

Stimulating fairness in peer-to-peer networks

Boudewijn Schoon



Stimulating fairness in peer-to-peer networks

Research Assignment in Computer Science

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

Boudewijn Schoon
✉ peer-to-peer@frayja.com

January 15, 2007

Contents

| | | |
|----------|----------------------------------|-----------|
| 1 | Introduction | 2 |
| 2 | Theoretical background | 4 |
| 2.1 | Nash equilibrium | 4 |
| 2.2 | Prisoners dilemma | 5 |
| 2.3 | Service maturation | 6 |
| 2.4 | Tragedy of the commons | 7 |
| 3 | Social aspects | 8 |
| 3.1 | User identity | 8 |
| 3.2 | User behavior | 11 |
| 3.3 | Stranger policy | 13 |
| 3.4 | User incentives | 15 |
| 4 | Weaknesses and solutions | 16 |
| 4.1 | local knowledge | 17 |
| 4.2 | global knowledge | 20 |
| 4.3 | Protocol misuse | 25 |
| 5 | Case studies | 28 |
| 5.1 | BitTorrent | 28 |
| 5.2 | Maze | 30 |
| 5.3 | KaZaA | 32 |
| 5.4 | MojoNation | 34 |
| 6 | Conclusion | 36 |

Chapter 1

Introduction

The internet has been and still is growing and is made up of a large amount of people. To provide service to these people a large amount of servers exists. But the demands of people are changing. They are no longer satisfied with a text based web page or even with high resolution images. As our available bandwidth is growing, we are slowly demanding more and larger files. It is impossible for central servers to provide the bandwidth to fulfill all these demands, which has opened the way for peer-to-peer computing. In peer-to-peer computing not only the bandwidth of the central servers is used, but the combined bandwidth of all the users in the peer-to-peer community. This community idea is not new. The first large scale application that used this technique was Napster, which started in June 1999. Nowadays there are many different peer-to-peer applications and they all have different strategies and goals.

In traditional peer-to-peer systems a peer is assumed to be obedient - she should adhere to the specified protocol without consideration of her own wants and needs. While in most cases this assumption of obedience is correct, it has been shown that some people will spend time and effort to misuse a peer-to-peer network to their own advantage. Such a user is called a *free-rider*.

In 2000 a measurement study [1] was done on the Gnutella file-sharing network. They established that almost 70% of Gnutella users do not share any files, and nearly 50% of all responses are returned by the top 1% of sharing hosts. In 2005 another measurement study [7] was done, which showed that the amount of free-riders increased to 85% of all Gnutella users. It is not only Gnutella that faces the problem of free-riders. Many, if not all, peer-to-peer networks have at least some users that do not give enough, if any, service back to the community.

The characteristics of peer-to-peer systems present interesting challenges and opportunities for the design of incentive-compatible systems. Some of these characteristics include: lack of central authority, highly dynamic memberships, availability of cheap identities, hidden or untraceable actions, and collusive behavior. After reading this report the reader will have a clear idea of these threats

and how to limit the damage that they can do.

Throughout this report several examples will be given. In order to improve the readability of these examples the following naming convention will be used: Alice, Bob and Dave will represent users that are correctly following a given algorithm, Carol and Charlie are colluders and Marvin and Mallory play the role of malicious users.

First of all chapter 2 will provide a theoretical background on peer-to-peer networks. Chapter 3, social aspects, centers around the human side and the affect that it has on a peer-to-peer system. Chapter 4 contains a taxonomies of weaknesses and possible solutions. Chapter 5 follows with a description of two systems that are currently in use and finally we present our conclusion in chapter 6.

Chapter 2

Theoretical background

Peer-to-peer networks have been around since before the year 2000 and much research and development has been put into the subject. There are also several theories from before the era of computers that can also be applied to the peer-to-peer networks that are in use today.

2.1 Nash equilibrium

The fairness of peer-to-peer systems can be analyzed by applying concepts from the branch of mathematics known as game theory. One of these concepts is the *Nash equilibrium*¹. In a game with two or more players, each player can choose her own strategy and depending on this strategy she can either win or lose the game. The game is said to be stable when no player can benefit from changing her strategy. This stable state is called a Nash equilibrium.

It is important for a peer-to-peer network to achieve a Nash equilibrium because this would imply that there would be no gain for peers by deviating from the strategy that was devised for the peer-to-peer network. And when everyone in the network is following the same strategy, it follows that all individuals have the same opportunities and this is fair to everyone that is participating. Unfortunately achieving such an equilibrium is very hard, if not impossible, because of some inherent restrictions of the internet. These restrictions and possible solutions will be discussed in chapter 4.

¹http://en.wikipedia.org/wiki/Nash_equilibrium

2.2 Prisoners dilemma

The *prisoners dilemma* is a well known problem where a Nash equilibrium exists. The prisoners dilemma is as follows: There are two players and a banker. Each player holds a set of two cards: one printed with the word ‘cooperate’, the other printed with ‘defect’. each player puts one card face down in front of the banker. by laying them face down, the possibility

| | | Alice | |
|-----|-----------|-----------|--------|
| | | Cooperate | Defect |
| Bob | Cooperate | 3, 3 | 0, 5 |
| | Defect | 5, 0 | 1, 1 |

Figure 2.1: Prisoner dilemma payoff matrix

of a player knowing what the other player selected in advance is eliminated. At the end of the turn, the banker turns over both cards and gives out the payments accordingly. If Alice defects and Bob cooperates, Alice will get the ‘temptation to defect’ (T) payoff of five points while Bob receives the ‘sucker’ (S) payoff of zero points. If both cooperate they get the ‘reward for mutual cooperation’ (R) payoff of three points each, while if they both defect they get the ‘punishment for mutual defection’ (P) payoff of one point. To constitute a prisoners dilemma the following equations must hold: $T > R > P > S$ and $2R > T + S$. The Prisoner Dilemma has a single Nash Equilibrium: both players choosing to betray each other. What makes this interesting is the fact that this equilibrium (both betraying) has less payoff than when both players would cooperate. But this optimal strategy is unstable, because an individual player will have an even better payoff when she defects, and therefore it is not an equilibrium.

But unlike with the prisoners dilemma, a peer-to-peer network is not a one-turn-game. A peer in a peer-to-peer network has far more turns, or opportunities to either cooperate or defect. Thus the Nash equilibrium of the prisoners dilemma does not apply in a peer-to-peer network. When a choice is presented to the player again and again, this is called the *iterated prisoners dilemma*[2] which was first discussed by Robert Axelrod in 1984. Axelrod discusses that altruistic strategies tend to do better than selfish strategies when the players are given choices repeatedly over a long period of time. The best strategy was found to be tit-for-tat (which is used by the BitTorrent protocol, which will be discussed in section 5.1). This very simple application has only four properties:

- It is *nice* because it always cooperates in the first round (an unknown peer is usually given some data for free.)
- It is *retaliating* because it will defect when the other player has defected (it will stop uploading to a peer that is not providing any download in return.)
- It is *forgiving* because it will start cooperating when the other player, who defected in a previous round, cooperates (it will start uploading again once it

received download from another player again.)

- And it is *non-envious* which means that it will not attempt to do better than others (the peers follow the defined protocol.)

All these properties give the iterated prisoners dilemma two Nash equilibriums: (1) when both players choose to betray each other and (2) when both players choose to cooperate.

A reason that it is hard or even impossible for a peer-to-peer network to achieve a Nash equilibrium is due to the fact that a peer-to-peer network is a practical application of the iterated prisoners dilemma. The iterated prisoners dilemma is a hypothetical game with no end, and a peer-to-peer network does have an end to the ‘game’.

2.3 Service maturation

In [15] the term *service maturation* is introduced. Service maturation reflects the time that is required before a user has received all the benefits that she wants. Once all benefits have been received, the service is called mature. Once a service is mature, a user no longer has any benefit from staying in the network. In terms of peer-to-peer networks this means that a peer can gain more benefit by betraying other peers once she has received all or most of the service that she wants. Figure 2.2 presents the four different service maturations and their relations toward each other.

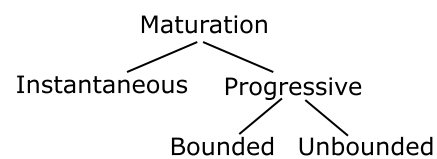


Figure 2.2: *Service maturation overview*

- A system that gives out all possible benefits immediately has *instantaneous maturation*. For example, this occurs when sharing a file in BitTorrent that is small enough to have only one part. This kind of maturation is difficult to secure because there is no reason, other than altruism, for a user to remain in the network after this one part has been received.
- We speak of *progressive maturation* when a system gives out the benefits over a period of time. Progressive maturation can be either *bounded* or *unbounded* based on whether the service has a known or fixed termination of pay-out. When the progressive maturation is unbounded a peer will continue to have some benefits for as long as she stays in the network. Because the benefit remains for as long as she stays in the network, this provides her with an incentive to do so. With bounded progressive maturation the benefits will, at one time, end. And with it the reason to remain in the network. We

see this occurring when a user wants to download a file and has completely downloaded all the data.

2.4 Tragedy of the commons

Another theory that can be applied to peer-to-peer networks is the *tragedy of the commons*. This theory states that when a group of people can freely use a common resource, this resource will likely be overexploited. The general idea of this theory is that a resource is maintained by a group of individuals. While the cost of maintaining the resource is shared equally between the individuals, the benefits of the resource can be shared unequally.

The following example demonstrates this idea: the use by individuals of communally owned land for the grazing of animals owned privately by those individuals. The utility to each individual of adding a single animal to her own herd is, more or less, the value of that animal; the cost to the individual is the consumption of the resources of that animal divided by the number of communal owners of the common. That is, the benefit to an individual of ‘hogging’ a resource inevitably outweighs the cost where communal resources are concerned. All economically rational herdsman in the community will add as many animals as they can to their own herds and as quickly as they can (before other herdsman do), meaning that the finite resources of the communal land will quickly become exhausted.²

The tragedy of the commons can also be applied to peer-to-peer networks. Looking at a file transfer system the following tragedy could occur: the combined upload bandwidth of all peers in a swarm can be seen as the common. This is provided, more or less, by all the peers. The gain for an individual peer is what she can download from the common. She is mostly interested in increasing her download. The upload that she is providing is, relatively speaking, only a small part of the common. So from her viewpoint it would change only little if she were to decrease her upload bandwidth and reduce her personal bandwidth costs.

²http://en.wikipedia.org/wiki/Tragedy_of_the_commons

Chapter 3

Social aspects

The user, or peer, is an important part of any peer-to-peer community. All websites and peer-to-peer communities exist to provide users with services. But with peer-to-peer networks the user is more than just a consumer; she is also a provider. Using the available resources of all users in a community naturally makes the community more efficient and allows for more or larger services to be provided. This is one of the most important reasons for the success of peer-to-peer networks.

Having a way to identify different users would be a great benefit when a decision must be made on providing some service to another peer. The identity of users is discussed in section 3.1. How users behave in a community and towards new users is discussed in sections 3.2 and 3.3. Finally section 3.4 gives ideas on how to get a user to cooperate without forcing her to do so. Figure 3.1 provides an overview of these topics.

3.1 User identity

A user identity is necessary when interaction between peers is required or encouraged. For instance: when treatment of other users will change according to the received treatment from them, it is required to distinguish other users from

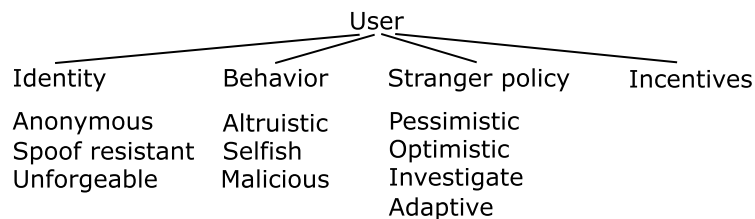


Figure 3.1: *Overview of social aspects*

each other. This is the basis of any strategy where behavior changes according to the behavior of others.

3.1.1 Anonymous users

Sometimes users may only be willing to participate in a network if there is a certain amount of *anonymity* [13] available. There are several levels of anonymity that can be recognized: no anonymity, pseudo anonymity (where the real-world identity is hidden), and complete anonymity (where all actions are completely disconnected from the peers real-world identity and his other actions).

- *No anonymity*: most peer-to-peer networks allow peers to use some pseudonym and their IP-address to identify themselves (i.e. BitTorrent, KaZaa, etc.) Although this is enough to provide each other with services, this does compromise their real-world identity because it can be traced back using the IP-address.
- *Pseudo anonymity*: to hide an IP-address a redirection scheme can be used with the disadvantage that the peers are no longer directly connected with each other. In a simple redirection scheme Alice might announce that she has a certain resource available while in fact she only knows that Bob has it. When Dave asks Alice to provide this resource Alice will first get the resource from Bob and then provide it to Dave. The original provider, Bob, is now effectively anonymous. Unfortunately this method can be fooled. When enough peers are working together, the actual identity of the original provider could still be known to one of them.
- *Complete anonymity*: to further hide the real-world identity it is required to change the pseudo identity and the redirection scheme every now and then. This further improves the chances of hiding the identity and is deemed strong enough to be called complete anonymity. A problem with changing your pseudo identity, that is done with complete anonymity, is that it is impossible to build some form of reputation system. Because users do not have enough time to build a reputation before their identity changes and they have to start building a new reputation again.

When a user is only interested in anonymity from others users, but is willing to trust one central server, then it is possible to build an anonymous system and still have a reputation system. TrustMe [16] is one of the systems that uses this strategy. Most peer-to-peer systems have no anonymity, this is by far the easiest to implement and has no additional communication overhead.

3.1.2 Identity spoofing

Most modern peer-to-peer systems will provide better treatment to peers that have treated them well in the past. This allows peers to receive something back for the effort that they put into providing a service. This whole idea is rendered useless when it is possible to assume the identity of someone else and receive the service that she has earned. Assuming the identity of someone else is called *identity spoofing* [13]. The most common way to avoid identity spoofing is by using a public/private key pair. With this technique Alice gives her public key to Bob. Bob can now decode messages that Alice encoded with her private key. And Bob can encode messages with the public key of Alice that only Alice can decode using her private key. A disadvantage of this technique is that the initial transfer of the public key can be intercepted by the malicious user Marvin. When this occurs Marvin can act as if he is Alice. This is called a *man-in-the-middle* attack. By using a trusted central server to handle login information and give out the proper identities to other peers, this problem can be minimized. However, this can only be done at the cost of adding a bottleneck and central point of attack for malicious users. In general a public/private key is safe enough to be used in a peer-to-peer system. However if the benefits of identity spoofing become high enough, man-in-the-middle attacks will increase and make the public/private key method ineffective.

3.1.3 Unforgeable identities

An *unforgeable identity* [13] should not only be spoof resistant but should also protect against *whitewashers* and *Sybil attacks*.

- Whitewashers are users that use all the benefits that a user identity provides and then discard this identity. Because user identities are often free, this can be done many times in succession with each new identity providing the user with more benefits.
- Sybil attacks are done by creating a large amount of user identities. These identities are then used to gain a large influence on the decisions of others.

Both these attacks use more than one identity to get around the protection of the system. An unforgeable identity must therefore be harder or more expensive to obtain than a normal identity, which usually incurs no cost. This in turn requires a trusted central system to generate and provide these identities. By requiring payment or another form of restraint upon the creation of a new identity, a user is no longer able to create new identities at no cost. It depends on the cost of the identity and the benefit of more than one identity whether or not this will help against whitewashing and Sybil attacks. Most large scale peer-to-peer systems (such as eDonkey and BitTorrent) try to get as much users as possible. This is a valid strategy because more users means more available resources. However, to

make it easy for users to become part of the system, there is usually no cost or registration required to join. Therefore, unforgeable identities are usually out of the question in peer-to-peer systems.

3.2 User behavior

There are many ways for a person to express herself; how a person expresses herself can be seen as her behavior. Since each peer in a network is usually represented by a person we should assume that a peer can also behave in more than one way. Most of the older sharing networks assume that all peers exhibit the same behavior. An example where this assumption proved to be incorrect is the Kazaa client. The original Kazaa client sends the nice rating for itself to a central server where this rating results in more or less download priority. Several months after the original release of Kazaa a new client became popular. This new client, Kazaa Lite, always sends the maximum rating of 1000 to the server, thereby receiving maximal download priority. This was a change in the behavior of the client, and posed large problems to the fairness in the Kazaa community. In this sense a single peer is like a person, it should not be expected that it will behave like every other peer in the network.

3.2.1 Altruistic

An *altruistic* [15, 17] user is ‘unselfishly concerned for, or devoted to the welfare of others.’ When a user provides service without directly or indirectly receiving service in return, she is behaving rationally. Fortunately there are altruistic peers in most peer-to-peer communities. Sometimes they exist because there are only altruistic clients available, and sometimes because the users simply decide to act on the behalf of the community. In the BitTorrent protocol a user can decide to be altruistic by staying online after finishing a download. There is no reason, other than altruistic reasons, for a user that has completed a download, to continue to upload.

3.2.2 Selfish

A *selfish* [13, 15, 17] user is ‘devoted to or caring only for oneself; concerned primarily with her own interests, benefits, welfare, etc., regardless of others.’ A selfish peer, who is otherwise known as a *rational peer*, will always use the knowledge and understanding that it has about the protocol to achieve maximum service from the network. However, it will not attempt to disrupt routing, censor data, or otherwise corrupt the system.¹ The following listed types of users are considered to exhibit selfish behavior:

¹This is a simplification of the term ‘rational’ as it is presented in [15].

- *Free-riders* are one of the best known types of selfish peers. A free-rider will ask for services from others but will refuse as many requests for service as it can. Example: A peer-to-peer client that allows the user to set her maximum upload speed, will become a free-rider if the upload speed is set to zero, or a speed that is lower than would be fair in respect to what she has downloaded from the community. This example contains the vague idea of ‘fairness’ which makes it hard to tell whether a peer is a free-rider or not. Some papers therefore describe a free-rider as a peer that does not give back any service to the community. But this is an unfair simplification of the definition of a free-rider.
- *Paranoid traders* are peers that do not trust other peers and do not want to provide more service than they receive. The only way a paranoid trader can achieve this, is to wait until a service is received and only then provide any service back. It stands to reason that if all peers in a peer-to-peer network have this behavior, that none will be the first to start giving service and all peers will wait indefinitely.
- *One-time risk-takers* are introduced to break the deadlock that exists with the paranoid trader behavior. A one-time risk-taker provides a free service to a peer the first time they meet. Further services will only be provided after a service is received. Unfortunately most peer-to-peer networks have free identities. This allows a peer to receive the services of another peer for free, get a new identity and then start the cycle again. A peer with this behavior is called a *whitewasher*. Simulations[17] were done with a peer-to-peer network only containing one-time risk-takers. This proved not to be a viable strategy. Sometimes this behavior will lead to situations where not all peers can complete the required services.
- *Periodic risk-takers* are introduced because even the one-time risk-taker can result in a deadlock. The behavior of a periodic risk-takers can solve this deadlock by giving away free service periodically. But of course there are no insurances that this will result in returned service. Furthermore, this behavior is ideal for free-riders who can now periodically gain free service. Although there are disadvantages to periodic risk-taking, it currently is the easiest way to avoid deadlocks. BitTorrent also uses this behavior with their optimistic unchoking. This concept is described in section 5.1.

Because a lot of peer-to-peer communities use open source software, changes to the behavior of clients is easier to make. When designing a protocol this should be taken into account. Nowadays it is usually assumed that the peers will behave selfishly and any feature that could be exploited by selfish peers are removed from the protocol, or modified to be more robust.

3.2.3 Malicious users

A *malicious* [13][15] user is ‘full of, characterized by, or showing malice; malevolent; spiteful: malicious gossip.’ The goal of malicious peers is to cause harm to either specific targets in the network or the network as a whole. To accomplish this, they are willing to spend time and resources on changing client software, hacking systems, distributing false information, etc. Some general ways to cause harm are described below:

- *Fooling the identity system*: if a peer-to-peer network uses some form of credits, then a peer can spend a certain amount of time gaining credits. After enough credits have been received, the peer can change her behavior and become a free-rider until she runs out of credit. A peer with this behavior is called a *traitor* (see section 4.3.1). In itself this technique is not unfair and should be seen as selfish rather than malicious behavior. However, when it is possible for a user to fake an identity or hack another peer in order to become someone else, this becomes malicious behavior. Another technique of fooling the identity system occurs when a peer-to-peer network has free identities and gives periodic or one-time service to new users for free. In that case it can become a target for whitewashers. When or before the other peers recognize the whitewasher for what she is, she leaves and rejoins the system with a new identity. She continues this until she has received all the service she wants. Preventing this kind of behavior can only be done to some extent. Ideas for this are presented in sections 3.1 and 4.1.2. However, securing the computer against hacking attempts to take over an identity (perhaps with viruses or worms) falls outside the scope of the peer-to-peer system.
- *Collusion*: collusion behavior suggests that peers work together to achieve an advantage over the other peers in the network. In itself collusion is not malicious behavior. Often it is encouraged for users to work together to achieve a goal. But with collusion the methods of gaining the advantage are malicious. In section 4.2.2 several collusion attacks are discussed.
- *Indirect disruptive behavior*: when the peer-to-peer network is not directly under attack but is changed through other means, this is called indirect disruptive behavior. A well known example of this is the Denial-Of-Service or DOS attack. This technique can be used to disrupt one of more parts of the network to remove a competitor or some central security peer

3.3 Stranger policy

The previous section described how a user can make her peer-to-peer application behave. This section will introduce another important type of behavior: how a peer

behaves towards a peer that she has not seen before. With the problem of free-riders like whitewashers it must be clear that the policy towards strangers is important. Too much free service, and the system will be plagued with whitewashers; not enough free service and the system will be prone to deadlock. An important issue is the definition of a stranger. It must be understood that a stranger is not always a peer that has just joined the network. It can also be a peer that has been in the network for some time, but whom you have not yet interacted with. Depending on whether your peer-to-peer network has global knowledge, you can either have only *local strangers* or both local strangers and *global strangers*. Also having global strangers has the advantage of knowing whether a peer only just joined, or has been interacting in the network for some time. However, global knowledge has the disadvantage of attracting an entire group of attacks that can be used to alter this knowledge. See section 4.1 and 4.2.

- Having a *pessimistic stranger policy* [13] means that a peer will never trust a stranger and will provide service to a peer only after she has received service from that peer. This stranger policy closely resembles the behavior of the paranoid trader that was described in section 3.2. It also has the same disadvantages: the network can deadlock because no one is willing to be the first to provide service.
- Having an *optimistic stranger policy* [13] means that a peer will always trust a stranger and provide free service in the belief that service will be returned some time in the future. This stranger policy resembles the behavior of a one-time or periodic risk taker that was described in section 3.2. It also has the same disadvantages: the peer can be cheated by free-riders like whitewashers.
- An *investigation policy* becomes an option because the pessimistic stranger policy has the possibility of deadlock. And the optimistic stranger policy has problems with free-riders. The investigation policy uses global knowledge to reduce the chance of providing service to a peer that has already received more service than what she has provided. But having global knowledge has the disadvantage of attracting global knowledge attacks, and it does not solve the problem with whitewashers.
- The *adaptive stranger policy* [13][4] uses the transaction history on services to strangers to estimate how likely it is that a new stranger will provide any service in return. This probabilistic policy only uses local knowledge and can adapt to the current rate of whitewashers in the network, at the disadvantage that you will also refuse to provide service to strangers that would have provided service in return.

3.4 User incentives

Some altruism is required to reduce the chance of deadlock in a peer-to-peer network. While chapter 4 will describe ideas on how the system can require some level of cooperation, this does not take into account the real-world user. However, the influence of the user should not be underestimated. When proper incentives are provided, the behavior of users can be changed for the good of the community. The new trend in peer-to-peer communities is a *social approach*. This approach can be seen as an elaborate *trust based system*, where the trust is based on amount of interaction, the age of interaction and how close interests of both parties match.

While this report is for the most part about ensuring that the user is not able to misuse the system, several ideas on how to influence the user that she will want to cooperate are also included. In essence these ideas are based on giving the user the sense that she belongs to a group. If she puts effort into proving herself useful to this group, then the group will respond positively towards her. The following ideas can be used to convince the user to cooperate:

- The client should allow the user to communicate with other users. A user is more likely to provide a service to someone that has asked for it personally than to someone who might not even be at her computer. This communication could be made more attractive by adding personal touches to your client. i.e. an avatar, font selection, user information, user blog, etc.
- The network should allow users to form subgroups with controlled membership. This will allow users to decide with whom they will interact on a regular basis. And users can set demands to be allowed in such a subgroup. A disadvantage to this idea is that such a group could take advantage of other users easily by demanding service and allowing the user only temporary access to the subgroup, or none at all. See the ‘picking on the newbie’ in section 4.3.1.
- The client should allow the user to decide where the bulk of her service is directed. This will allow a user to decide for herself instead of relying on the application to decide for her. It is also important that other users realize what other users are doing for them. All local knowledge should be made available to the user.

Chapter 4

Weaknesses and solutions

The core aspect of a peer-to-peer system is that it allows multiple users to interact with each other. However, because of the free and anonymous nature of the internet, it is hard to control this interaction. This lack of control is currently impossible to solve. However, it is possible to design a system that minimizes the damage that is done by malicious users. In order to design such a system it is necessary to have a clear idea of the problems, solutions, and problems that those solutions in their turn present.

There are only a few definite solutions to all peer-to-peer related problems. Two of these are *out-of-band trust*[15] and *trust software*[15]. With out-of-band trust, obedience is enforced external to the peer-to-peer system. This can occur when the system is used between a group of friends, or centrally administered machines within corporations, academic institutions, and government agencies. The idea behind trust software is that the user is prevented from modifying their software, and must therefore behave obediently to the protocol. We call these solutions *utopia*.

Unfortunately both out-of-band trust and trust software are usually not available. With out-of-band trust it will not be possible to harness the services of the vast amount of users on the internet and trust software is difficult or maybe even impossible to achieve with our current hardware. Therefore we will look at more realistic solutions in the following sections.

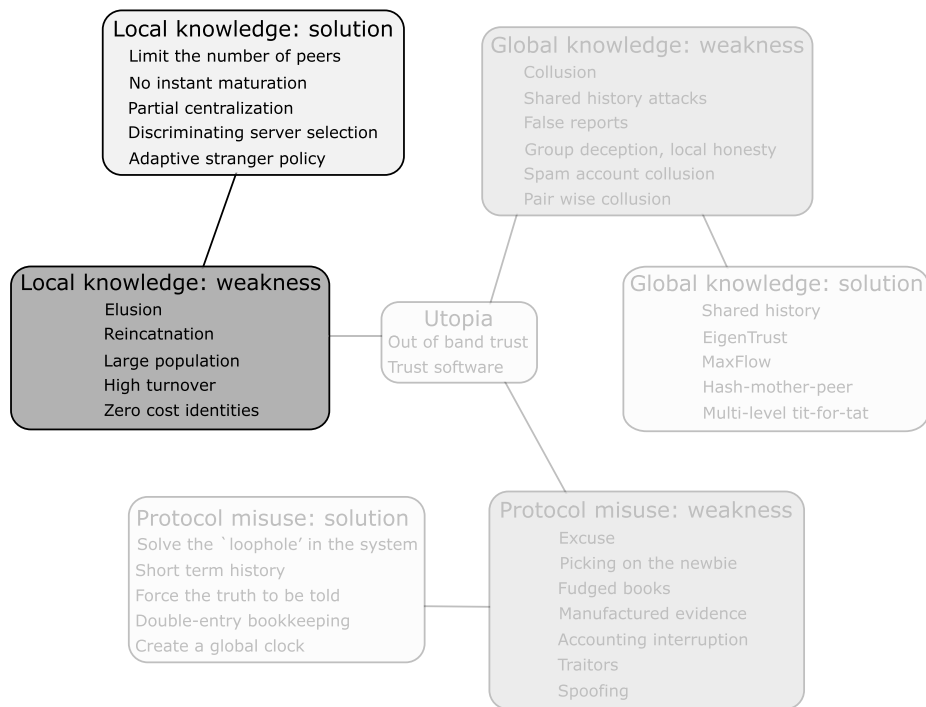


Figure 4.1: Overview of weaknesses and solutions of local knowledge

4.1 local knowledge

When designing a protocol for a peer-to-peer system, there are roughly two choices: use only local, or both local and global knowledge. With *local knowledge* a peer only knows what happened to her, she has no idea what is going on between other peers. This has the advantage that all her information is first hand and therefore correct. However, it has the disadvantage that her knowledge does not grow very fast, and is therefore not scalable as the number of peers increases. With *global knowledge*, which will be discussed in section 4.2, a peer also receives information about the interaction that other peers are having. This way all peers have a lot of information but can not be certain that this information is correct. The following sections will present the concept of service maturation followed by the weaknesses and available solutions in respect to peer-to-peer systems that only use local knowledge.

4.1.1 Whitewashing

Most peer-to-peer networks will provide a new peer with some free service, this is often necessary because a new peer would not be able to offer anything in return

for service received in the future. When a peer continually acquires a new identity and announces herself as a new peer, she is *whitewashing* her identity. All of the following weaknesses in peer-to-peer protocols allow some form of whitewashing or are a benefit for whitewashers in general. It must be clear that whitewashing is one of the bigger problems facing a peer-to-peer network. The following issues are relevant to whitewashing:

- Elusion and reincarnation are presented in [15]. *Elusion* involves a peer disconnecting from the network before it is required to provide some service in response to the service that she was given. *Reincarnation* is repeated elusion. Most peer-to-peer clients allow elusion, simply by allowing the user to select her own maximum upload speed. This gives her the opportunity to give less service than she receives, effectively eluding the system. BitTorrent is such a network that continues to provide service to peers that have not provided enough or any service in response. This is a consequence of the *optimistic unchoking* that is valued part of the BitTorrent protocol. See section 5.1 for more information on optimistic unchoking. It is not uncommon for a peer-to-peer network to allow a peer to remember those peers with which previous interaction was made. This can allow Alice to stop providing service to Marvin once she detects that Marvin is not giving any service back. This is when free-riders start using reincarnation, or whitewashing.
- Because most peer-to-peer networks have *zero cost identities*, it is no problem for Marvin (from the previous example) to reenter the network under the name of Mallory. He will now receive more service from Alice until Alice detects that Mallory is also not providing any service. This of course continues until Marvin a.k.a. Mallory has received all the service that he needs.
- When there are only a few peers in the network, then a user identity would become useless to a free-rider in a short time, forcing her to use techniques like whitewashing. When on the other hand the network consists of a large amount of peers, then a single user identity might suffice making it easier for a user to become a free-rider. As stated in [4] a *large population* and a *high turnover* increases the benefit to free-riders.

4.1.2 Solutions

There are several ideas on how to limit the damage that is done by the techniques described in the previous section. Unfortunately these techniques merely limit the damage.

- The system could be designed in such a way that there is *no instantaneous maturation* [15]. This will ensure that peers need to stay in the network for

a longer time. This increases the chance that peers will meet more often, allowing peers to gather historic information on other peers to use in their decision whether to interact with them or not. See section 2.3 on service maturation.

- A cost can also be associated to the creation of a new identity. This will *limit the number of peers* [15] in the network. When the costs of a new identity outweighs the benefit of reincarnation this will eliminate whitewashing. Unfortunately not all peer-to-peer communities are able to do this. When an identification cost exists, there will also be some connection between the real-world user and the peer. Some communities are not prepared to provide this. See section 3.1.
- *Partial centralization* [15] suggests that some aspect of centralization is introduced to make peers obedient. This could take the form of secure peer identification or a central rendezvous point as is used in BitTorrent. Unfortunately this will not have a positive effect if it does not increase the identification cost or allows detection of free-riders. When it is important to a peer-to-peer network that no central authority exists a hash-mother-peer approach can be used. See section 4.2.3.
- *Discriminating server selection* [4] is a method that can be used to increase the chances to encounter the same peers again. This helps against the large population problem discussed in the previous section. This method can, of course, only be used when a community contains several servers that group peers together into smaller sub communities.
- The *adaptive stranger policy* [13, 4] that has been discussed in section 3.3 will use the transaction history on services to strangers to estimate how likely it is that a new stranger will provide any service in return.

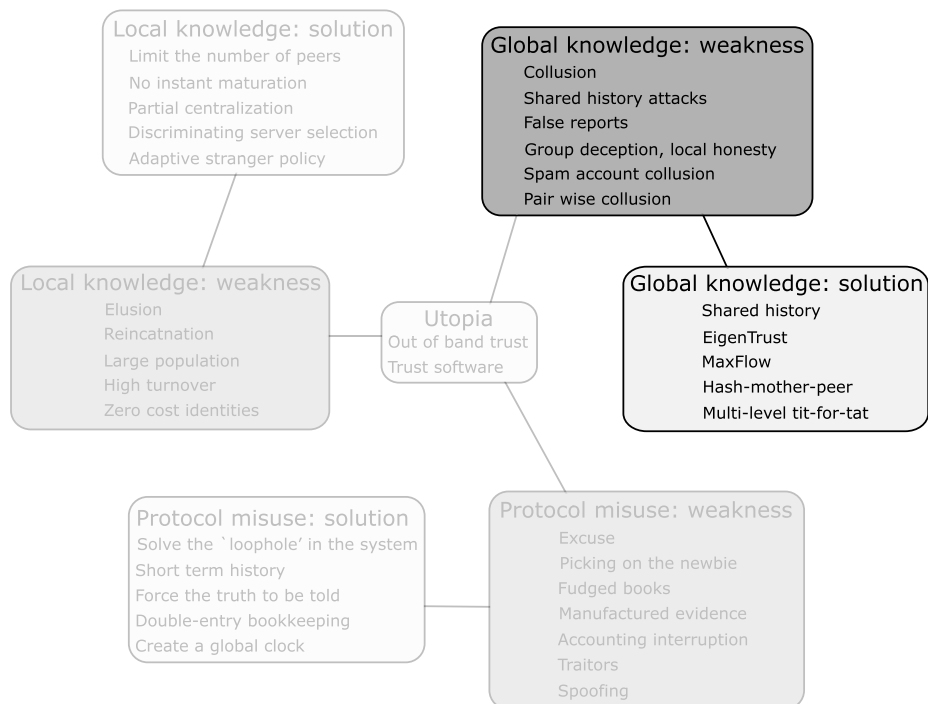


Figure 4.2: Overview of weaknesses and solutions of global knowledge

4.2 global knowledge

As stated in section 4.1: when designing a protocol for a peer-to-peer system, there are roughly two choices: use only local, or both local and global knowledge. With local knowledge, which has been discussed in section 4.1, a peer only knows what happened to her, she has no idea what is going on between other peers. This has the advantage that all her information is first hand and therefore correct. But it has the disadvantage that her knowledge does not grow very fast. With global knowledge a peer also receives information about the interaction that other peers are having. This way all peers have a lot of information, both local and global, but can not be certain that all this information is correct. The following sections will present the weaknesses and solutions in respect to peer-to-peer systems with global knowledge.

4.2.1 Whitewashing

Global knowledge is the logical next step after local knowledge, but what does it solve in respect to the free-riding problems that are known in a local knowledge system? If Marvin does not provide service to other peers, then the other peers can tell each other this through the global knowledge. This will cause other peers to stop

providing Marvin with services. This way Marvin can receive less service with a single user identity, forcing him to either start cooperating or get a new identity. So global knowledge encourages both cooperation and whitewashing. When the cost of the new identity outweighs the benefit of cooperation, more users will start behaving in behalf of the community.

4.2.2 Collusion

The biggest problem with global knowledge is that we cannot trust the information that we get from the other peers. It is obvious that Marvin can lie when Alice asks him how much he has contributed to the community. But Alice can ask other peers about the behavior of Marvin. But do we trust these other peers?

Let us look at the following situation: Carol asks Alice for service and Alice has to decide whether or not to give it. Alice starts asking other peers about their interaction with Carol. Charlie replies that Carol has provided him with an enormous amount of service. Alice is now convinced that Carol has done her part for the community and decides to provide the requested service to Carol. The problem here is that Carol and Charlie are working together to convince the other peers that one or both are doing a great job to the community. This concept is called colluding and is the largest problem for a peer-to-peer network that uses global knowledge.

- Shared history attack[4] and collusion are more or less the same thing, although collusion tends to be used as a term to group various collusion problems into one name. The term shared history attack focuses on attacks that target distributed hash tables. A distributed hash table can be used to synchronize global values between individual peers.
- *False reports*[4] is where a peer gives out false information about another peer to increase or decrease her reputation. The following false claims can be made:
 - *to receive service* is the collusion attack that is made by Carol and Charlie at the beginning of this section.
 - *not to have received service* allows the attacker to lower the reputation of another peer. Thereby raising the chance of receiving service herself.
 - *to have provided a service* allows the attacker to boost her own reputation.
 - *not to have provided service* provides no benefit to the attacker.
- ‘*Group deception, local honesty*’ [15] and *spam account collusion*[11, 10] are both the same thing. When your system has some form of credits, then colluder Carol can gain credits by providing service to Charlie. If Charlie only exists to increase the amount of credits of Carol, these peers

are colluding. This is usually done with more than one ‘Charlie’ and with all Charlie’s running on the same physical machine as Carol. Proof is presented in [11] that this behavior is very effective in the Maze network that has been used in China for some time now. The Maze network is discussed in section 5.2.

- *Pair wise collusion*[11, 10] is another way to exploit collusion. It involves two peers providing service to each other. This form of collusion works well in the Maze network because this networks uses a credit system where it costs less credits to receive a service that is pays to provide service. So two peers providing each other with service gain more credits by providing service than they lose by receiving service. This occurs often between peers, but when both peers are owned by the same user and are on the same physical machine.

4.2.3 Solutions

There are a lot of different techniques to face the problems of collusion. Most of them either require a trusted central authority or combine the opinions of all peers in the network and hope that this levels out any attempt at colluding. Although global knowledge can reduce free-riding, it is important to realize that all these solutions can be subjected to collusion themselves.

Shared history can be both a good and a bad thing. A peer-to-peer network that uses a shared history can decrease free-riders that would flourish in a local knowledge system. The downside of having a shared history is that it is susceptible to most collusion attacks that are described in the previous section. The solution for this is yet another layer on top of the shared history. In effect it all comes down to trust. The problem is that trust can not be measured by a peer-to-peer network. It will therefore most likely use either a value that is provided by the user, a value that is calculated amongst several peers or a value that is based on local knowledge. Figure 4.3 gives an overview of the information that a peer can work with and where this information is based on. The decision that the peer needs to make is based on one or a combination of local knowledge, global knowledge and

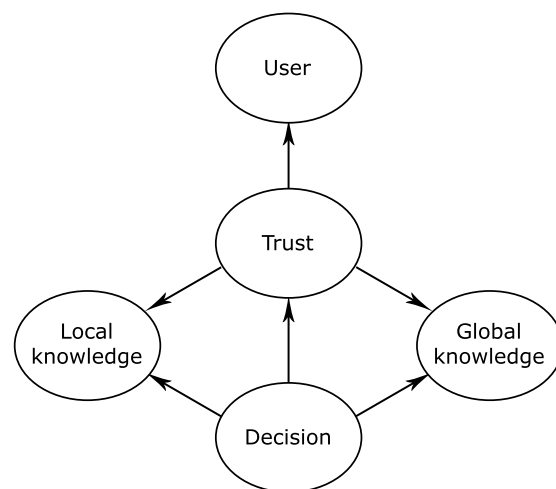


Figure 4.3: *Flow of trust*

trust. Trust is again based on one or a combination of local knowledge, global knowledge and a user defined value. Unfortunately this gives a lot of dependencies on local and global knowledge. While local knowledge is safe but slow to gather, global is unsafe and fast to gather. The only up side of using trust in the decision making is the user defined value. However, most of the time you will not have such a value because an application should be able to function without user input.

There are many different solutions to achieve a shared history, the following list presents some of these:

- EigenTrust[9] uses the combined opinions of all the peers in the network to compute a global trust value for each individual peer. MaxFlow[4] achieves the same result using graph techniques. Both techniques are not ideal and are subjective to collusion. For networks with a large amount of users this can work, because the large amount of opinions will help reduce the effect of colluding. But in [11] the data from the Maze system was diagnosed with both EigenTrust and other detection methods based on the large amount of available information, and this showed that EigenTrust did not always perform as expected. For example: some seeders were given a low trust value because they interacted with peers that themselves had a low trust value. Reputation systems are discussed in [15] and [4].
- The *Havelaar*[5] system is a strategy to reduce collusion in peer-to-peer networks. The problem of collusion occurs when a colluder Carol can create and control peers and use these peers to change reputation scores or other statistics that are used for fairness. Some of Carol's collusion attacks can be detected (several detection methods are presented in [11]) but this requires an overview of all the transactions that occurred with Carol. This is only feasible when a central peer is available and trusted to gather all this information.

In *Havelaar* no central peer is used. Instead the responsibility of gathering the transaction information of Alice is given to roughly seven other peers. In this case Alice is called the *successor-peer* and the seven other peers are called her *mother-peers*. In *Havelaar* each peer is a mother-peer and is responsible to audit her successor peers. All peers therefore have the responsibility to audit roughly seven successor-peers and are themselves audited by roughly seven mother-peers. The most important part of this strategy is the match between mother- and successor-peers. This match is achieved by using a mathematical function on the user-id. This function provides a list of mother-peer-id's that are in turn used. Because all peers know this mathematical function, all peers know who the mother-peers are of each peer. The colluder Carol now has to convince all her mother-peers of her good behavior. If the 'evidence' is overwhelmingly positive then the mother-peers can interpret this as collusion behavior and punish Carol by giving her a low score. When Alice wants to provide service to Carol, she first contacts Carol's mother-peers and requests

her score. Depending on the result, Alice will continue to interact with Carol or not.

There are problems with this idea though. Because most peer-to-peer networks allow a user to create her own identifier, it is possible to control who your mother-peers will be. If you control enough mother-peers, then you control your own reputation. There also remains the problem of the central nature of this strategy. The mother-peers provide a good target for malicious users who want to punish a mother-peer who is giving them a bad reputation.

- The *TrustMe* system (from [16]) also uses the idea of mother-peers, but has a different way to match a mother-peer with her successor-peers. When Alice connects to a TrustMe network, she first connects to a so called ‘bootstrap’ server. This bootstrap server tells Alice who her successor-peers are and notify her mother-peers that they have a new successor-peer. Another important part of TrustMe is that all messages are encrypted using public/private keys. Each peer has two of these pairs. One to communicate with all other peers and one to communicate with her mother-peers. These key pairs are provided by the bootstrap server when the peer connects. When Bob wants to contact the mother-peers of Alice (because he wants to provide a report on a transaction or because he wants to know the reputation score of Alice) he broadcasts messages to all other peers. Only the mother-peers of Alice can read these messages because only they have the appropriate private key. The reply of the mother-peers is also indirect to ensure that the identity of the mother-peers remains a secret.

The advantage of this strategy is anonymity. This ensures that mother-peers can not be subjected to DOS, or other directed, attacks. The disadvantages are the bootstrap servers that give out the public-private keys and match mother- and successor-peers. Finally, this strategy gives a larger overhead in communication because all messages between mother and successor-peers must be broadcasted, because their identity and location is unknown.

- In [10] both the local knowledge approach and the global knowledge approach is merged. They present *multi-level tit-for-tat*. Their findings show that having a trust system that is based on friends and friends-of-friends is better than either trusting only local knowledge or the global knowledge of EigenTrust. From [8]: The tit-for-tat protocol is a typical local knowledge system. It is *nice* in the sense that a peer cooperates as long as the other peer does. It is *retaliatory* which means that a peer stops cooperating when the other peer stops. It is *forgiving* because it will cooperate when the other peer starts cooperating again, and it has *clear behavior*. These properties have made the tit-for-tat protocol very popular in both game theories and application like BitTorrent.

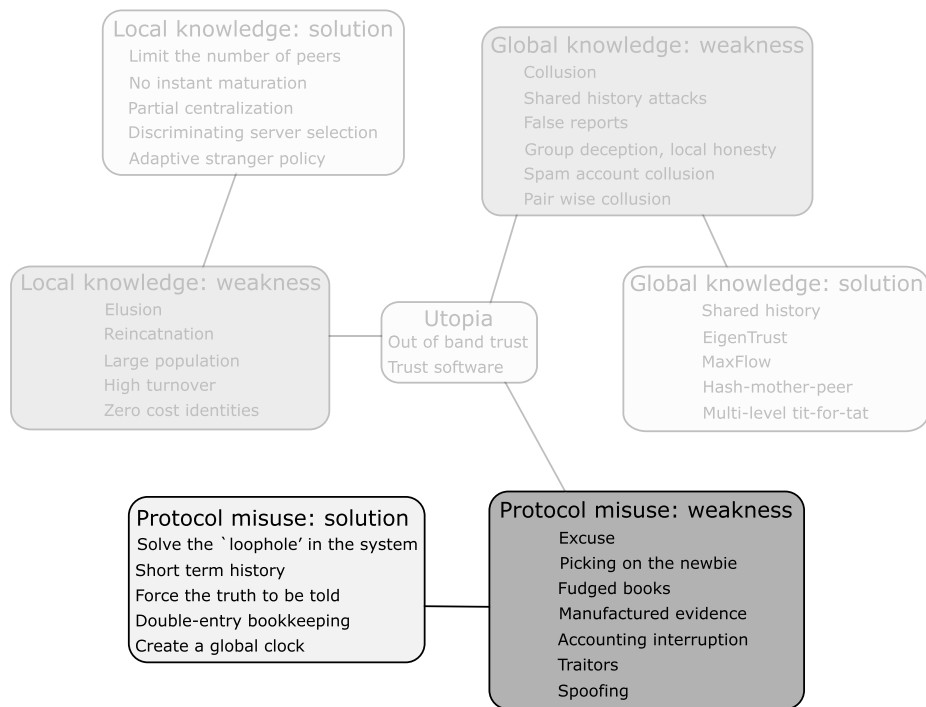


Figure 4.4: Overview of weaknesses and solutions of protocol misuse

4.3 Protocol misuse

When designing a peer-to-peer system careful consideration must be taken in designing the protocol. Some protocols are easier to mislead or take advantage of than others. In this section we will discuss several weaknesses and solutions that can apply to the protocol of a peer-to-peer system.

4.3.1 Weaknesses

- *Excuse*[15] is one of the best examples of a protocol weakness. Most peer-to-peer networks will have some form of excuses that allow a peer to tell others that it can no longer provide a service. An example could be a remote backup system like Samsara. This system requires every peer to contribute as much space as it consumes. If the system policy is overly generous to a recently crashed peer by not requiring them to prove they are maintaining their quota, a malicious peer may exploit this by repeatedly claiming to have crashed.
- *Picking on the newbie*[15] where strangers take advantage of other peers, is in contrast to the whitewashing problem. The problem of picking on the

newbie occurs when older peers take advantage of strangers or newbies and can occur when strangers are required to ‘pay their dues’ by requiring them to provide service to the community for some period of time before they can receive a service. If not carefully designed, an older peer that has already paid his due could continue to request service from new peers and not provide any service back to the system.

- *Fudged books*[15] is a problem for auditing systems. An auditing system relies on the accounting records being tamper-resistant and difficult to forge. A specific auditing attack is *manufactured evidence*. In this scenario, a user who is not doing his part for the community is able to provide ‘proof’ of cooperative behavior nonetheless. A particularly destructive auditing attack is *accounting interruption* where specific peers, who are responsible for the auditing, are targeted to disrupt their auditing activity. This might be accomplished by a DOS attack, worms or viruses, etc.
- *Traitors*[4], are peers who acquire a high reputation score by cooperating for some time. They then traitorously turn into free-riders before leaving the system. The following section will describe a way to reduce the effect of traitors on the network with a technique called ‘short term history’. The behavior of traitors has already been described in 3.2 under the name of *fooling the identity system*.
- *Spoofing* occurs when Marvin takes the identity of Alice to use the benefits that she has earned by providing services to the community. This problem is similar to the traitor, except that with spoofing it is another user who assumes the identity and changes the behavior of a honest user.

4.3.2 Solutions

- Out of all the weakness that are mentioned in the previous section, the excuse is the most easily solved. Because this problem is caused by faulty protocol design, it is logical that this problem is solved by removing the part of the protocol that allows the excuse. Usually the protocol needs to be expanded with options that can provide the same functionality without the risk of an excuse.
- A simple way to reduce the effect of traitors on the network is a technique called *short term history*. This technique is introduced in [4]. When the history is short, all peers will quickly forget the services that a traitor provided. This should ensure that the traitor is required to continue cooperation, and limits the advantage of spoofing an identity. The obvious downside to this is that services that are provided in the past are no guarantee to receive service in the future. Which is unfair and can be a reason for users to decrease the amount of service that they are willing to provide.

- With *force the truth to be told*[15] peers can usually only believe what they observe for themselves. When other peers provide information some form of secure history should be used to verify the truth. TrustMe from [16] does this to some extent.
- *Double-entry bookkeeping*[15] duplicates auditing information on several peers. This should reduce the problem of accounting interruption and reduce the chance of collusion between the auditing and audited peer.
- *Creating a global clock*[15] can give more information about historic transaction. This can slow down and reduce the benefit of some collusion attacks, and it can make it easier to recognize collusion attempts.

Chapter 5

Case studies

The previous sections presented an overview of ideas and techniques that can be used to stimulate fairness in peer-to-peer networks. However, they did not provide an in depth description on the workings of a peer-to-peer community. In the following sections we will present four such communities: BitTorrent, Maze, KaZaA and MojoNation.

5.1 BitTorrent

BitTorrent is a peer-to-peer file distribution tool that employs a tit-for-tat incentive mechanism to reduce free-riding and increase user cooperation. Since its release around 2003, it has proven to be very popular: CacheLogic estimates that BitTorrent generated about 30% of all US Internet traffic in June 2004.

In [3] Bran Cohen describes the architecture, or technical framework, of BitTorrent as follows: A BitTorrent *deployment* starts with a static file with the extension ‘.torrent’. This file describes the content, its length, name, hashing information, and an url to a tracker. A tracker is responsible for helping peers to find each other. While in most peer-to-peer systems, the system itself allows some form of searching for new content to download, the BitTorrent protocol does not. It relies on external means, such as a web site or email, to distribute the ‘.torrent’ files. Once a user has a ‘.torrent’ file for the content that she is interested in, she can connect to the associated tracker. With the help of this tracker a list of other peers is received. She can now start connecting to these other peers in order to participate in the distribution of the content. We say that she is now in *the swarm*.

While in a swarm a peer will try to download small *chunks* of the content. The content itself is divided into chunks typically a quarter megabyte large. The ‘.torrent’ file contains a SHA1 hash for each chunk to allow a peer to check the integrity of the downloaded data. To promote further parallel downloading these chunks are again separated into smaller chunks typically sixteen kilobytes large.

These smaller chunks are requested from other peers in the swarm. At least some, usually five, requests are outstanding to avoid delays between chunks. A delay could be very costly because of the nature of TCP. Another important aspect of chunks, is the decision on what chunk to request from another peer. BitTorrent has different strategies for the different phases that a download goes through: The first chunk will be chosen randomly. If a rare chunk would have been chosen it could be some time before this is received (because the demands on this chunk and the peers that have it are high.) Once a complete chunk has been received the peer can send this chunk to other peers which in turn will increase the chance of receiving chunks from them. Most of the time the peers will try to decide which chunks are rare and attempt to download these as soon as possible. Using this strategy the chances are minimized that, when the original seeder leaves, some chunks will no longer be available in the swarm. When almost all chunks have been received, the endgame mode starts. This strategy sends several requests for the same chunks. It will allow the fastest peer to complete the transfer and send cancels to the slower peers.

BitTorrent does not try to limit the amount of data that it uploads to other users. Instead it changes the speed at which it uploads. When an upload speed is lowered, then this is called *choking*. A BitTorrent client usually unchokes the upload to four other peers. This allows the congestion control that is build into TCP to reliably saturate the upload capacity. The client uses the current download-rate to decide which connections to choke or unchoke. To avoid situations in which resources are wasted by rapidly choking and unchoking, the peers decide whether or not they want to choke every ten seconds.

Another important part of the choking algorithm is the *optimistic unchoke*. This allows Alice to unchoke Bob for thirty seconds. If Bob responds by unchoking his connection to Alice, then Alice can decide whether Bob provides better speed than any of her currently unchoked peers. Thus this optimistic unchokeing allows a peer to discover better connections than she currently has.

The observation that BitTorrent experiences remarkable good performance is strange when viewed from the point of free riders. When Marvin decides to stop uploading to other peers, this is punished by the peers removing Marvin from their unchoked connections. But they will continue to try to optimistically unchoke their connection to Marvin. And each time that this occurs Marvin receives will a thirty second data burst. Even in a small swarm, this will eventually give Marvin enough data to complete the download. But if it is so easy to free-ride in BitTorrent, why does it work so well? [6] gives some ideas on this. They credit this to two the following factor: By having the '.torrent' files, and their distribution, external to BitTorrent, the swarms are isolated from each other. Because of this isolation, it becomes easier for swarms with high cooperation to flourish and swarms with low cooperation to die out.

5.2 Maze

Maze is a peer-to-peer file sharing application that is developed and deployed by an academic research team. Maze uses a centralized, cluster-based search engine, and is additionally outfitted with a social network of peers. The hybrid architecture of Maze offers exact keyword-based search, simple locality-based download optimizations, and also reduces dependency on the central cluster. Maze is deployed across a large number of hosts inside China's internal network. As of October 2004, Maze includes a user population of about 410K users and supports searches on more than 150 million files totaling over 200TB of data. At any given time, there are over 10K users online simultaneously, and over 200K transfers occurring per day.

In order to use Maze, a user must first register herself. This registration has no costs but the research team gets the means for authentication and statistics gathering. At startup a Maze peer first connects to a central server for authentication. Following this the peer uploads an index of the files that she has in her Maze directory. The peer also sends periodical heartbeats to signal that she is still online. Because the server has the file indexes of all peers, full-text queries can be conducted on the set of online peers. When a peer wants to download a file, she can make connections to multiple peers and download chunks in parallel. The server provides a list of peers that are in close proximity according to the amount of prefixes that are shared in the peers IP-address. Because Maze tries not to have the peers to depended on the central server, each peer also maintains three lists of her own:

- The *friend-list* is a list that is bootstrapped from the central server when the peer first registered. A peer is added to this list if it proves to be reliable and is removed when it is unreliable.
- The *neighborhood-list* is a list that contains online peers that share the same B-class IP-address.
- The *high-reputation-list* is a list of peers who currently have high reputation scores.

Both the friend-list and the neighborhood-list help against problems like large populations and high turnover which were discussed in section 4.1.1. The high-reputation-list gives an incentives to provide good and reliable service, because through this list they become known to a large audience.

In Maze an *incentive system* is used where points are rewarded for uploading, and points are deducted for successful downloads. When this incentive was being developed, the amount of points gained or lost was discussed on the Maze forum. The community decided to use the following rules:

1. New users are initialized with 4096 points.
2. Uploads: +1.5 points per MB uploaded.
3. Per file downloaded:
 - -1.0/MB downloaded within 100MB.
 - -0.7/MB per additional MB between 100MB and 400MB.
 - -0.4/MB between 400MB and 800MB.
 - -0.1/MB per additional MB over 800MB.
4. Download requests are ordered by $T = requestTime - 3logP$, where P is the total points that a user has.
5. Users with $P < 512$ have a download bandwidth of 200KB/s.

It is obvious that a peer will gain more points by uploading than she will lose by downloading the same amount of data. This means that, over time, the total amount of points in the system will grow. Although this might seem strange, this is a deliberate choice. It is based on the belief that contributing users should earn more rights to download. For example: if Alice uploads 267MB, she will have earned enough points to download 628MB. And Carol, who does not upload at all, will eventually be deprived of points and will have to upload something before she can continue.

According to the incentive mechanism a new user will receive 4096 points to start downloading immediately. This allows a user to download 4GB worth of 1MB-sized files, or 5.2GB worth of 400MB-sized files, or 6.8GB worth of 800MB-size files. These initial points allow a new user to download enough files to be able to upload something at a later time and without this upload a user would eventually use up all her points, rendering her account useless. But because there are no costs associated to registering a new account, it is only a small task for a user to create a new account and start with 4096 points again. In [19] it is estimated that free-riders, like whitewashers, are responsible for 51% of the downloads but only 7.5% of the uploads. Because of the central server it is possible to detect whitewashers, to some extent, but currently this behavior is not being punished.

Each user has her own point score, which is stored in the central server, these points can be used to receive downloads, but they are also used to give a priority between users, when resources are scarce. This point score, is global knowledge and therefore it is likely to be susceptible to collusion attacks. Several distinct attacks have been registered in Maze, we will discuss both spam account collusion and group-based collusion.

- *Spam account collusion* occurs when a user registers more accounts besides her 'main' one, and uses them to gain points. Because of the free registration, this is easy to do. The setup is as follows: Carol has one 'main' account: C_1 , and two spam accounts C_2 and C_3 . Carol sends a large file from C_1 to both C_2 and C_3 . She can continue to send data until C_2 and C_3 have

no more points left to allow them to download. In the meantime the point score of C_1 is growing by 1.5 points per MB that is uploaded to C_2 or C_3 . The following two detection methods are from [11]: The *repetition-based collusion detection* works when a colluder uses the same file when uploading to her spam accounts. In one occasion a colluder was found to have transferred the MSDN DVD, of around 3GB, repeatedly for 29 times. And another way to detect this collusion is by looking for a single peer that uploads large amounts of data to a selected number of peers before moving on to upload to other select peers. (The normal behavior of a peer would be to upload relatively small parts to a wide variety of peers.)

- *Group-based collusion* occurs when two or more peers use each other to boost their points. This is usually done in groups of two. In this case Carol uploads data to Dave, and Dave uploads data to Carol. Because downloading costs less than what is gained by uploading, both Carol and Dave end up with more points. Sometimes this collusion is done in circles, where Carol will upload to Dave, Dave to Eve, and Eve back to Carol.

5.3 KaZaA

An overview of the KaZaA overlay network and search mechanism is provided by [12]. KaZaA does not use a dedicated server for tracking and locating content. However, unlike in other peer-to-peer networks like BitTorrent, not all peers are equal. KaZaA has two classes of peers, Ordinary Nodes (ON) and Super Nodes (SN). The more powerful peers are SNs and have greater responsibilities. As shown in figure 5.1, each ON is assigned to a SN. When an ON launches the KaZaA client, the ON establishes a TCP connection with a SN. The ON then uploads to its SN metadata for the files it is sharing. This allows the SN to maintain a database which includes the identifiers of all the files its children are sharing, metadata about the files, and the corresponding IP addresses of the ONs holding the files. The two tier approach of KaZaA exploits differences between peers. The peers in the higher tier are the more powerful in terms of connectivity, bandwidth, CPU power, and non-NATed accessibility.

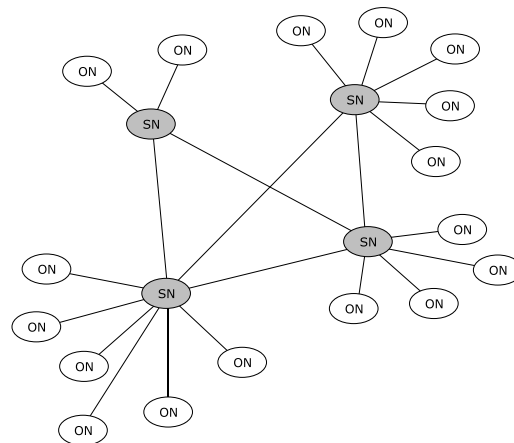


Figure 5.1: Supernodes and Ordinary nodes in a KaZaA network

For each file that it is sharing, the metadata that an ON uploads to its SN includes: the file name, the file size, the ContentHash, and the file descriptors (for example, artist name, album name, and text entered by users.) The file descriptors are used for keyword matches during searching. The ContentHash provides each file with a hash signature, this signature is used to verify that the file was not corrupted during the transfer.

When a ON wants to find a file a search request is send to the SN. The SN searches a local database and forwards the request to the SNs that she is connected to. Because SNs make and break connections between each other, the available sources for a file change in time, resulting in a global search instead of a local one.

A disadvantage of using KaZaA is that it is very difficult to install without installing several other applications as well. These other applications include programs what some, if not most, people would call Spyware. Spyware is software that transmits personal information to external sites where it is usually used to provide insight into the users behavior and to provide specific ads to increase effectiveness of adverts on web sites. [14] describes several Spyware techniques, the following two are applied to the standard KaZaA client.

The GAIN AdServer software, which is installed along side KaZaA, collects and transmits the information regarding the web surfing behavior and other non-identifiable information of the user to its remote servers and creates a user profile. The privacy policy of GAIN network says that it collects information about the web sites visited, the amount of time spent on each web site, number of clicks, user response to online advertisements, web log information, system settings, software and versions used, first name, country, city, postal code and other non-personal identifiable information on web pages and online forms and gator-eWallet password if the user has one. It also reads the cookies stored by other web sites and transmits to their own servers and third parties to request more appropriate ads. Once the user profile is created it allocates memory on the user machine. Then it will download and store the pop-up advertisements and banners that match the profile. It triggers these pop-up advertisements and banners in real time when the user is surfing the related content on the web even though the software with which it was distributed is not active. Though they claim that they do not collect any personal identifiable information, the first name along with postal code, city makes some sort of personal identification and is objectionable to many users. The GAIN AdServer regularly communicates with their remote servers and rarely third party servers to update and maintain the gain supported software, to retrieve advertisements and banners. And also depending upon the number of advertisers the GAIN network is supporting, the GAIN Ad- Server takes sufficient amount of memory, bandwidth and the power of the machine thus degrading the performance of the user machine.

Cydoor, which is also installed along side KaZaA, also delivers advertisements based on the user behavior of web surfing. Cydoor collects user data such as gender, age, interests, marital status, salary, area code, country and education and

distributes it to the parties that advertise with the Cydoor software. The Cydoor component works in the same way as GAIN AdServer by allocating some memory to download and store the advertisements. It also has an auto-update functionality that regularly contacts its servers and third parties to update their ads. The third party ad servers associated with cydoor technology makes use of cookies. Cydoor works with third-party ad servers such as ValueClick, Commission Junction, Advertising.com, RealMedia. com, BeFree and others to serve advertising to the Cydoor network. Cydoor in its privacy policy states that the information collected by these third parties are not under the control of Cydoor but they behave according to their own privacies. All of these are basically adware that serves advertisements based on the user web interests. ValueClick serves ads and its advertisers pay to ValueClick only when the user clicks on the ads displayed. ValueClick does not collect any personal identifiable information. It collects information such as browser type, IP address, Operating system and version, web pages visited, user response to ads etc.

Many users today use KaZaA-Lite. KaZaA-Lite is an unofficial client that emulates the official one and participates in the KaZaA network. During the search process, a KaZaA-Lite ON first sends its query to the SN to which it is connected. The ON then often disconnects and connects with a new SN, and resends the query to the new SN. During a specific search, the ON may hop to many SNs. The ordinary node typically maintains the TCP connection with the last SN in the sequence of hops, until another search is performed. Another important reason why KaZaA-Lite became very popular is that KaZaA-Lite sends a larger participation level to other nodes than the official KaZaA client would. A client with a higher participation level will get priority over those with a lower participation level. This level increases when more files are shared, but KaZaA-Lite lies about this level in order to gain an advantage over the original KaZaA client. Other differences between KaZaA and KaZaA-Lite are that KaZaA-Lite can refuse to provide other clients with a list of shared files, it also allows the user to decide whether not to be a Supernode, and it is rumored to have no Spyware.

5.4 MojoNation

MojoNation is an example of a peer-to-peer network that did not become a success. MojoNation was a scalable and secure marketplace to publishing and share files. The system used a digital currency called Mojo to compensate a user when she provided a service for another user. When the system was first released to the public in July 2000 it was used by more people than during initial development and small bugs in the protocol became apparent. An example of this was the 'hello' message that was automatically send. Peers were able to change the Mojo cost associated with this response and thereby 'steal' large amounts of Mojo from other peers. The developers had to fix these bugs by changing the protocol which was

a time consuming job. There are many aspects that contributed to the eventual failure of MojoNation, but one of the programmers contributed [18] it mostly to the following two reasons:

When a peer connected to the network for the first time she did not yet know any other peers. In order to introduce a new peer to other peers in the network a single introducer node was used. This did not give any problems until in October 2000 an article called 'Forget Napster & Gnutella: Enter Mojo Nation' was posted on the website slashdot.org. This publicity caused an immense increase in new users. While the previous day only 300 copies of MojoNation were downloaded, after the article this had increased to almost 10,000 copies per day. The central introducer was completely overwhelmed and since most of the new users were unable to enter the network it severely reduced the performance. The problem was solved by introducing more central introducer nodes into the network, but it took several days before this was implemented and the damage to the reputation of MojoNation was already done by then.

The second reason for the failure of MojoNation was centered around the availability of the data. In the design of MojoNation the decision was made to split a file across several nodes, thereby making it more easily available to others. The number of peers that were required to rebuild the file was equal to half of the total number of shares that were generated. This worked perfectly when there were enough of more peers available, but when less peers became available this tactic had the opposite effect and the file could no longer be completed. And when the users noticed that they could no longer acquire what they wanted they stopped using MojoNation, further decreasing the availability of files. In this respect the designers of MojoNation greatly overestimated the time that users would be online. Although there were peers that were online for long periods of time, they could not compensate for the vast majority of users that used analog modems to connect to the internet for only short periods of time.

Chapter 6

Conclusion

Peer-to-peer networks exist to provide service to users. Much can be gained for the community as a whole by using the available resources of individual users. However, this is at the risk that individuals will attempt to use the community to their own advantage, i.e. free-riders.

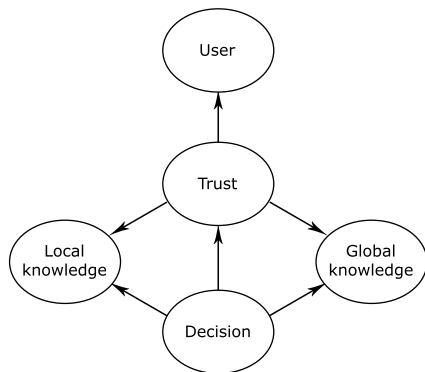


Figure 6.1: *Decision based on trust*

A peer in a peer-to-peer network that is designed to only use local knowledge can only learn from personal experience. Although this is a safe approach it is not always scalable as the community grows. Also free-riders can often misuse the free identities of the system to endlessly keep receiving service. Local knowledge will only be secure when a user can only have a single identity, i.e. by making identities expensive or time consuming to acquire.

Using global knowledge in the form of a reputation system, tokens, or counters solves the scalability issues. It also gives an incentive for users to be good to the community because this behavior will become known to more users in a shorter time period. However, global knowledge is relatively easy to influence by colluding with other users or spam accounts. Global knowledge will therefore only be secure when no false claims about identity or behavior can be made, which is difficult, if not impossible.

The risks of local and global knowledge can be reduced by using the weighed opinions of other users. The weigh factor is based on how much trust is placed on these opinions. Although these trust or reputation systems help, they will be unable to solve the problems because they depend on the same misleading information that they attempt to secure: local and global knowledge, see figure 6.1. The only way

to completely secure a trust system can be accomplished when the user provides trust values for all other users with whom she interacts. Unfortunately this is not desirable for a peer-to-peer network where the enormous amount of users is one of the biggest advantages.

Fortunately it is not necessary for a peer-to-peer network to be completely fair towards all participating users. There will always be users that provide more service than others, perhaps because they are one of the few available sources or they simply want to be helpful towards the community (i.e. altruistic users.) A consequence of some users providing more service is that others will have to provide less. The difficulty in designing a peer-to-peer system lies in ensuring that there are enough incentives to get people to provide service and that it is difficult for people to deviate from the specified protocol.

Bibliography

- [1] E. Adar and B.A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10):2, 2000.
- [2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1985.
- [3] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 6, 2003.
- [4] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. *Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, 2004.
- [5] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Havelaar: A Robust and Efficient Reputation System for Active Peer-to-Peer Systems. Computer Engineering and Networks Laboratory (TIK), ETH Zurich, CH-8092 Zurich.
- [6] D. Hales and S. Patarin. How to cheat bittorrent and why nobody does. Technical report, TR UBLCS-2005-12, Department of Computer Science University of Bologna, May 2005.
- [7] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on Gnutella revisited: the bell tolls? *Distributed Systems Online, IEEE*, 6(6), 2005.
- [8] S. Jun and M. Ahamad. Incentives in BitTorrent induce free riding. *Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 116–121, 2005.
- [9] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. *Proceedings of the twelfth international conference on World Wide Web*, pages 640–651, 2003.
- [10] Q. Lian, Y. Peng, M. Yang, Z. Zhang, Y. Dai, and X. Li. Robust Incentives via Multi-level Tit-for-tat.
- [11] Q. Lian, Z. Zhang, M. Yang, B.Y. Zhao, Y. Dai, and X. Li. An Empirical Study of Collusion Behavior in the Maze P2P File-Sharing System. Technical

- report, Technical Report MSR-TR-2006-14, Microsoft Research, February 2006.
- [12] J. Liang, R. Kumar, and K.W. Ross. Understanding KaZaA. *Submetido para publicacao*, 2004.
- [13] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing P2P reputation systems. *Computer Networks*, 50(4):472–484, 2006.
- [14] L. Nandikonda. Users Should Be Concerned of Spyware in Free P2P Software.
- [15] S.J. Nielson, S.A. Crosby, and D.S. Wallach. A taxonomy of rational attacks. *Proc. 4th IPTPS*, 2005.
- [16] A. Singh and L. Liu. TrustMe: anonymous management of trust relationships in decentralized P2P systems. *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, pages 142–149, 2003.
- [17] K. Tamilmani, V. Pai, and A. Mohr. SWIFT: A system with incentives for trading. *Proceedings of the 2nd Workshop on Economics of Peer-to-Peer Systems, June*, 2004.
- [18] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. *1st International Workshop on Peer-to-Peer Systems*, 2002.
- [19] M. Yang, Z. Zhang, X. Li, and Y. Dai. An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. *IPTPS.05*, 2005.