 16,2006
[2]http://msnp.sourceforge.net
[3]http://libgmail.sourceforge.net



```
Search Existing Frame
Search existing
GMAIL email:                                passwd:
MSN email: [redacted]@hotmail.com           passwd: [redacted]
[Search gmail]  [Search msn]
✓ msn: [redacted]phorst@hotmail.com (áƒ¦   Annaaa.... (F)            anna<3jelmer      (L))
✓ msn: [redacted]l@hotmail.com (Clarence)
✓ msn: [redacted]dl@hotmail.com (Anouk      GESLAAGD!!!!)
✓ msn: [redacted]_jr@hotmail.com (Bob)
✓ msn: [redacted]@hotmail.com (Koolen)
✓ msn: [redacted]80@hotmail.com ([redacted]
✓ msn: [redacted]uw@hotmail.com ((F) Anne (F))
✓ msn: [redacted]llens@hotmail.com (josje)
✓ msn: [redacted]anbroek@hotmail.com (Spanbroek XL)
✓ msn: [redacted]rampt@hotmail.com ([redacted])
✓ msn: [redacted]@hotmail.com (Liz)
✓ msn: [redacted]omper@hotmail.com ([redacted]
✓ msn: [redacted]7@hotmail.com (Ron)
✓ msn: [redacted]intveld@hotmail.com ([redacted])
✓ msn: [redacted]@hotmail.com (caroline (#))
✓ msn: [redacted]vdhooft@hotmail.com ([redacted] scriptie schrijven :()
✓ msn: [redacted]devnull.nl (Jurriaan)
✓ msn: [redacted]8@hotmail.com ([redacted])
✓ msn: [redacted]r@hotmail.com (Â©Â©Â© [redacted] Â©Â©Â© Back2Architecture)
✓ msn: [redacted]hccnet.nl [redacted] @hccnet.nl)
[Search Tribler Network]
```

Figure 4.6: Prototype of MSN identity extraction interface.

MSN contact the email addresses are shown in a list and can be searched on the Tribler network. The contact details are only email addresses and therefore the service is `email`. Figure 4.7 gives an example of the prototype user interface after an extraction.

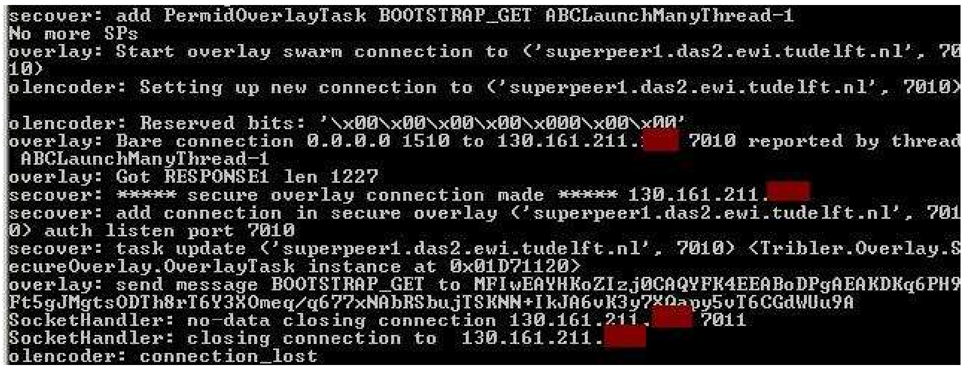


```
Search Existing Frame
Search existing
GMAIL email: [redacted]@gmail.com           passwd: [redacted]
MSN email:                                  passwd:
[Search gmail]  [Search msn]
✓ email: [redacted]s@hotmail.com ([redacted])
✓ email: [redacted]vandewerk.nl ([redacted] van de Werk)
✓ email: [redacted]bransonguevara.nl ([redacted] van de Werk)
✓ email: [redacted]rgiel.nl ([redacted])
✓ email: [redacted]roteus-eretes.nl ([redacted])
✓ email: [redacted]ktopoes.nl ([redacted])
✓ email: [redacted]hotmail.com ([redacted])
✓ email: [redacted]@hotmail.com ([redacted])
✓ email: [redacted]gmail.com ([redacted]g)
✓ email: [redacted]_bots@yahoo.com ([redacted]Bots)
✓ email: [redacted]_v@hotmail.com ([redacted] Visser)
✓ email: [redacted]icolo.nl ([redacted])
✓ email: [redacted]sanoma-uitgevers.nl (breakout)
✓ email: [redacted]@gmail.com ([redacted] Brijder)
✓ email: [redacted]hotmail.com ([redacted])
✓ email: [redacted]m@student.tudelft.nl ([redacted]opdam)
✓ email: [redacted]@hotmail.com ([redacted]boot)
✓ email: [redacted]@hotmail.com ([redacted]Draaijers)
✓ email: [redacted]irgiel.nl ([redacted])
✓ email: [redacted]@hotmail.com ([redacted] Lysen)
[Search Tribler Network]
```

Figure 4.7: Gmail identity extraction prototype interface.

4.2.3 Bootstrap Handler

The BOOTSTRAPHANDLER class implements the actual handling of messages and storing and searching of identities. It implements the mechanisms as dis-



```
secover: add PermidOverlayTask BOOTSTRAP_GET ABCLaunchManyThread-1
No more SPs
overlay: Start overlay swarm connection to ('superpeer1.das2.ewi.tudelft.nl', 70
10)
olencoder: Setting up new connection to ('superpeer1.das2.ewi.tudelft.nl', 7010)

olencoder: Reserved bits: '\x00\x00\x00\x00\x000\x00\x00'
overlay: Bare connection 0.0.0.0 1510 to 130.161.211.[redacted] 7010 reported by thread
 ABCLaunchManyThread-1
overlay: Got RESPONSE1 len 1227
secover: ***** secure overlay connection made ***** 130.161.211.[redacted]
secover: add connection in secure overlay ('superpeer1.das2.ewi.tudelft.nl', 701
0) auth listen port 7010
secover: task update ('superpeer1.das2.ewi.tudelft.nl', 7010) <Tribler.Overlay.S
ecureOverlay.OverlayTask instance at 0x01D71120>
overlay: send message BOOTSTRAP_GET to MFIwEAYHKoZIzj0CAQYFK4EEABoDPgAEAKDKq6PH9
Ft5gJMgtsODTh8rT6Y3X0meq/q677xNAbRSbujTSKNN+IkJA6vK3y7X0apy5vT6CGdWUu9A
SocketHandler: no-data closing connection 130.161.211.[redacted] 7011
SocketHandler: closing connection to  130.161.211.[redacted]
olencoder: connection_lost
```

Figure 4.8: Request to Superpeer on Tribler Network.

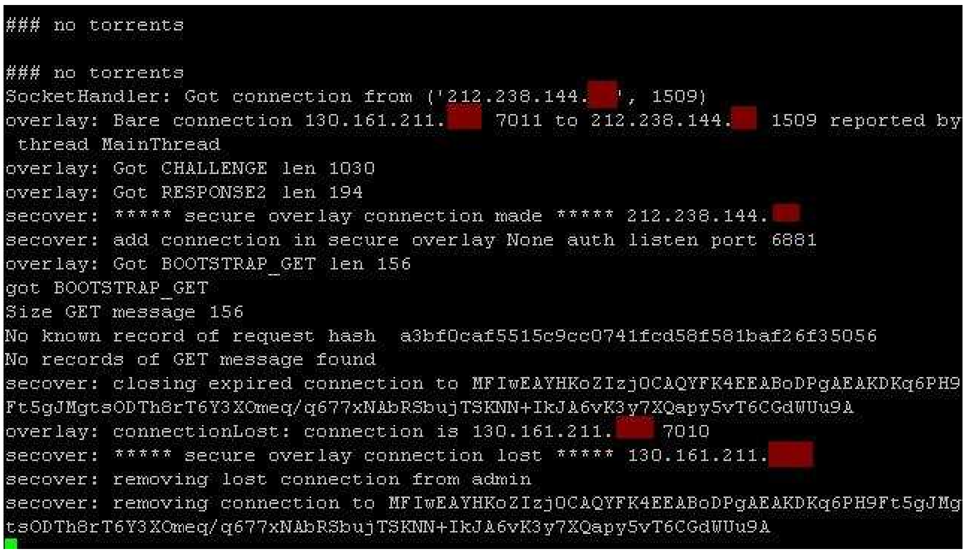
cussed in this chapter. The user can insert information of his identity, search for others and the class gossips identity information around. The class accepts the messages BOOTSTRAP_GET and BOOTSTRAP_SET. Figure 4.8 shows a command line log output of a search.

4.3 Tests

We tested the system in a small scale test environment. Example output on a superpeer is shown in figure 4.9. A search resulting in a found PermID is shown in figure 4.10 .

We created a fake social network with a superpeer having 400 known PermIDs with each four identity services. We ran ten instances of a client, which were all friends. In this test the PermID, IP address and port of friends were known. Each of the friends insert their (random) values for the four identity services. This information is spread immediate among friends and the superpeer. Each of the peers also randomly searches 20 of the 1600 ($= 400 * 4$) possible identity representations with an interval of ten minutes. After all the test runs each of the peers knows all its searches.

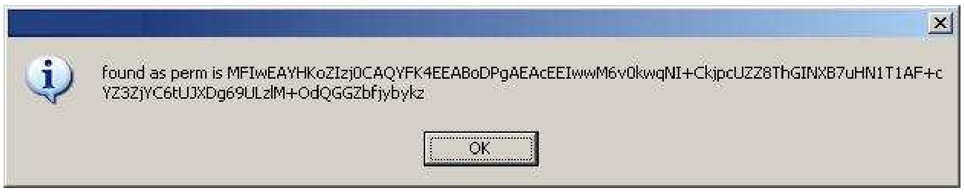
In order to test whether the superpeer could handle a lot of requests, we also ran the ten friends constantly sending random requests to the superpeer. After a message was send another one was send directly hereafter. On the superpeer rate limits were disabled to allow receiving of all messages. The superpeer could handle the continues stream of messages and a reply was send on every request.



```
### no torrents

### no torrents
SocketHandler: Got connection from ('212.238.144.[redacted]', 1509)
overlay: Bare connection 130.161.211.[redacted] 7011 to 212.238.144.[redacted] 1509 reported by
 thread MainThread
overlay: Got CHALLENGE len 1030
overlay: Got RESPONSE2 len 194
secover: ***** secure overlay connection made ***** 212.238.144.[redacted]
secover: add connection in secure overlay None auth listen port 6881
overlay: Got BOOTSTRAP_GET len 156
got BOOTSTRAP_GET
Size GET message 156
No known record of request hash  a3bf0caf5515c9cc0741fcd58f581baf26f35056
No records of GET message found
secover: closing expired connection to MFIwEAYHKoZIzj0CAQYFK4EEABoDPgAEAKDKq6PH9
Ft5gJMgtsODTh8rT6Y3XOmeq/q677xNAbRSbujTSKNN+IkJA6vK3y7XQapy5vT6CGdWUu9A
overlay: connectionLost: connection is 130.161.211.[redacted] 7010
secover: ***** secure overlay connection lost ***** 130.161.211.[redacted]
secover: removing lost connection from admin
secover: removing connection to MFIwEAYHKoZIzj0CAQYFK4EEABoDPgAEAKDKq6PH9Ft5gJMg
tsODTh8rT6Y3XOmeq/q677xNAbRSbujTSKNN+IkJA6vK3y7XQapy5vT6CGdWUu9A
```

Figure 4.9: Example output on peer after recieving GET message.



found as perm is MFIwEAYHKoZIzj0CAQYFK4EEABoDPgAEAcEEIwwM6v0kwqNI+CkjpcUZZ8ThGINXB7uHN1T1AF+c
YZ3ZjYC6tUJXDg69ULzlM+OdQGGZbfjybykz

Figure 4.10: Prototype Pop-up Result of Tribler Network Search and Found Per-mid.

4.4 Discussion

The gossiping of information will slowly spread all information. As the information is in injected the P2P system it will probably be saved by at least the superpeers but a lot of other peers might known the information as well. A search for an identity is very likely to get results if the search value is in the system. The biggest issue in this solution is that everybody can insert false identities. A solution could be to introduce a trusted third party. Every user should sign up and prove its identity to the third party so that searches can be verified. The central server could sign verified identities and give the signature to the user. This makes the user able to prove its claim about his identity. This however requires a central server and thus has scalability issues. Also the question is who would run and be responsible for this server. The owner should be trusted and willing to invest without any return. Therefore a decentralized verification solution is required in a global social P2P network.
In order to prevent malicious insertion of identity values we use signatures. We can always confirm that the inserted value is done by the PermID it signed the values with. The PermID prevents peer spoofing. However a person may fake its identity. For example Bob may insert a service email with value *alice@mail.com*. The common way (used by MSN, Skype,Hyves) of verifying is to use direct verification. The final implementation of the bootstrap search should include some form of direct verification. Our social network however can aid to make a more efficient social based verification possible.

Direct Verification Direct verification means that the mapping of identity to PermID is directly verified with the owner of PermID. This could be for example by mouth or telephone. Most current systems use a form of social challenge-response. Once a presumed person is found a challenge is send to that person. For example a question only that person could know. If that person responds with the correct answer its identity is considered verified. The users of the system are responsible for correct verification. For good conclusive verification the challenge response should be send and replied in both ways between two peers requiring four messages. This makes the verification process lengthy and complex and it is therefore also possible to use smaller number of messages. For example Skype [3] uses only one response message. This makes the responsibility of the end-user bigger but decreasing complexity of the verification process.

Verification by friend Most social networks have common friends. If some of the already verified friends have verified common friends we could assume their identities as already verified. A mapping could include credentials signed by our friends that these mappings are correct. Of course the level op trust in this verification must be lower than that of direct verification. This could however decreases the number of direct verifications dramatically as the social network grows.

Chapter 5

IP Discovery

Building the social network, as discussed in Chapter 4, is the first step in creating a social P2P network. The second step is creating and maintaining Internet data connections between peers. A data connection must be created between friends in the network, and thus peers must be able to connect to the current IP address of their friends. As outlined in Section 2.4.2 connecting to peers is problematic due to low peer reliability, peer unconnectability, and unstable IP addresses.
Unconnectability is outside the scope of this thesis, but several promising techniques exist to overcome the problem [18]. Online time measurements are available [21, 39]. These studies conclude that the majority of the peers are unavailable most of the time. However, to the best of our knowledge no real IP address change analysis has been done. In order to fill this gap the IP address change dynamics of peers has been studied as part of this thesis and is discussed in this chapter. The first dataset is based on a BitTorrent community, of which the peers were monitored for about a week. The second dataset is based on four months of observed data from the bootstrap peers in the Tribler network. Based on the two datasets can be concluded that IP address of peers are not very dynamic.
Section 5.1 will first discuss and explain the measurement on a BitTorrent community and an analysis of the Tribler network in Section 5.2. The results of both analyses are combined in Section 5.3. Finally, the results of this chapter are discussed in Section 5.4.

5.1 BitTorrent Community Measurement

The question is how dynamic IP addresses of peers are, how often peers in P2P networks change their IP address, and how available they are. An analysis should be done on a P2P system. We wanted to do a measurement on a BitTorrent community which has identifiable users, because this way users can be tracked across

From HTML	From Peerping
SwarmInfoHash	SwarmInfoHash
Nickname	
Connectable	Internal
Percentage	Pieces
Time - Idle	Time Of Update
Clientname	(Part of) PeerID
	IP Port

Table 5.1: Data from HTML Scraper and Peerping module that overlap.

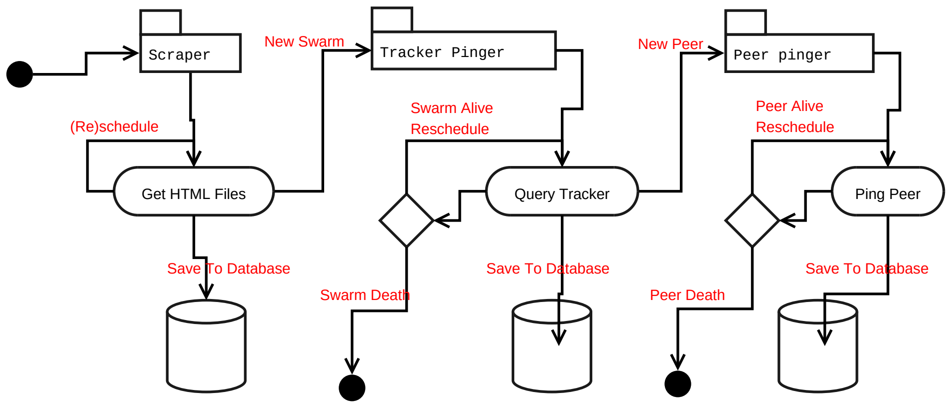


Figure 5.1: Scheme of BitTorrent community measurement.

sessions. A measurement has been done on a BitTorrent community (`fileporn.org`) with around 90,000 registered users. This closed community has fixed nicknames with which we can identify people. Statistics of the download behavior of these users are shown on the website. Statistics are shown for each swarm. On the other hand the IP addresses are distributed by the tracker. Our first attempt was to measure `filelist.org`, a community with more diverse content, but the administrators unfortunately removed statistics during the development of our measurement software.

5.1.1 Measurement Methodology

Given that we would have the IP addresses of nicknames at certain times we could calculate statistics about changing IP addresses. Unfortunately the system does not provide these directly for privacy reason. However, since we can also poll in-

dividual peers for their progress and progress data is also available on the website we can try to link the data from the website and the peers. Information is received from three sources: the website HTML scrapes, from tracker request and from individual peer request (PeerPing). The two sources having the overlap in information, the nicknames and IP address information are the HTML scrapes and the peerpings. The overlap in the two sources is shown in Table 5.1
Two pieces of software were used to monitor the HTML pages on the one hand and the tracker and peers on the other hand. The scheme for this measurement is shown in Figure 5.1. The HTML page scraper used is an adaption of the system by Roozenburg [43]. The scraper downloads the torrent file and pages of individual swarms and saves the results on average every five minutes, but times may differ due to resource limitation on the system. Not all information on the HTML files is important. Most of it are layout,comments or annotations. Although all HTML files are saved for each swarm, the following information is filtered per user per scrape moment:

- SwarmInfoHash: unique hash value of the swarm
- Time: time of the HTML scrape
- Nickname: the nickname in the website system
- Connectable: Yes if the peer is connectable and No if it is not
- Percentage: percentage of the completion of the download
- Idle: time since last progress update received from the peer
- ClientName: name and version of the client

The torrents saved by the HTML scraper are input for the Peerping module. The torrents are equally spread among eight parallel running tracker and peerping clients. The tracker requester and peerping clients we developed gather information based on the torrent files. These clients poll the tracker every 15 minutes for new peers. Every known peers is also polled every 10 minutes if it is a still downloading (leecher). Because seeders do not change value they are only polled every 30 minutes to see whether they are still available. Each time a peer is connected, an update is requested by the PeerPing client and the following information is stored in a file per swarm:

- Time: time an update is received from the client
- IP and Port: IP address of peer and the port used by the client

- Pieces: number of pieces completed
- Internal: I if the peer was ping from the system or E when the connection originated from the other peer
- PeerID: 20 character self chosen name of peer. First eight characters are a code for the client name.

After collection of the data the raw data is matched as discussed in Section 5.1.3.

```
Fubar4004  Yes 100.00% 1d07:53:15 16:22 Azureus/2.4.0.2  1149609549
LordSilent Yes 100.00% 1d00:52:05 21:13 uTorrent/1500    1149609549
77nn77     No  100.00% 16:21:57   20:27 uTorrent/1500    1149609549
maciek82   No  100.00% 6:48:47    7:44  BitComet/0.66    1149609549
SVAMPEN    No  100.00% 3d18:32:00 19:27 uTorrent/1500    1149609549
whitelight No  93.26%  1:28:54    0:32  Azureus/2.4.0.2  1149609549
qwerty2r3r No  32.57%  1:02:41    3:35  BitTorrent/4.1.2 1149609549
xip        Yes 20.79%  3:49:30    10:41 Azureus/2.4.0.2  1149609549
karelcaca  No  12.84%  1:54:38    3:16  BitTorrent/4.1.2 1149609549
```

Output 1: Example HTML scrape data (nickname, connectable, percentage done, time, idle, client, and scrape time).

```
1149544820 87.5.214.61    49177 1409 I -AZ2402-GUCZPdrZr5wR
1149544828 193.77.246.216 81    1409 I -AZ2402-q89g9KhHHQ6o
1149544897 81.228.26.36   59612 1409 I -AZ2402-5ex5ADHouMQz
1149545179 75.10.66.5     55170 1409 I -AZ2402-IILOboA0hp5p
1149560219 212.1.157.98   49153 38   I M4-4-1--75fcedf62121
1149554372 87.5.214.61    49177 1409 I -AZ2402-GUCZPdrZr5wR
1149554550 80.202.215.133 11704 1409 I -BC0061-AsR22eeBsjwq
1149554558 88.108.172.45  41952 1409 I M4-4-1--04df5e442baa
1149554559 84.230.152.223 18760 1409 I -AZ2402-wu1jHaVnGOgz
```

Output 2: Example peerping data for swarm (time, IP address, port, internal, and PeerID).

5.1.2 Dataset

The dataset contains seven days of data. Only seven days are available because the systems bans non-contributing members such as the scraper client used. This contains information from 796 swarms of the community. The information is stored in log files. Output 1 shows a small subset of data collected from the HTML pages. In output 2 some lines of the peer measurement are shown. It must be noted that the PeerID is something like: ’-AZ2402-q89g9KhHHQ6o’, where the first eight characters represent the client.

The dataset may contain wrong information since the information is provided to the system by users. However, social control is high in the closed community and malicious users are banned quickly. Since the up and down data must be in balance a faulty swarm can easily be identified and false information is low.

5.1.3 Matching HTML Scraper and Peerping

The two sources of information are HTML and peerping. No direct relationship between the sets exists. It is necessary to connect them. The matching between HTML scraper results and peerping results is based on matching of similar data. For example percentage can be calculated from the number of pieces divided by the total number of pieces from the torrent file. The matching is done in three steps:

1. Inner swarm possibility selection
2. Inter swarm possibility selection
3. Combining inner and inter swarm results and selecting most likely results

The matching client scans each swarm and looks for possible peerpings matches for each nickname in the HTML Scraper data. The client also looks for nicknames that were in multiple swarms at the same time and for peerping IP addresses that were also in the same number of swarms during that time. These two steps in the matching both result in possible peerping peers for each nickname. As a third step the system combines the two results and if possibles chooses a match between the nickname and peerping peer. It was possible to match on average 43% of the nicknames to peerid(s) in each swarm. Of the nicknames 4% did have possible matches but a clear choice could not be made. On average 63% of the nicknames were unidentifiable because no possibilities existed. This is mostly because peers are seeders all the time or unconnectable due to network restrictions such as firewalls. Due to the lack of information, peerping data from these peers can not be acquired and no link between the sets can be created. Of the unidentifiable nicknames, 72% was always a seeder or unconnectable. These results are shown in Table 5.2.

Inner swarm possible matches

The idea of inner swarm matching is to create lists of possible PeerIDs for each nickname in the system. This is done per swarm. The system does so by taking two consecutive occurs of a nickname in a swarm from the HTML set. All possible peerping results that can are in range of the two consecutive occurs are

Match found	43%
Ambigous matching	4%
No match found	63%, of which 72% was either seeder or unconnectable

Table 5.2: Average percentages matching results per swarm for the BitTorrent measurement

selected. The systems first takes all peerping results between the time of the two HTML results. This includes many impossible results. The results are filtered by client name, percentage and connectability. This selecting is done for every two consecutive measures. The time between two occurs in the HTML set is five minutes. This is quite short since peers are only polled every 15 minutes. Therefore, if no results are found the five minute window is widened. Instead of taking two direct consecutive occurs the system can take three or more consecutive occurs and take the outer two as boundaries. The pseudo algorithm of this selection for each individual swarm is shown in Algorithm 1.

Algorithm 1 Inner Swarm

```
HtmlPeers := Read()
PeerpingPeers := Read()
for all  pp IN PeerpingPeers do
   if PP with similar IP,Port,Client,Increasing Percentage Done then
      MERGE
   end if
end for
for all TP IN HtmlPeers do
   occurs := TP.data
   window=0
   while Not found possibles do
      window =+ 1
      for all $T_n$,$T_m$ IN occurs SUCH THAT $T_m$ = $T_{n+window}$ do
         PossibleTemp := $\{Peerpingpeer(s) \| Swarm, T_m \leq T_{peerping} \leq T_n\}$
         PossibleTemp := Filter(PossibleTemp, $\%T_m \leq \%Peerping \leq \%T_n$)
         PossibleTemp := Filter(PossibleTemp, $TPClient \equiv PeerPingClient$)
         PossibleTemp := Filter(PossibleTemp, $Connectable \vee (Unconnectable \wedge external)$)
         Possibles := $Possibles \cap PossibleTemp$
      end for
   end while
   TP.InternalPossibles = Possibles
end for
```

Inter swarm possible matches

Since some peers download from multiple swarms at the same time these peers must occur in both HTML and peerping data. The system sweeps over all the HTML files in order to find these multiple swarm users and saves the information of each occur of a nickname and its timing. It also counts different amounts of overlaps and the timing. A peer may for example be in five swarms during one period and in only three during an other period. The results are also filtered for matching percentages, connectability and client names.
Overlaps can be found on multiple levels, the user can be in more than two swarms at the same time. Because overlap on low levels a common and a lot of peers have similar low level overlaps a selection is done. Criteria for saving the overlaps are:

1. If only one overlapping IP address with PeerIDs found

2. If minimal five level overlaps save all overlaps from level five and higher. This is because the results showed that overlap levels up to 3-4 is still high with a lot of different IP addresses at the same time and therefore not likely. Starting from level five overlap not much collisions occur.

3. Minimal level three and the most occurring overlap overall is also the only overlap in the maximal level overlap

4. Minimal level three and more than one overlap in highest level and only one in highest overlaps. If the highest overlap is most in highest level use it.

Combining Inter and Inner swarm matches

For each swarm the most likely matches between PeerPing and nicknames are saved. This is based on inner swarm and inter swarm possibilities. All nicknames of a swarm are put in a queue and the system checks for the most likely matches of each nickname based on the following criteria:

1. Inter - Inner swarm match:

   (a) If only one (same) match in both inner and inner match.

   (b) Intersection of inner and inter swarm possibilities gives one result

   (c) Maximum match in inner swarm is also in inter swarm match

2. checkOneMatch: Find nickname with only one possible peerping and that possibility has been identified at minimal two occurrences.

3. checkMoreMatch: If the most occuring match is at least twice as much as the second most. And that possibility has been identified at minimal two occurrences.

4. removeZeroMatchResults: Remove nicknames that do not have any possible match and mark those as 'Unidentifiable'

If a match is found the match is saved, the nickname is removed from the queue and the peerid is remove from all other possibility lists. The system sweeps over all nicknames multiple times until no more resolves are done. This process is repeated for every swarm. This results in a list of PeerIDs, including IP addresses used by thoses peerids, for each nickname. From these results a list of time, IP address and port at which a nickname is seen can be created, which are discussed in Section 5.3.

```
1160352436.415  82.157.141.59   6881
1160367186.879  82.157.141.59   6881
1161085908.742  82.157.141.59   6881
1161104614.420  82.157.141.59   6881
1161123541.294  82.157.141.59   6881
1161138517.617  82.157.141.59   6881
1161158705.957  82.157.141.59   6881
1161173262.847  82.157.141.59   6881
1161188092.717  82.157.141.59   6881
```

Output 3: Tribler data containing timing and IP address data for a PermID

5.2 Tribler Analysis

The BitTorrent measurement, as discussed in the previous section, is the first source of IP address change data. The second source is from the users of the Tribler system. The Tribler system uses bootstrap peers to enable new users to quickly be able to use the system. From data of the bootstrap peers, we observed the IP address and PermID data of users in the Tribler network. PermIDs are fixed for an installation of the Tribler client and PermIDs are also unique. Thus, because the data shows information for each PermID it is possible to extract the different IP addresses of a PermID and thus of a user. Output 3 shows some example data. During creation of the results it was clear that some people run multiple instances of the client at the same time with the same PermID but from different IP addresses. This probably caused by copying the installation to multiple computers. Because the goal of the analysis is to see changes of IP address for one client this kind of behavior blurs the data. These peers are also filtered from the results. If

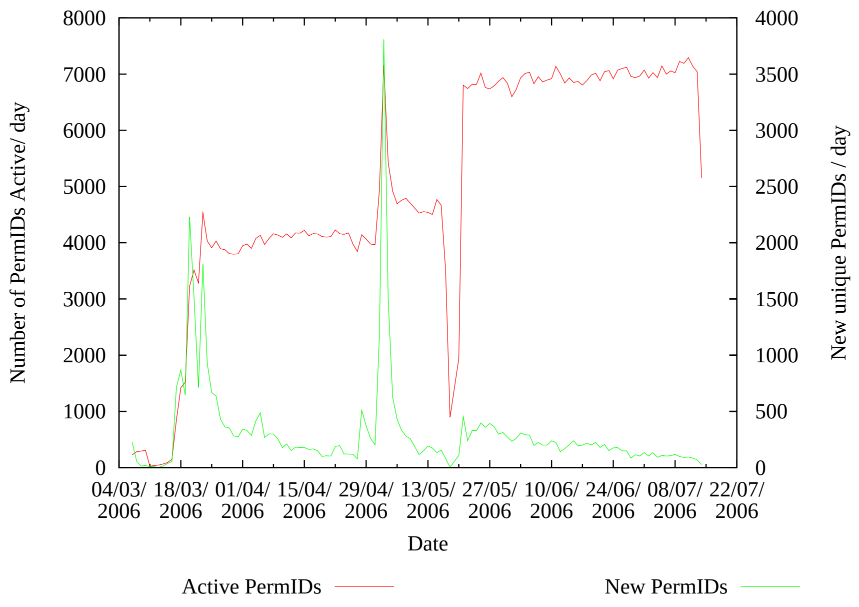


Figure 5.2: New arrival of PermIDs and active PermIDs in the Tribler network.

a peer switches back and forth a number of times between IP addresses within a short amount of time it is very likely that the user runs multiple instances.
Figure 5.2 shows the number of active PermIDs and new arrivals on the Tribler network. Showing almost an continuous measurement for $4\frac{1}{2}$ month. The graph shows some peaks and a gap. The peak around May 2th is due to Tribler press coverage. The gap in both graphs is caused by a bootstrap peer crash on May 18th. The strange increase after the server crash comes to together with the hosting of an open source movie ”Elephants Dream”[1] by Tribler. This probably caused an increase in users of Tribler.

5.3 Results

Both the BitTorrent community measurement and the Tribler analysis provide similar lists that contain IP address and timing information per identifiable peers. From these lists information about the dynamics of IP addresses of peers can be extracted or calculated. The results of IP address change behavior are shown in the next subsection. From the BitTorrent community measurement it is also possible to extract availability and connectability data, which are briefly discussed as well.

[1]Elephants Dream: http://orange.blender.org/

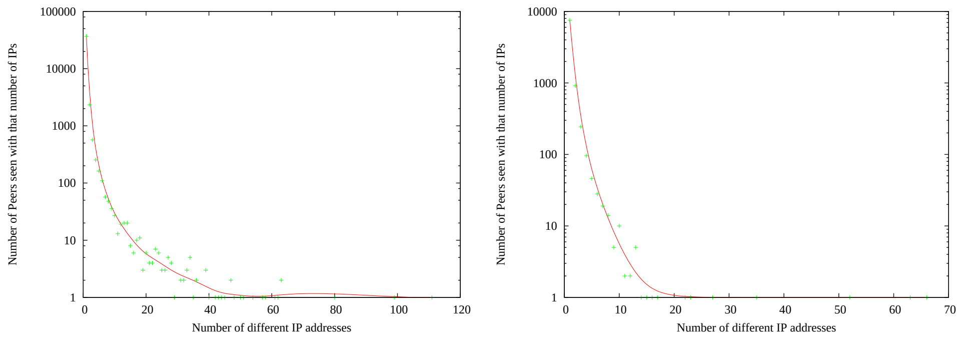


Figure 5.3: Number of Peers with number of IP addresses. The left figure shows the Tribler network, the right figure the BitTorrent Community Network. The lines are Bezier approximations of the data points.

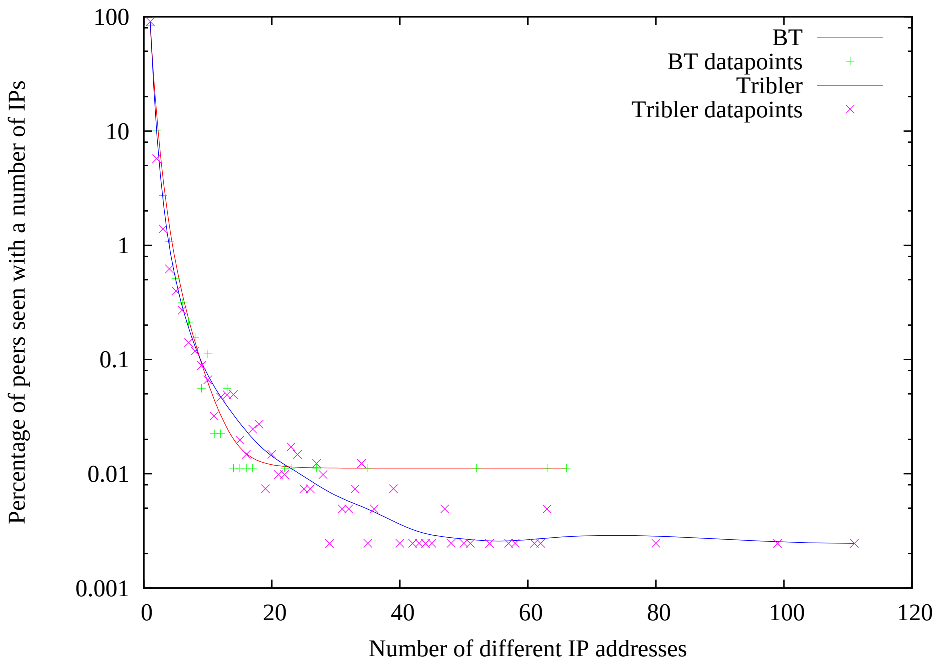


Figure 5.4: Number of Peers (in percentage) with number of IP addresses used on a BitTorrent Community Network and the Tribler network. The lines are Bezier approximations of the data points.

	Tribler	BitTorrent Community
Single IP address	36830(90.7%)	7545(84.4%)
Two IP addresses	2327(5.7%)	909(10.2%)
3-4 IP addresses	818(2.0%)	339(3.8%)
5-10 IP addresses	440(1.1%)	122(1.4%)
11+ IP addresses	191(0,5%)	20(0.2%)

Table 5.3: Number op IP addresses used by peers in Tribler and BitTorrent community.

5.3.1 IP Changes

The goal of the research done for this chapter has been to determine the IP address dynamics of peers in a P2P network. From the results can be seen that IP addresses of peers are not very dynamic. In both Tribler and the BitTorrent community most people in the network use only one IP address. Figure 5.3 shows the percentage of users that use a certain number of IP addresses in the Tribler network and the BitTorrent community. In the Tribler network 91% of the users only have one IP address during their time using the client. The vertical axis is on a logaritmic scale because the number of user using more IP address drops very quickly. The percentages drops to 6% of the users having two IP address and less than 2% of the users use more than five addresses. The Figure also shows the same for the BitTorrent measurement, in which 84 % of the peers only uses one IP address. In the BitTorrent community also only 2% of the peers uses more than five IP addresses. The percentages are shown in Table 5.3.
Figure 5.4 shows both datasets in one figure, which shows remarkable similarity between the two sets on the number of IP numbers per users. Both lines follow roughly the same trend. The resemblance of the two independent datasets confirms the validity of the analysis. Only in the end the lines get a bit different, but this is due to the fact that the Tribler dataset has more users.
Thus only small percentage of the users could be a potential problem due to changing IP address. The number of users with many IP addresses is very low, although it must be noted that some peers come online with a different IP address almost each time and some peers use a lot of IP addresses.
Peers might not use their P2P client all the time, and therefore IP address (switch) data is only from observed sessions of the P2P clients. Respectively 9% and 16% of the users of Tribler and BitTorrent community are seen with multiple IP addresses. Please note that this does not say anything about the amount of observed switches. Figure 5.5 and Figure 5.6 show the amount of switches of the peers that use at least two IP addresses. On the horizontal axis are the number of IP addresses and on vertical axis the number op changes that peers have. Almost ev-

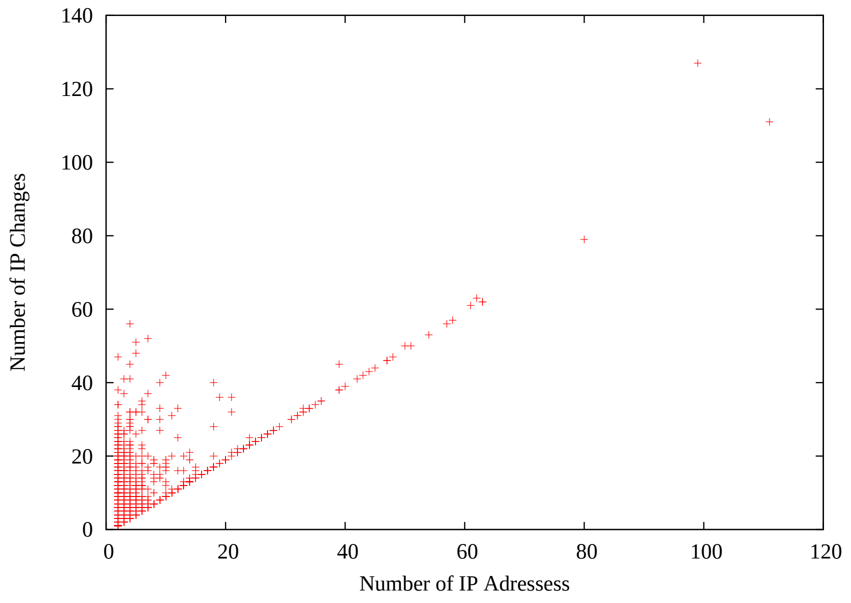


Figure 5.5: Number of IP address changes versus the number of IP addresses in the Tribler Network.

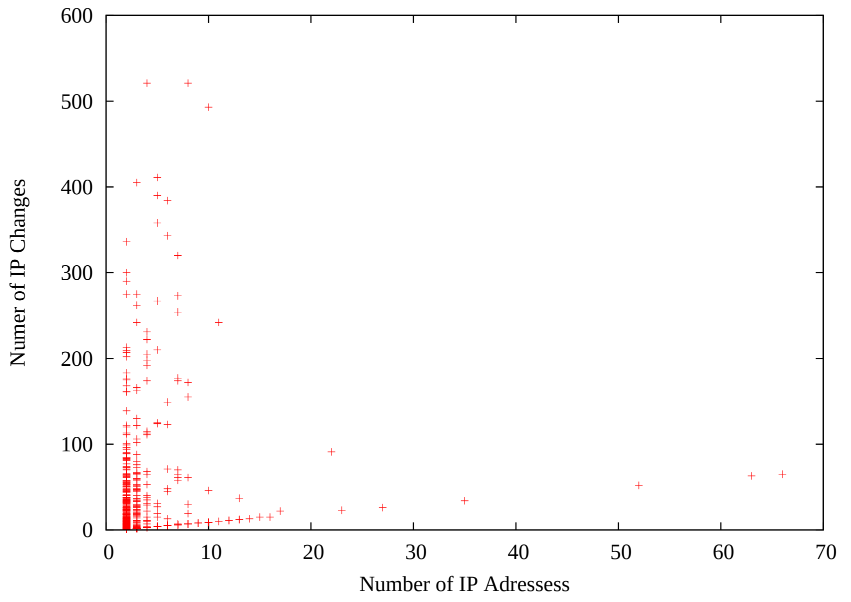


Figure 5.6: Number of IP address changes versus the number of IP addresses in the BitTorrent Community.

ery peer which uses 25 IP addresses or more in both datasets use an equal number of changes of IP address. Thus, peers that have a lot of IP addresses tend to acquire a new IP address every switch. On the other hand peers in the lower region have a lot more changes than the number of IP addresses and thus they change between IP address from a small pool of addresses. A strange difference between both datasets is the amount of changes in the BitTorrent dataset. The peers in the BitTorrent community change back and forth a lot more often than peers in Tribler. This could point out that some of the matching on the BitTorrent community contains a small amount of pollution.

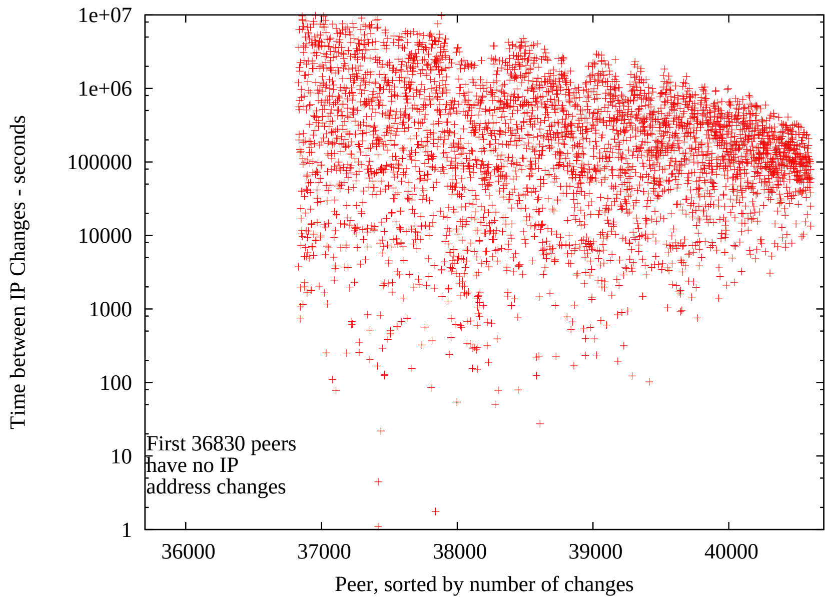


Figure 5.7: Average time between IP address changes of peers in the Tribler Network. The first 36830 peers have no change in IP address and are not shown.

The average amount of time that the peers are observed with one IP address before changing to an other IP address are shown in Figure 5.7 for Tribler and in Figure 5.8 for the BitTorrent community. It shows the average time between the moment a peers is observed with an IP address for the first time until it is observed with an other IP address. All the peers with IP address changes are shown and are sorted on the horizontal axis by the number of IP changes. The average time between switches in the Tribler graph is 288556 seconds, around 80 hours. In the BitTorrent network it is on average 10 hours. Thus even peers that do change, on average do not change very fast, which makes it easier for a social P2P network to find a changing IP address. However some peers switch between IP address after a short amount of time, as is also confirmed by Figure 5.9, which shows the minimal change times of peers in both datasets. This figure shows that some of

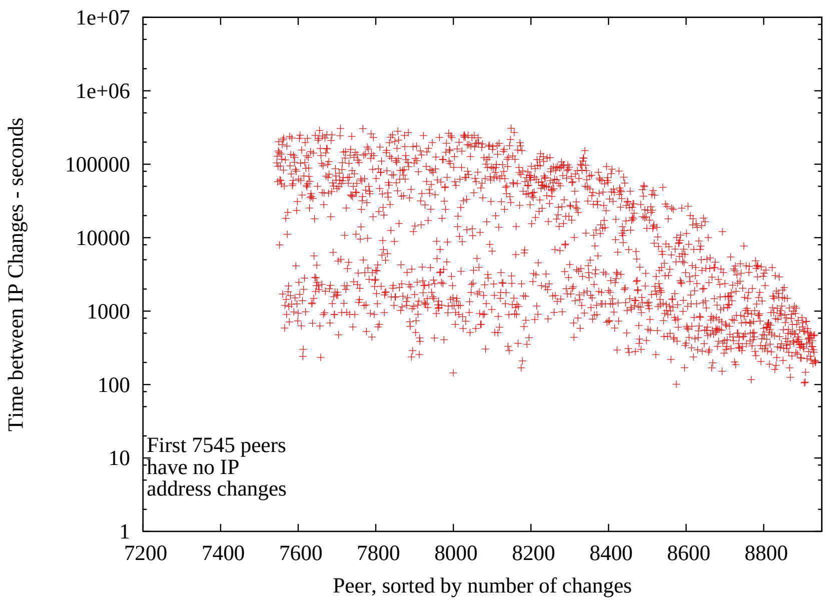


Figure 5.8: Average time between IP address changes of peers in the BitTorrent community network. The first 7545 peers have no change in IP address and are not shown.

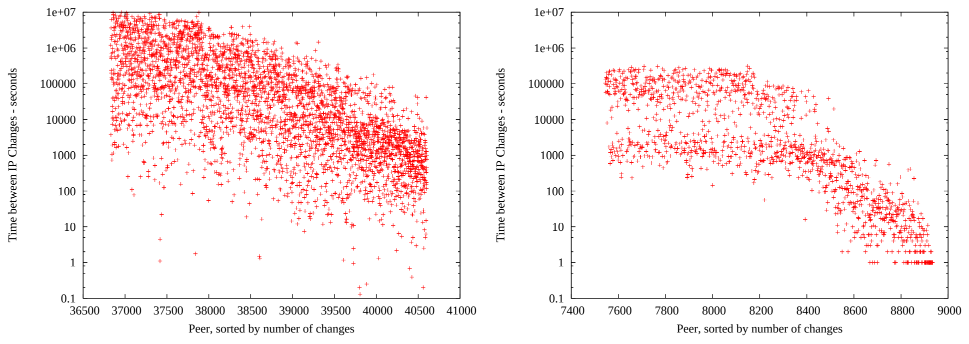


Figure 5.9: Minimal time between IP address changes per peer. The left figure is in the Tribler Network, the right figure in the BitTorrent community. The peers with no change in IP address are not shown.

the often switching peers could potentially change their IP address after a couple of minutes or even quicker.
For the social look up of changing IP addresses it would be interesting to see what the chance is that a peer is connectable or unconnectable at a previous IP address. Such numbers could be used to predict the chance that a connect to a peer will fail or succeed due to an IP address change. In a social P2P network the same users are connected over and over again. Therefore it is especially interesting to able to have knowledge about the probability distribution of an IP address change of a peer that has been seen with the same IP address multiple times. The graph in Figure 5.10 gives for both Tribler and the BitTorrent community the percentage of peers that had a different IP address at time t_{n+1}, given that they have had the same IP address n times. The chance that a peer will have changed its IP address are quite small. For example, if a peer has only been seen once, the chance it has a different IP address the next time is 9.5%. However, if the peer has been seen at the same IP twice this percentage already drops to a mere 0.80%. The graphs do not exactly follow the same trend, as with the number of IP addresses, but in both graphs a drop is fast. Also in both datasets, after the inital drop, the line starts to climb a bit again. A possible explanation is that some peers may keep their IP address for a short time, but in the long run a change is likelier to occur. For example a DHCP[2] lease may expire. Figure 5.11 show the individual graphs of the two data sets. It is quite striking that both datasets have a similar climb. Also in both datasets the minimim of the datapoints is almost the same; in both BitTorrent and Tribler it is around 21. To calculate the figure we used an interval of an hour, which could point to a change after around one day, thus be caused by peers returning a next day. We studied the effect of changes to this interval, which is not shown here. But, changing the interval does not change the form of the figure much and only moves the minimum point of the graphs to lower number a bit.

5.3.2 Availability and Connectability

The previous section shows that on average the IP address of peers are mostly static. Although already a lot of research has been done regarding the online time of peers in P2P networks [21, 39] the BitTorrent community measurement contains data on availability. Results from that data, presented here, can verify the claim that peers are highly unavailable. We extract from the log files for each peer its status, being either online or offline, during the measurement. The log files have an interval of five minutes and thus the status can be determined with a threshold of five minutes. Figure 5.12 shows the percentage of the total time that

[2]Dynamic Host Configuration Protocol

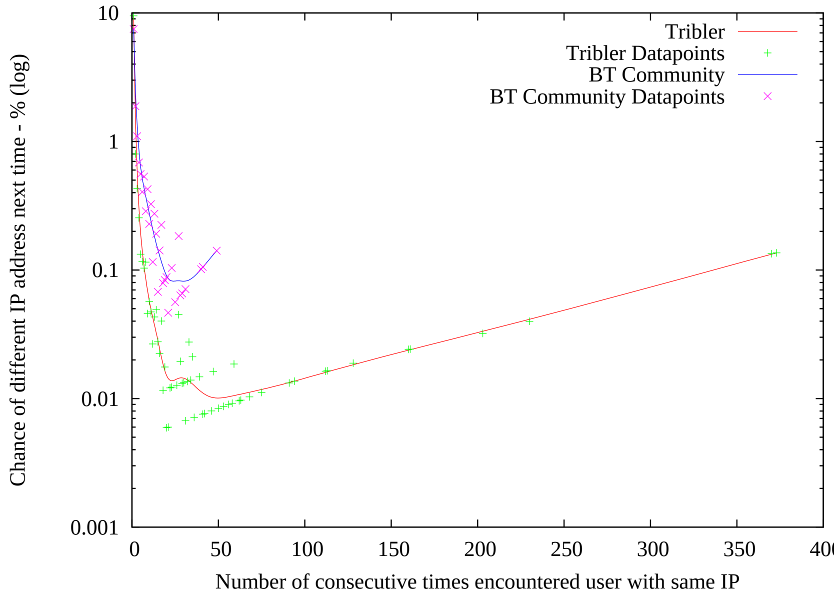


Figure 5.10: Chance of a peer returning with a different IP address if encountered for a number of consecutive times for both the Tribler and the BitTorrent Community network. (Only counts > 3600 minutes. The lines are Bezier approximations of the data points.)

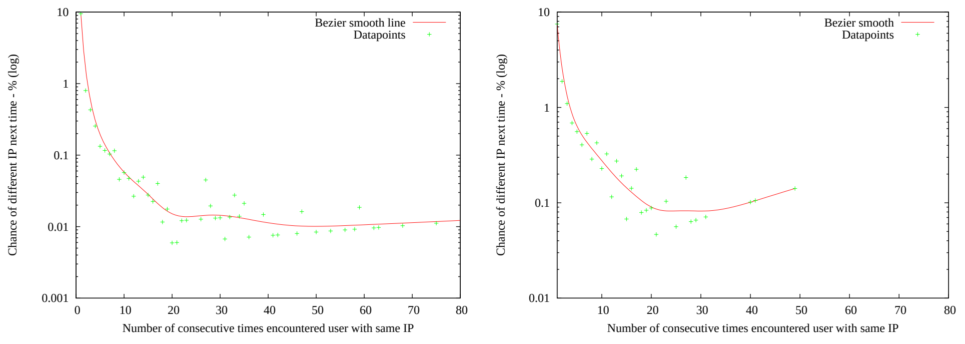


Figure 5.11: Chance of a peer returning with a different IP address if encountered for number of consecutive times. Tribler network on the left, BitTorrent community on the right. (Only counts > 3600 minutes. The lines are Bezier approximations of the data points.)

each peers was online. The figure gives the distribution of online time.
The data shows confirms earlier findings that most peers are not online all the time and in fact the majority of the peers are most of the time offline. In the figure the most reliable peers are on the left, which shows that only 1.3% of the peers is available more than 99% of the time and 10.1% of the peers is less than 1% of the time using the system. During the measurement a peer was on average 33.3% of the time connected to BitTorrent community.
Connectability is not researched here but it is also available in the statistics of the BitTorrent community. 61% of the peers were always connectable. So a 4 out of 10 peers were at leasts once not connectable. Such a low connectability could lead to a lot of problems in connecting to a peer.
Thus connecting to a random known peer has a very high chance that the peer is either unconnectable or unavailable.

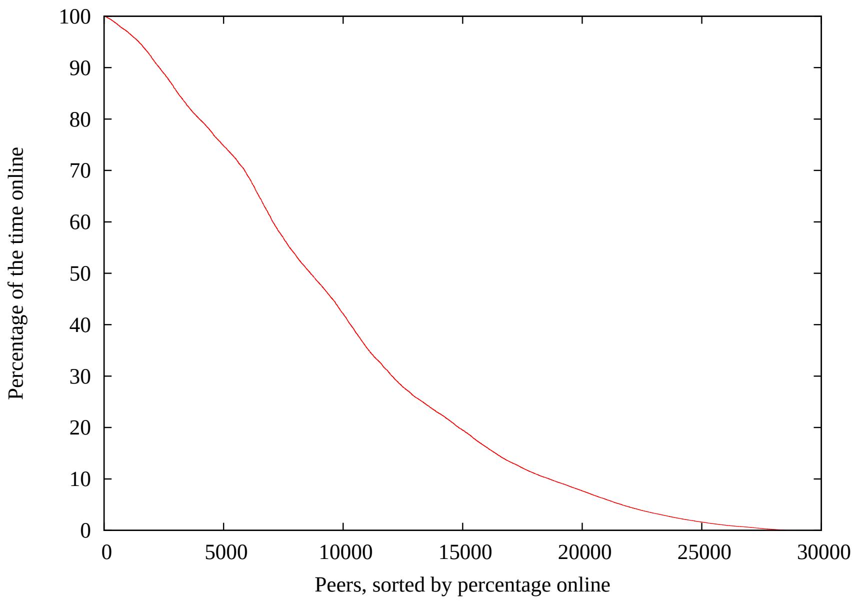


Figure 5.12: Percentage of the time users are online in the BitTorrent community.

5.4 Discussion

The Tribler analysis gives a good analysis of the dynamics of peers. Peers are identifiable by PermID. Peers can not forge their PermID and therefore no ambiguous data about the addresses can be in the dataset. However the measurement is on a fairly new network. Figure 5.2 shows that a lot of people only use the

client once or twice. This is probably because people only try the new client, especially after media attention. Also Tribler at this moment does not really exploit social control. The BitTorrent community does use social control and a lot of people are using the system continuously. It must be noted however that some users maybe running their client while using other trackers. In contrast to the Tribler network IP addresses are not directly linked to identifiable peers. In order to create these links the best possibilities were selected, as discussed in Section 5.1.3. Some faulty links may be presumed, and thus some of the final data used for the result might be ambiguous. Reviewing the data of the previous section shows that the independent datasets show much likeness, which is a confirmation that the BitTorrent measurement and matching process has been quite accurate. Although some extreme values are in Figure 5.6 on the whole the results are very similar and are likely to be correct. The combination of the two independent datasets and the similarity also strengthens the credibility of the conclusions.
The research in this chapter has been done as part of the IP discovery problem, as the third step required in creating a social P2P overlay network, as discussed in Section 3.2.3 and shown in Figure 3.2. A mechanism is required in a social P2P system to be able to find the current IP address of peers is required. Although the percentage of peers using multiple IP addresses is low some peers change address. It is important to notice that most peers actually only switch between a pool of IP address, therefore saving the last known couple of addresses of a peer can aid in rediscover the peer. Furthermore, in most cases peers do not change IP addresses in short time. Thus IP numbers are slow changing data and a good candidate of spreading the changes would be gossiping in the social network. Implementing a gossip mechanism which updates peers in the social network upon an IP address change is very likely to succeed in updating changes. Peers should announce their new address to all online peers in their social network as soon as a change occurs, and continue to spread IP addresses of itself and others around the social network. Up to a number of levels of FoFs, the complete social network of a peer can be kept up to date with latest addresses with only a small overhead in bandwidth since spreading the address can be done quite slowly. As discussed in Section 2.2.2 a network of thousands of friends and FoFs can be up to date in a couple of hours. Given that a social network feature is available in the system that provides exchanges of friends list, it would be easy to calculate common friends or common FoFs. Peers should always spread updates of their addresses. So in case a peer can not be connected at the latest known IP address, the friends and FoFs of the peer can be queried for updates. If any of these peers was online the last time the peer was online or it received a gossip, the new address can be acquired. For example in the relative small social network of Figure 3.1 consider a connect must be made to peer B by peer A, and B changed its address since the last connect. The latest address can be request from friends of peer B, the peers C, E, F, N and O. Each

peer could gossip address of friends and FoFs to all his friends and FoF, and thus spreading the information across four levels in the social network. If the update has been gossiped around, all the peers in this network could have the information, even if peer A did not forward anything. For example peer J and K could get the information from FoF I of B and peer D could get it from friend I or FoFs M, C and E. The chance that at least one of the peers is available is high. This way peer A is likely to acquire the new address of B.

Although peers may not change IP address very fast the previous section showed that the chance is high that a peer is unavailable or unconnectable. Results on availability on regular networks will probably be even lower because the BitTorrent community only gives access to peers that contribute. The community thus has a strong incentive for peers to stay online. The high chance of unavailability and unconnectability could mean that a peer sending an update or gossip of IP addresses has a high chance of failure. The unavailability and unconnectability are much more of a problem than IP address dynamics, and the effectiveness of the gossiping solution could decrease due to unavailability and unconnectability.

A lot of research is done to overcome unconnectability due to firewalls or Network Address Translation boxes, such as firewall puncturing [18] and UPnP[3]. More research should be done and a system should use methods to overcome unconnectability. Also instead of using a push gossip method, peers may also use a pull gossip method [34]. Unconnectable peers may be unable to receive, but can pull the information from connectable peers.

Unavailability can not be solved by technical mechanisms since it is the user who shuts down a computer or P2P client. However, incentives could increase the willingness of users to stay online. A social network could deliver good incentives, since social control is high and people tend to be altruistic towards friends.

[3]Universal Plug and Play: http://www.upnp.org

Chapter 6

Conclusions and Future Work

In this thesis we have presented a solution to the problem of creating an overlay network based on real social networks. Two main problems have been discussed. The first is the social network discovery problem, which deals with the creation of the social network and finding friends. The second part is the determination of the IP addresses of peers in the social network, which is required to create and maintain connections between peers in the social P2P network.

In order to solve the social network discovery problem, we have designed and implemented a module for Tribler which allows the users to search for people, known to them, using social identifiers such as email addresses or telephone numbers. The user can search for an identity mapping from a real-life identifier to a so-called PermID. In order to decrease the complexity for the user, input of this module can be from existing social networks. Two examples have been built for MSN Messenger and GMail contact lists. A small test has been carried out.

In order to find the IP addresses of people in the social network, we have proposed to use the social network. Using social connections will only succeed if a peer is able to connect to at least some peers in the network. Whether a peer is reachable is based on the availability, connectability, and IP address dynamics of peers. Connectability is outside the scope of this thesis. Availability has already been researched. In order to research the IP address dynamics of peers, a BitTorrent community and the Tribler network have been analyzed.

This chapter gives our conclusions with regard to these two main problems in Section 6.1. Some recommendations for future research and improvements are given in Section 6.2.

6.1 Conclusions

Based on the results of our research in social network discovery research, we can state the following conclusions:

1. Gossiping will spread the local knowledge of identities to all peers in the social network, which creates distributed knowledge of identity mappings. Because of the limited bandwidth usage per client and the relatively limited number of users in the social network of each peer, this method is very efficient and scalable.

2. Identity mappings are spread around the social network. If already connected to a social network, common friends are likely to have the identity information. This will aid the scalability of the system. Using superpeers as backups for information exchange makes the network operate even if no social network has been built or no common friends exist.

Based on the IP address dynamics analysis, we can state the following conclusions:

1. Most of the peers only have one IP address, and if a peer is seen with the same IP address a couple of times, the chance that a IP address change will occur is slim. This will make rediscovering peers easy using a social network. The chance that multiple peers change IP address is extremely low, making it possible to look up IP addresses through the social network as long as the social network is reasonably large.

2. The social network is known and every peer announces its IP address to friends and FoFs. Therefore a peer can calculate for each of its friends to which peers IP addresses have been announced. In case of an unconnectable peer a lookup can be very specific. This creates a small number of lookups and makes this method scalable and efficient.

3. The social network also creates a high level of trust so, even though a IP announce should be signed, a denial of service attack is unlikely since the announce is done in the social surroundings of the peer.

4. Most peers are unavailable most of the time, which proves to be much more of a challenge than the dynamics of IP addresses. However, unavailability does not impact the building of the social network connection.

6.2 Future Work

Finally, we present some possible future research for social P2P overlay networks and some technical improvements:

1. Although a user can search the network, and is sure that the values inserted are in fact inserted by the owner of the PermID, users can insert false data. A confirmation mechanism should be implemented before peers can fully acknowledge other peers as their friend. Further research into this subject is required to make the mechanism less vulnerable to attacks.

2. Gossiping is a good mechanism to spread slow changing information, especially combined with semantic routing or social networks. It would be useful to research the possibility of one gossip mechanism to support multiple types of information. This could lead to an integration with buddycast [40] and other (social) gossip methods in Tribler. Gossiping should be researched in more depth to create a more advanced gossip method.

3. Our current social network discovery mechanism has been tested on a very limited scale; it should be emulated or tested in a social P2P system.

4. The results from the IP dynamics analysis can be used to implement an address discovery mechanism. This has to be implemented and tested. The data acquired from both Tribler and the BitTorrent community can be used to run an emulation or simulation. This could prove in more detail that the social network can resolve the IP discovery problem.

5. The IP discovery is very likely to succeed in finding a peer that has changed IP address and is online. However, knowledge of unconnectable or unavailable peers should be considered as well to improve efficiency and performance. More specific research could combine data about connectability, availability and IP dynamics.

Bibliography

[1] K. Aberer, P. C. Mauroux, and M. Hauswirth. A framework for semantic gossiping. *SIGMOD Rec.*, 31(4):48–53, 2002.

[2] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases (extended abstract). In *PODS*, pages 161–172. ACM Press, 1997.

[3] S. A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internel telephony protocol, Dec 2004.

[4] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Understanding and deconstructing bittorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[5] K. P. Birman. The surprising power of epidemic communication. In A. Schiper, A. A. Shvartsman, and H. W.and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 97–102. Springer, 2003.

[6] BitTorrent. Draft bittorrent dht protocol http://www.bittorrent.org/draft_dht_protocol.html [online].

[7] N. T. Borch. Improving semantic routing efficiency. *Hot-P2P*, 00:80–86, 2005.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[9] H. Chen, M. Yang, J. Han, H. Deng, and X. Li. Maze: a social peer-to-peer network. In *In Proc. of CEC'04-East*, Sept. 2004.

[10] ABC [Yet Another Bittorrent Client. pingpong-abc.sourceforge.net[online].

[11] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[12] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003.

[13] K. J. Delaney. With nbc pact, youtube site tries to build a lasting business. Wall Street Journal, June 2006.

[14] DirectConnect++. dcplusplus.sourceforge.net [online].

[15] P. S. Dodds, R. Muhamad, and D. J. Watts. An experimental study of search in global social networks. *Science*, 301:827–829, Aug. 2003.

[16] J. R. Douceur. The Sybil attack. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *Peer-to-Peer systems, First International Workshop, IPTPS 2002*, volume 2429 of *LNCS*, pages 251–256. Springer, 2002.

[17] D. Eastlake and P. Jones. RFC3174 - US Secure Hash Algorithm - I (SHA1). Network Working Group, 2001.

[18] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. arXiv report cs.NI/0603074, arXiv, mar 2006.

[19] B. Furht, D. Kalra, F.L.Kitson, A.A.Rodriguez, and W.E. Wall. Design issues for interactive television systems. *IEEE Computer Magazine*, 28(5):25–38, May 1995.

[20] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[21] K. P. Gummadi, R. J. Dunn, S. S., S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329, New York, NY, USA, 2003. ACM Press.

[22] R. A. Hill and R. I. M. Dunbar. Social network size in humans. *Human Nature*, 14(1):53–72, 2003.

[23] I-Share. I-share: Sharing resources in virtual communities for storage, communications and processing of multimedia data. http://www.freeband.nl/project.cfm?language=en&id=520 [online].

[24] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, Tokyo, Japan, Mar. 2004. IEEE Computer Society.

[25] S.D. Koolen. A review of social p2p overlay networks. Literatur Research Assignment Computer Science, TU Delft, November 2005.

[26] Y. Kulbak and D. Bickson. The emule protocol specification. MSc thesis, The Hebrew University of Jerusalem, Jan 2005. Distributed Algorithms, Networking and Secure Systems Group.

[27] D. B. Lange. Present and future trends of mobile agent technology. In K. Rothermel and F. Hohl, editors, *Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, page 1. Springer, 1998.

[28] J. Liang, R. Kumar, Y. Xi, and K.W. Ross. Pollution in p2p file sharing systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1174–1185. Dept. of Comput. and Inf. Sci., Polytech. Univ., Brooklyn, NY, USA, March 2005.

[29] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *IPTPS '01: The First International Workshop on Peer-to-Peer Systems*, pages 94–103, London, UK, 2002. Springer-Verlag.

[30] S. Marti, P. Ganesan, and H. Garcia-Molina. Sprout: P2p routing with social networks. In W. Lindner, M. Mesiti, C. Türker, Y. Tzitzikas, and A. Vakali, editors,

EDBT Workshops, volume 3268 of *Lecture Notes in Computer Science*, pages 425–435. Springer, 2004.

[31] P. Maymounkovand and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric. In *International Peer-to-Peer Symposium (IPTPS02)*, 2002.

[32] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[33] A. Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P 2004)*, pages 202–209, Zurich, Switzerland, Aug. 2004. IEEE.

[34] K.D. Ryu M.S. Khambatti and P. Dasgupta. Push-pull gossiping for information sharing in peer-to-peer communities. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages pp. 1393–1399., Las Vegas, Nevada, June 2003.

[35] S. Nielson, S. Crosby, and D. Wallach. Kill the messenger: A taxonomy of rational attacks. In *In Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, feb. 2005.

[36] A. Parker. Peer-to-peer in 2005. Technical report, Cachelogic, 2005.

[37] Elizabeth Pennisi. How did cooperative behavior evolve? *Science*, 309(5731):93+, July 2005.

[38] E. C. Perkins. Ip mobility support for ipv4 - request for comments: 3344. Technical report, IETF Network Working Group, August 2002.

[39] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS'05),*, Feb 2005.

[40] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. van Steen, and H. J. Sips. Tribler: A social-based peer-to-peer system. Technical Report 2006-002 (presented at IPTPS 2006), Delft University of Technology, Feb. 2006. (presented at the 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)).

[41] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[42] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[43] J. Roozenburg. A literature survey on bloom filters. Research Assignment, November 2005.

[44] S. Saroiu, K. P. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Syst.*, 9(2):170–184, 2003.

[45] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Second Annual ACM Internet Measurement Workshop*, November 2002.

[46] I. Stoica, R. Morris, D. Karger, F.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, October 2001.

[47] Y. Upadrashta, J. Vassileva, and W. Grassmann. Social networks in peer-to-peer systems. *hicss*, 07:200c, 2005.

[48] Mahadevan Venkatraman, Bin Yu, and Munindar P. Singh. Trust and reputation management in a small-world network. *icmas*, 00:0449, 2000.

[49] R. Wakikawa, J. T. Malinen, C. E. Perkins, A. Nilsson, and A. J. Tuominen. Global connectivity for ipv6 mobile ad hoc networks. Internet-Draft, Nov. 2002. Work-in-progress.

[50] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–5, Washington, DC, USA, 2002. IEEE Computer Society.

Appendix A

Identity BootStrap Message Specification

The two new message are a request (GET) message and a store (SET) message. The SET message forwards information to other peers. It can consists of mappings for a number of peers. The GET message can request certain hashes of information and optionally also request all information of permanent identifiers. The `BOOTSTRAP_GET` message is a dictionary containing the following key and values

- 'permid' is the permanent identifier of the requester of the search
- 'searches' is an array of dictionaries containing:
  - 'service' the textual representation of the service. For example email or msn
  - 'hash' is the sha hash value of the concatenation of the service and the original value. For example hash(email:me@mail.com)
- 'permids' (optional). Array of permanent identifiers. Allows the requester to pull mapping values of the given PermIDs.

The `BOOTSTRAP_SET` message is an array containing dictionaries with the following key and values:

- Array:
  - 'permid' the PermID that is mapped.
  - 'mappings' is an array of mappings of the PermID:

- ∗ ’service’ the textual representation of the service. For example email or msn
- ∗ ’hash’ is the sha hash value of the concatenation of the service and the original value. For example hash(email:me@mail.com)
- ∗ ’insertion’ integer time of the moment the value was inserted by the original PermID in seconds after 1-1-1970 GMT
- ∗ ’signature’ the ECC signature of the hash of a concatenated string ’service + BASE64 of hash + time of insertion’. The signature consist of an tuple of the two signature values of an ECC signature
- ∗ ’mapping’ (optional) The non hashed value of the mapping. Can be omitted too ensure privacy
- ∗ ’ip’ a dictionary containing address information of the peer
    - · ’ipport’ tuple of the IP address and port number
    - · ’last_seen’ time the IP address was inserted
    - · ’signature’ the ECC signature of the hash of string representation of tuple(ip,port,last_seen)

Since it is an array it can hold mappings of more than one permid.