A Peer-to-Peer Incentives Mechanism For Sharing Small And Rare Files

Torsten Ackemann

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

2011

Declaration

I, Torsten Ackemann, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

The peer-to-peer paradigm is an important alternative to the traditional client-server model in computer networks, making up a significant share of the bandwidth used globally.

In client-server scenarios there usually is an external reason of why the server provides its service to the clients. But there usually is no external incentive in peer-to-peer networks to share data. In fact there are two good reasons not to. Firstly, providing data to another node consumes bandwidth, which will always be limited and whose use may might incur a cost. Secondly, the process of making data accessible is also costly. The data needs to be obtained, its existence needs to be advertised and individuals need to decide which data to share. An incentive is required for nodes to offer their resources.

We propose a generalisation of the BitTorrent incentives mechanism that improves it in two important ways. It works for a broader range of files in terms of size and popularity, enabling a simple BitTorrent-like tit-for-tat incentives mechanism for files that do not work with BitTorrent. At the same time it provides peers with an incentive to share more files.

In BitTorrent, peers download pieces of the same file from each other. This is a bartering ring of length 2. Our algorithm extends this idea by allowing pieces of different files to be exchanged and by allowing longer rings with more nodes to be formed. For this, rings need to be identified in an overlay graph that consists of the nodes and potential downloads among them. But no node has knowledge of the graph other than its direct neighbours. For the incentives mechanism to work once rings have been identified, a group consensus needs to be reached to start the downloads. We propose distributed algorithms for these problems and evaluate them experimentally using a simulation. We are able to show that in some cases our incentives algorithm works better for small and rare files than BitTorrent.

Acknowledgements

First I must thank my PhD supervisor Wolfgang Emmerich for his never-ending patience. Without his continuing support this work would not have been possible. His guidance on my research and on writing a dissertation was essential. I would like to also thank my second supervisor Cecilia Mascolo for her support, especially in developing the topic of this thesis.

I am very grateful for the financial sponsoring of my studies by Kodak Eastman Ltd. I am also grateful for the financial support by the RESERVOIR and PLASTIC European Union projects that allowed me to finish this thesis.

I would like to thank Mark Handley, Søren-Aksel Sørensen, David Rosenblum, and Steve Hailes for their very helpful advice at various stages of my PhD studies. Thanks also to Franco Raimondi for teaching me mathematics, once more. I'm also very grateful for interesting discussions on the topic with Richard Gold, Mohamed Ahmed and Vladimir Dyo.

Thanks to the technical support group and especially Denis Timm. Thanks to all the many people at the UCL Department of Computer Science, who were all so incredibly helpful and essential for the the productive and supportive environment that was so important for the creation of this work.

Thank you Stefanie for having supported me every day with your love and understanding.

Thanks to my family for keeping me grounded. Finally I want to express my gratitude to my parents Werner and Gerda Ackemann. Their support in every conceivable way created so many opportunities in my life which I then simply had to take.

Meinen Eltern gewidmet.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Peer-to-peer networks, or more precisely file sharing networks, are firmly established as one of the major applications on the Internet. A large share of Internet traffic is generated by file sharing networks. In 2005 up to 50% to 60% of upstream traffic and over 70% of downstream traffic was generated by peer-to-peer applications. (Parker, 2005)

While many peer-to-peer downloads are, in fact, illegal, there is an ever growing list of legal applications and content. Linux distributions such as (Fedora, 2010) use BitTorrent as a way of distributing DVD images, and popular applications such as (BBC iPlayer, 2010) and (Skype, 2010) also make use of the principle.

Since there are no dedicated servers in a peer-to-peer network, all services are provided by autonomous peers to each other and hence all peers are in the role of servers as well as clients. This has a number of advantages such as inherent scalability and reliability and a highly dynamic adaptation to constantly changing network conditions. But there are also a number of problems that arise from such a massively decentralised infrastructure. There is no central server to maintain a database of peers, the services they require and the services they offer. How do queries reach a peer that can provide the requested service? The network is essentially anonymous and peers often have difficulty judging each other's reliability and trustworthiness. Peers do not have a reason to provide services to others.

This is the point that we will concentrate on in this work. Peers need an incentive to provide services to others. A number of different approaches have been suggested, but they are either too heavyweight to be of practical use, or they are too specialised, working only in limited scenarios. In this work, we will investigate a new approach that

is both universally applicable to many different types of services simultaneously and lightweight in resource usage at the same time. We will focus on the special case of file sharing, or more specifically on extending file sharing to small and rare files.

1.1 Peer-to-Peer Networks

A peer-to-peer network is an overlay network that typically consists of a very large number of nodes. Each node is able to provide services to others and has the same role as the other peers in the network. There are no dedicated servers or dedicated clients, all nodes are peers of each other.

This approach has some inherent advantages over the traditional client-server paradigm. The main advantage is that it scales well. It works with a very large number of nodes without incurring too much additional load on nodes when compared to a small network. A peer-to-peer network is also typically much more resilient as there is no single point of failure. If one peer leaves the network, another peer will likely be available to take over its functionality. The absence of a centralised server also means that there is no centralised control over the system.

In the case of file sharing applications, peer-to-peer networks typically provide a basic infrastructure as well. Peers can find out about each other through the overlay network, and there is some form of service lookup or file search function available. In Gnutella for example, each peer has a number of direct neighbours, and search requests are received from and forwarded to those neighbours. This way, every peer is connected to every other peer through the overlay network.

More generally speaking there are two distinct functionalities in the network. The first functionality is that peers can provide services to each other. The second functionality is that peers can look up services and/or other peers, i.e. there is a service directory or similar. For example, in the case of Gnutella, there is HTTP based file transfer as the service and the aforementioned built-in file search capability as the service lookup function. We will discuss Gnutella in more detail in Section 2.7.

Since the topic of this thesis is to examine the *cooperation* among peers, we assume that there is a peer lookup function as well as a service search function already available to all peers in the network. We further assume that any peer can potentially communicate and exchange services with any other peer, although peers are not fully

aware of the condition and size of the complete peer-to-peer network. This is essentially the case for nodes in today's Internet.

1.2 The Problem: Cooperation

One of the main problems of peer-to-peer networks is that peers, unlike servers in traditional client/server settings, usually do not have a reason to provide services to any other peer. Each peer is an autonomous entity with no direct ties to other peers. This is different to a client/server setting where there is a motivation for the server to serve the clients, for example because they both belong to the same owner, or there is a service agreement between the operator of the server and the operator of the client, or the owner of the server has cost-reductions elsewhere (as in e-banking) or generates revenue through the service (as in e-commerce).

However, there are still some peers that do provide a service to other peers. Those peers typically do it out of ignorance, for not even being aware of it, or out of indifference, as their uplink bandwidth would simply go un-used otherwise and their own download service quality does not suffer from uploads. There are some other possible motivations such as idealism (someone has to provide services for the peer-to-peer network to work), altruism, or because there are external motivations (the serving peer has an interest in the file being distributed).

But since there usually is no reason to provide services to other peers, most peers in fact choose not to provide services to anyone if they can avoid it, for example in peer-to-peer networks like Napster or Gnutella. This was revealed by several measurement studies, most notably those from (Adar and Huberman, 2000), who showed that almost 70% of Gnutella users shared no files and nearly 50% of all responses were returned by the top 1% of sharing hosts.

Peers not contributing is foremost a problem of fairness, but more importantly the service quality of the network as a whole suffers badly from this, because only those services that are on offer can be consumed. Thus, if only few peers offer services, only those few services can be consumed. To put it differently, the average service quality in the network is the better the more the peers contribute to it by providing services to other peers.

In short, peers are typically not willing to *provide* services, but they participate in

the peer-to-peer network in order to *consume* services. Since only peers can provide services in a peer-to-peer network due to the absence of a server, it is essential that peers have a reason to offer and provide services to other peers; they need an *incentive* to cooperate.

1.3 Shortcomings of Current Approaches

Most of the important innovations in peer-to-peer technology were introduced by file sharing applications and not by scientific research. This has only changed in recent years with the introduction of distributed hash tables (DHTs) into popular file sharing applications. DHTs, which were developed in 2001 simultaneously by a number of groups, provide the fundamental technology for a fully distributed information lookup system. Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron and Druschel, 2001), Tapestry (Zhao et al., 2002) and Kademlia (Maymounkov and Mazières, 2002) are the best know examples. For a full picture of the state of the art, we thus will have to look at both scientific publications as well as technologies used in existing peer-to-peer networks.

While the first generation of applications such as (Napster, 2010) and (Gnutella, 2010) did not have any incentives mechanism, simple incentives schemes were implemented for the next generation of applications such as (EDonkey, 2010). In parallel, academia was looking at the problem on a more general level, either with peer-to peer networks or with mobile ad-hoc networks in mind. Both are somewhat similar in that they deal with a number of independent nodes.

There are two basic types of mechanisms used widely. The first type are reputation based systems, where peers know about each other over a long period of time and act towards a peer according to past behaviour of that peer. (eMule, 2010) is an example for this approach. The second type are incentives based mechanisms, where the design of the network provides peers with a reason to cooperate. The most prominent incentives based models are based on a tit-for-tat like mechanism, meaning that a peer only gets something if it gives something, and the more it gives, the more it gets in return. BitTorrent (Cohen, 2003) is an application that uses this approach. There is a third type of approach, establishing a micro-economy between the peers, but after the demise of Mojo Nation (Wilcox-O'Hearn, 2002) there is no network left based on this approach.

All three types of networks are discussed in much more detail in Chapter 4.

When looking at the literature, a number of different approaches have been suggested, but all of these approaches seem to have one of two problems. Either they are *universal but too resource intensive*, like a reputation based scheme or a micro currency, or they are *lightweight but too specialised* and are efficient only in certain scenarios, like BitTorrent. The different approaches and their shortcomings are discussed in detail in Section 2.7.

Too Heavyweight

There are a number of universal but heavyweight approaches such as reputation based models and micro currency models, but these approaches do not take the general abundance of resources on the peers and their volatile nature into account. Only a fraction of the bandwidth and the CPU time is typically used on most peers, and those resources cannot be saved for future use. The files that peers offer are typically already present anyway and so also represent no significant investment to them. Thus, offering resources to others without getting a ”fair” value in return does not necessarily lower a peer's utility.

Therefore, an overly fair and accurate scheme that enables peers to save resources that will then be wasted only introduces additional complexity to the system without yielding an actual gain. Many popular approaches require a high communication overhead to work, while even a simple approach can increase the average service quality considerably. (Ranganathan et al., 2003)

Too Specialised

Current tit-for-tat based approaches are usually lightweight but too specialised. They work very well in certain scenarios as the success of BitTorrent shows, but they make assumptions about the network or about the peer's preferences that limit their usefulness to a small number of well defined scenarios. For example, BitTorrent does not work well with small files or rare files, or when nobody has an active interest in the distribution of the file and is willing to provide it (*seed* it).

1.4 Requirements for a Better Approach

In this thesis, we are looking for a new incentives mechanism for peer-to-peer networks that is both lightweight and universal at the same time. Ideally it should require as little effort by the peers as possible while being scalable to networks with massive number of peers. Its performance should match current mechanisms while being more versatile. It should also work with many different services simultaneously, as that is the only way of providing an incentives mechanism to unpopular services (such as rare files in file sharing).

The most important requirement for a better approach is that it is both **lightweight** and **universally usable** for a broad range of scenarios simultaneously in the same network. By lightweight we mean that no or very little state needs to be maintained, and by universal we mean that it should work for broad range of peers and resources, both popular and not popular ones, at the same time in the same overlay network.

Furthermore, a better approach should fulfil a number of additional requirements. We understand that these requirements are rather broad and that they are partly contradictory.

decentralised The system does not rely on a centralised server to provide services. This is what constitutes a peer-to-peer network.

not implementation dependent Some simple incentives schemes are hard coded into the actual software and can easily be circumvented by using a modified client in which this functionality is disabled. This approach also does not work with open protocols and standards. The original eDonkey client contained this functionality (see Section 2.7.1).

adaptive The incentives scheme needs to be highly adaptive, constantly adapting to the ever changing environment. Peers will permanently start and stop services and this will have to be taken into account by the mechanism. This is not critical for example in a currency based system but essential in a tit-for-tat based system such as BitTorrent (see Section 3.1.2).

robust The system should be robust towards unexpected behaviour. It will be designed to provide the optimal performance, but it has to be able to cope with colluding

peers that do not independently strive for maximum utility but which instead work together to help gain one particular peer an advantage at the expense of other unrelated peers. Trust based systems are particularly vulnerable to this, see Section 2.7.2.

secure There is also the more direct attack on the network, where peers try to actively sabotage the network, for example by using denial of service attacks or by manipulating the provide services e.g., providing wrong data in the case of file sharing. The system should be resilient to these attacks.

1.5 Thesis Contribution

In this thesis we have developed a novel tit-for-tat mechanism to incentivise nodes in a peer-to-peer network to share small and rare files. We analyse the problem, develop distributed algorithms for our proposed solution, design a peer-to-peer system around these algorithms and implement and evaluate them using a simulation.

The specific contributions that have resulted from this research are as follows:

1. We analyse the problem of peers cooperating over time with sets of other peers, and using game theory aim to develop a distributed incentives compatible direct-revelation mechanism. This game theoretic analysis assists us in understanding the problem and in expressing the desired outcome. We then show why mechanism design using current knowledge in the field does not assist us in the design of an incentives mechanism, which leads us to pursue an engineering approach in the remainder of the thesis.

2. We generalise the BitTorrent algorithm to enable its use in a much broader range of settings. By allowing pieces of different files to be exchanged and by extending the tit-for-tat relationship from two peers to a ring of peers, peers play tit-for-tat against the ring instead of against one other peer.

3. We devise a token based distributed algorithm to discover rings (or cycles) in the service overlay graph that is constructed from the peers and potential or actual services provided between those peers.

4. We give a message based distributed group consensus algorithm to manage those incentives rings. Members of a ring need to coordinate starting and stopping of services for the tit-for-tat principle to work.

5. We design a peer-to-peer system based on the aforementioned ring algorithms. This includes a message-based communication infrastructure and concrete specifications and pseudo-code implementations of the ring algorithms.

6. We propose the design and implementation of a Java-based simulation based on the system. This includes implementations of all the algorithms and modules of that design. It also includes an implementation of the BitTorrent algorithm for comparative purposes in the evaluation. We compare the performance in a Gnutella scenario and we discuss the performance limitations of our algorithms.

1.6 Thesis Outline

Chapter 2 uses game theory to analyse our problem. We introduce mechanism design and describe the application of mechanism design to devising an incentive mechanism. While this analysis aids in understanding and framing the problem examined in this thesis, it fails to deliver a suitable incentive mechanism. At the end of the analysis, we review related work, both in academic publications as well as peer-to-peer network implementations. We discuss the incentives mechanisms in early peer-to peer networks, and then at the three main areas of current incentives mechanism: trust/reputation based systems, micro-currency based systems and tit-for-tat based systems.

In Chapter 3, we discuss the principles of our solution to the problem. We explain how the tit-for-tat principle can be generalised to incentives rings that are applicable to most services and peers in heterogeneous environments. We introduce our incentives algorithm that is based on sharing pieces among files and on forming incentives rings. We then discuss how to find those rings and how to manage them. In the related work section we discuss ring-based systems and BitTorrent-like systems.

While most of Chapter 3 focuses on the network as a whole or at best on complete rings, in the following Chapter 4 we present the design of distributed algorithms that are used in the proposed solution. This chapter focuses on the peer's point of view and how the principles could actually be implemented. We discuss the four building blocks

peers, files, services and rings, and their integration.

Chapter 5 contains the evaluation of our proposed solution. We discuss the evaluation method and argue why our proposed solution is best evaluated with a simulation. We describe the simulation that implements the distributed algorithms that we have presented in the previous chapter. We use an event-based simulator tool specifically tailored for large scale peer-to-peer networks. This chapter also contains a discussion of the implementation and an identification of the scenarios we need to simulate. We then discuss the results of the simulations that we performed and can show that our solution outperforms the state of the art in important ways.

Finally we provide a summary of this thesis and make suggestions for future work in this area in Chapter 6.

Chapter 2

Problem Analysis

In this chapter we will formally describe and analyse the problem at hand. First we specify the constraints and requirements for our problem in an attempt to narrow it down to a level that can be solved. Then we describe the scenario and the limiting assumptions we make in order to get to the core of the problem. As related work discovered, this problem area is best modelled using game theory, a sub discipline of micro economics. We will use game theory as well to formally model the problem. At the end of this chapter we will have a look at mechanism design, the science of designing games. In theory, mechanism design should lead us towards a solution, but we will show why this does not prove to be the case for our problem.

2.1 Constraints and Requirements for a Solution

There are a number of constraints that we have set. The first is that there are *no external incentives* in the system. There are no ulterior motives for peers to provide a service to others, such as a movie studio having an incentive to distribute move trailers. There is no payment of any kind between the owners of peers and there is no micro currency to facilitate payments within the network.

While this constraint does not have to be true for every single peer, as some peers for example might choose to share files out of altruism alone, it is a defensive worst case assumption. If the mechanism works well with no altruistic peers then it will only work better with them.

The second constraint is that there is no static peer identity and as a consequence there is no widely spread trust/reputation model already available in the network, and establishing one specifically for the peer-to-peer network creates too much overhead in

terms of both required infrastructure and mental transaction costs for the owners of the peers.

The main requirements for our approach are that it speeds up the download of small files and rare files at least in some scenarios while not being significantly worse than BitTorrent for large and popular files.

2.2 The Peers and the Network

A typical peer-to-peer file sharing network consists of hundreds of thousands of peers or more. The peers all have a set of limited resources, such as bandwidth, CPU time or available disk storage. We assume that some of these resources are un-used at least some of the time. Peers allow other peers to use their resources, typically in form of a service such as downloading a file.

The peer-to-peer network establishes a search infrastructure that allows peers to find each other's files. In addition to the search infrastructure, there is a way for peers to provide services to each other. In the case of file sharing this would be the download of a file from one peer to another, supported by the underlying TCP/IP protocol layer.

Typical peer-to-peer networks operate on top of the already existing Internet. That means that from an Internet point of view, the peer-to-peer network is entirely an application layer network. From a peer's point of view that means that generally communication with any other peer is possible, a concept called openness. Thus, services can be provided to and requested from any of the other peers. We assume that there are no issues with network modifying technology such as NAT (network address translation) or firewalls.

The peers and the services can be seen as a *service graph*. In this graph, the peers make up the nodes and the service relationships between the peers are directed edges in the graph (see Figure 2.1). There can be more than one service from one peer to another, but that still results in only one directed edge in the graph.

In many networks it is common that the edge between many nodes is bidirectional. This is true if larger files are broken into pieces and the two peers download different pieces of the same file from each other.

Peers only know about their direct neighbours in the service graph, and then only have limited information about them. They know if there was an upper limit to the

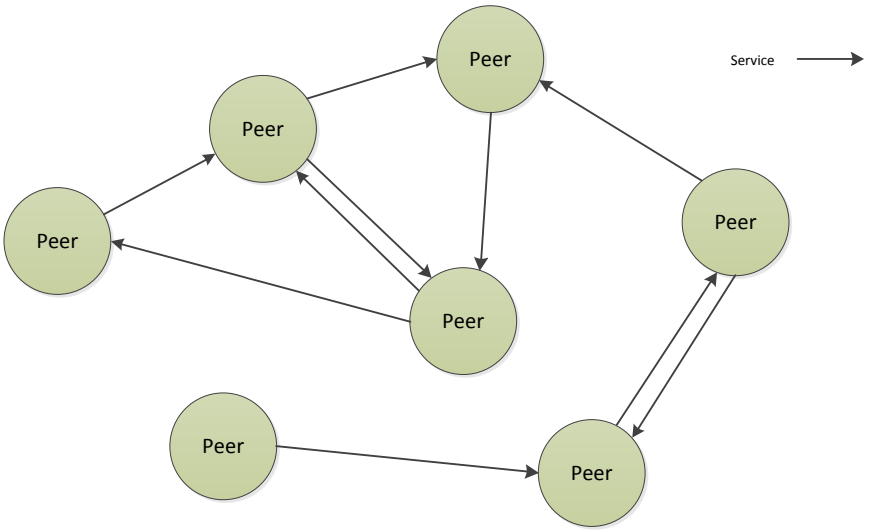

Figure 2.1: Service Graph

quality they could provide to ”downstream” neighbours, and they know what quality they receive from their üpstreamñeighbours. They also know that the service graph is dynamic and will evolve over time.

2.3 Assumptions

We will have to make a few assumptions to simplify the problem. Some of these assumptions simplify the model we have of a peer, while others relieve us from potential network trouble or hardware issues.

The first assumption is that surplus bandwidth and CPU time on a peer are wasted if they are not used for the peer-to-peer network. It is impossible to save CPU time for future use. This is a very straightforward assumption and it is necessary because without something to offer, a peer cannot expect to get a good service in return.

We also assume that peers act rationally and that all peers are independent of each other. This rules out collusion between peers, where some of the peers work together to gain an unfair advantage.

The next assumption is that there are no quality issues with the resources and that a peer can precisely control the quality of a service it provides. That means that there is no delay or jitter in the network, and that data streams can have a relatively constant throughput per time unit, at the speed that is chosen by the peer. This assumption is

necessary to keep the problem as simple as possible to be able to concentrate on the core of the problem. It is also a reasonable assumption since quality problems are most often found on the edge of the network, where peers or rather their owners can do something about those problems. Furthermore, without this assumption in place, it will be harder to maintain the assumption that peers act rationally.

A similar assumption that is essential for our work is that we are dealing with fixed peers only, or at least peers that are not constrained in terms of power, which in turn limits the ability to communicate with others at will. With these two assumptions we can assume that two peers can always communicate with each other and that they can provide a service to each other if they chose to do so.

We further assume that resources are relatively worthless to peers and therefore it makes sense for peers to trade un-used resources with other peers in the form of services. A peer would rather get a slightly better service quality by using its resources than wasting them un-used. Fairness, i.e. getting an equally valuable service in return, is not necessarily in the interest of a peer. The relative worthlessness of the resources in question make it uneconomical to install a complicated system for the allocation and for the accounting, charging and billing of those resources between the peers. This assumption is necessary because otherwise peers would be largely unwilling to provide services to others and the whole concept would not work. This assumption rules out peers on most mobile nodes, as those typically have significantly higher bandwidth costs.

Finally, we assume that there is a peer-to-peer infrastructure in place. Every peer can communicate and exchange services with every other peer, and there is a service lookup/search function available such as a distributed hash table or servers similar to BitTorrent trackers. This work is focussed on incentives and we will rely on existing solutions for search functions to avoid distraction from the main problem.

It is also important to note that we make absolutely no assumptions about the nature of the files that are being exchange. They can be in a known format or in an unknown format. They can be encrypted symmetrically so that only few people who have the key are able to read them. They can also be encrypted in an asymmetric fashion for a digital rights management platform. The actual content of the files is irrelevant to this work.

2.4 Game Theory

Now that we have described the scenario and made a number of assumptions that allow us to narrow down the problem, we can try to model it. We aim to create an abstract representation that allows us to examine and analyse certain properties of our problem and hopefully to find a possible solution.

When looking at related work, it appears that *game theory*, a branch of micro economics, is a suitable tool to model our problem and that *mechanism design*, another closely related field, provides a set of tools for designing a solution.

In game theory, a *game* is a series of one or more moves (out of a set of different possible moves) made by a number of different *actors*, either in sequence or simultaneously. The set of different possible moves is called *strategy space*. Each actor has its own *utility function*, which defines the value of a particular outcome of the game for this actor and which thus expresses the actor's preferences. The actor will then use a *strategy* in choosing its moves that maximises its utility. This is called *rational* or *selfish* behaviour. Depending on the nature of the particular game, the game might be played only once, or it might be repeated a number of times or infinitely. This allows actors to learn from the outcome of previous games and adapt their strategy for future rounds of the game. A game reaches an *equilibrium* if all actors play best strategies. There is usually a number of equilibria in a game.

As can be seen in Figure 2.2, from the peer's point of view, a peer is playing a game with all the other peers that it knows about through the quality of the services received and provided. Service qualities can be 0 of course.

2.5 The Game

The application of game theory to our problem is quite straightforward. We introduce a game which we call the *service quality game*. Each peer is an actor, and the possible moves that a peer has consist of the combinatorial set of the qualities of the services provided to all the other peers it knows of (where not providing a service to a peer is equal to a quality of 0). The utility of a peer is based on the service quality it receives from other peers. We introduce time slots, where each slot represents one round of the game. The infinitely repeated game consists of only one move in which the peers decide on the service quality they provide to others, based on their preferences and on

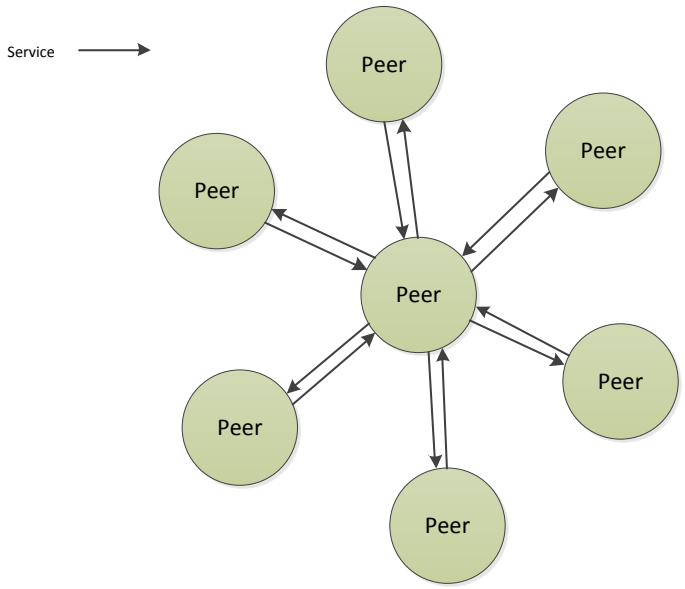

Figure 2.2: A Peer and its Neighbours

their observations in previous rounds. Service quality here denotes the actual amount of resources given to that service out of the maximum resources, i.e. in file sharing the upload speed granted to one particular peer out of the total upload bandwidth.

In order to describe the game formally, we will first introduce peers, services and peer types. Then, as the utility is comprised of the payoffs of the different services a peer is engaged in, we have a look at the payoff that a service generates for a peer. From this we can define the total payoff of a peer. This leads us to the utility function, which describes the relation between a certain outcome of the game and the associated payoff. We then talk about the strategy space of a peer and possible and useful strategies. Finally, we will consider possible stable outcomes (equilibria) of this system. Based on the description of the game from a peer's point of view, we will then describe the game (the mechanism) as a whole in the following section.

2.5.1 Formal Model Of The Game

Peers and Services

There is a set of peers and there are services being exchanged by them. A service is provided by one peer to one other peer.

Let $p_s, p_r \in Peers$ be the sending and the receiving peer.

Let $Services$ be the set with all possible services provided by any peer.

Then $s(p_s, p_r) : Peers \times Peers \mapsto Services$ is a service provided by peer p_s to peer p_r.

Peer Types

Each peer p has private information about its preferences and beliefs, the so called *type* $\theta_p \in \Theta_p$. θ_p is the most preferred outcome of the game for the peer. Θ_p is the peer's type set, the set of all different preferences of the outcome that a peer can have.

Generally, one cannot be certain which outcome a particular peer prefers, and so that unknown preference of peers has to be a variable of the system. For example, in a game of Russian roulette we cannot be sure about the recklessness of a player and how early or late he prefers to quit. This recklessness is then modeled as the type of the actor.

Quality of a Service

Let us first define the quality of a service between two peers. The quality of a service is predominantly determined by the resources that the providing peer allocates to it, but it may also be limited by the quality that the consuming peer can receive. For example, the consuming peer might not have enough bandwidth to receive a file download at the maximum speed that the sender would be willing to send the file with.

Let $t \in \mathbb{N}$ be the current time slot.

Let $q(s, t) : Services \times \mathbb{N} \mapsto \mathbb{R}$ be the quality of service s in time slot t (e.g. the transfer speed in file sharing) .

$q_{smax}(s, t) : Services \times \mathbb{N} \mapsto \mathbb{R}$ is then the maximum quality that the sending peer p_s is willing or able to provide to the receiving peer p_r.

And $q_{rmax}(s, t) : Services \times \mathbb{N} \mapsto \mathbb{R}$ is the maximum quality that the receiving peer p_r can utilise.

For services whose resources do not require capacity on the receiving peer such as CPU time, $q_{rmax}(s, t) = \infty$. Both of these qualities depend on physical limitations on the peer as well as the distribution of available resources on the services.

Now we can define the actual quality of the service s as the minimum of these two qualities:

$$q(s,t) = min(q_{smax}(s,t), q_{rmax}(s,t)) \tag{2.1}$$

The normalised quality $\hat{q}_{p_r}$ on the receiving peer p_r, which we use to abstract from the particular amount of resources available to a peer is then

$$\hat{q}_{p_r}(s,t) = \frac{q(s,t)}{q_{rmax}(s,t)} \tag{2.2}$$

with $0 \leq \hat{q}_{p_r}(s,t) \leq 1$, where 0 means that no service is provided and 1 means that the peer is willing to use all its resources for this service.

Utility of a Service

The previously defined service quality is independent of a peer's priorities and policies, it is simply a description of the level of resources that a peer receives. *Utility* on the other hand is the value that a given service quality has for a peer. There might be a linear relationship between the two, but that is not necessarily so. For example, any download at all is better than no download, but a faster download is not necessarily worth much more to a peer. Downloading a file in five minutes instead of ten minutes does not necessarily provide twice the value to the peer.

Thus we need a utility function, which is potentially different for each peer and which takes the service quality as the input and outputs the value of that quality:

$$u_p(\theta_p, s, \hat{q}) : \Theta_p \times Services \times [0..1] \mapsto \mathbb{R} \tag{2.3}$$

with θ_p being the peer's type set, s being the service, and $\hat{q}$ being the normalised quality of the chosen service.

See Figure 2.3 for a graph with three different types of utility functions. Risk-neutral ones, risk-averse ones (even low quality has enough utility to the peer) and risk-seeking (only high quality has a sufficiently high utility, a low quality service is almost useless).

Payoff of a Service

The payoff of a service to a peer now is the actual value of that service to the peer, given its quality and the utility function of the peer. This is the application of the utility function on the actual quality the peer gets in a specific time slot.

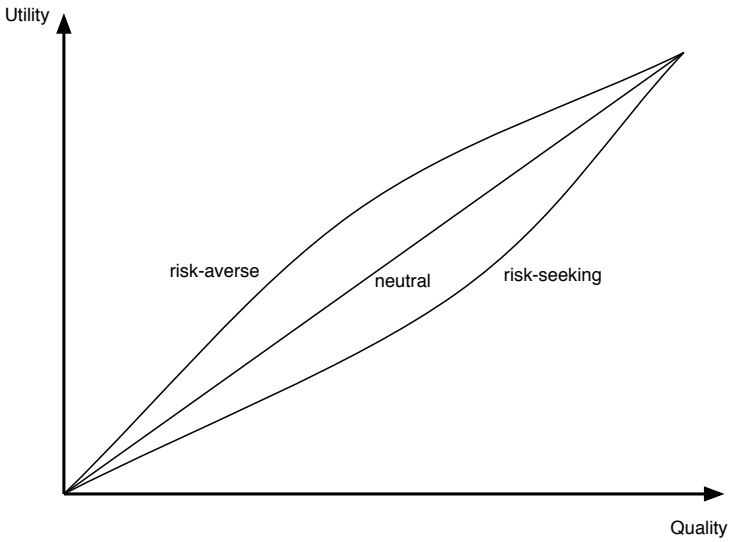

Figure 2.3: Utility Function

Let $t \in \mathbb{N}$ be the current time slot.

Let $p \in Peers$ be a peer.

Let $\theta_p \in \Theta_p$ be the peer's type.

Let $s \in Services$ be a currently provided service.

Let $q_p : Services \times \mathbb{N} \mapsto \mathbb{R}$ be the quality of the service s by peer p in time slot t.

Let $u_p : \theta_p \times Services \times \mathbb{R} \mapsto [0..1]$ be the utility for service s with normalised service quality q_p.

Then the payoff for peer p for service s is

$$\pi_p(\theta_p, s, t) = u_p(\theta_p, s, q_p(s, t)) \tag{2.4}$$

Total Payoff for a Peer

The total payoff of the peer is simply the sum of the payoffs of all services of the peer. Since provided services usually have a negative payoff and consumed services a positive payoff, we will separate them in the formula for clarity only.

Let $I_p \subset Services$ with $\{s_i \in Services | \exists p_s \in Peers; s(p_s, p) = s_i\}$ be the set of services provided to the peer p.

Let $O_p \subset Services$ with $\{s_i \in Services | \exists p_r \in Peers; s(p, p_r) = s_i\}$ be the set of services provided by the peer p.

Let $S_p = I_p \cup O_p$ be the set of all services on peer p.

Then the payoff for a peer is

$$\Pi_p(\theta_p, S_p, t) = \sum_{i_p \in I_p} \pi_p(\theta_p, i_p, t) + \sum_{o_p \in O_p} \pi_p(\theta_p, o_p, t) \tag{2.5}$$

Now we have a formula that describes the value that the current state of the game has to a peer. This allows a peer to express a preference of one state over another. For this, the peer needs to devise a set of actions, i.e. a strategy, to follow that will result in a desired state with a high payoff.

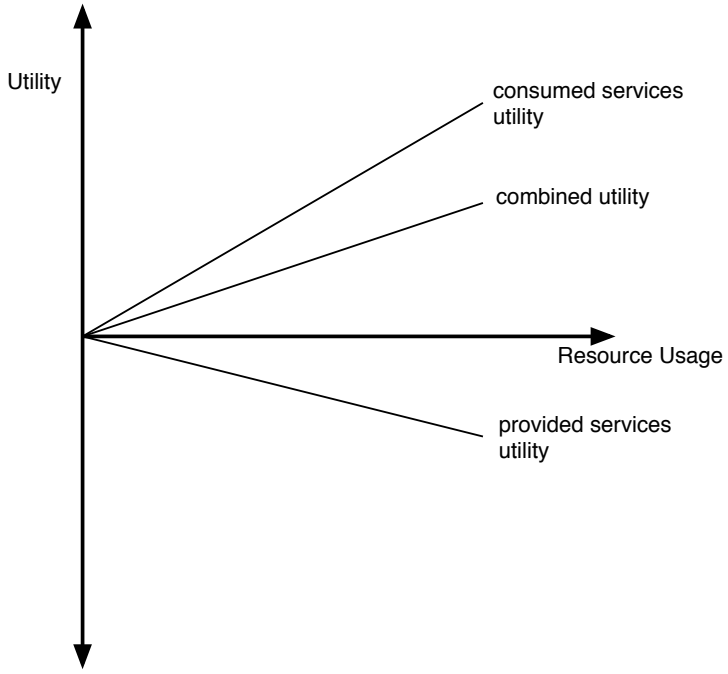

Figure 2.4: Combined Utilities

In Figure 2.4 we show why it is in the peers' interest to provide any services to others. This is because we assume that spare resources are fairly worthless to a peer, certainly less valuable than the services they receive from other peers. If we assume a linear relationship between offering a better service and getting a better service in return (the ideal situation), we will get a higher utility the more active we are (i.e. the more we provide).

2.5.2 Strategies

A strategy denotes all moves that a peer can make in the course of a game. This strategy is expressed as a *strategy function*, which generally speaking takes the current state of

the game as argument, for example in form of a state vector, and which returns the next move as the result. In our case of a single move game, the argument would then be state of the game after the previous game round, i.e. the moves made by the peers in the previous game (and optionally subsequently earlier games).

The *strategy space* then is an n-dimensional matrix comprising of all parameters of the strategy function. It contains all possible moves that can be made by the peer.

In our case, the state of the game is the list of services the peer provides and consumes together with their qualities. The output of the strategy function is then the list of services that the peer provides together with their qualities.

Let i_x and o_x be services, $x \in \mathbb{N}$.

Let $\mathbf{in}_p(t) = (q_p(i_1, t), q_p(i_2, t), ..., q_p(i_m, t)) \in \mathbb{R}^m$ be the ordered set of qualities of the consumed services of peer p in time slot t.

Let $\mathbf{out}_p(t) = (q_p(o_1, t), q_p(o_2, t), ..., q_p(o_n, t)) \in \mathbb{R}^n$ be the ordered set of qualities of the provided services of peer p in time slot t.

Let $\mathbf{s}_p(t) = \mathbf{in}_p(t) \circ \mathbf{out}_p(t) = (q_p(i_1, t), ..., q_p(i_m, t), q_p(o_1, t), ..., q_p(o_n, t)) \in \mathbb{S}^{m+n} \subseteq \mathbb{R}^{m+n}$ be the combined vector of all incoming and outgoing service qualities of a peer p in time slot t. $\mathbb{S}$ is the set of all possible quality vectors.

From the vector $\mathbf{s}_p(t)$ we can construct a matrix that contains service qualities for a number of time slots. Let $\mathbf{M}_p[u, m+n] = (s_p(1), ..., s_p(u)) \in \mathbb{S}^u$ be the quality matrix of u consecutive time slots.

Using the quality matrix we can then derive a quality scalar from the matrix so that $\mathbf{q}_p : \mathbb{S}^u \mapsto [0..1]$.

The strategy function $\phi_p : \mathbb{S}^{m+n} \mapsto \mathbb{R}^n$ then maps previously consumed and provided service qualities to the new set of provided service qualities for the next time slot.

More concretely, we are interested in the measurements from the last l time slots. Thus, in our case the matrix M contains those measurements. So the strategy function for peer p is in our case is the following:

$$\mathbf{out}_p(t) = \phi_p((\mathbf{s}_p(t-l), \mathbf{s}_p(t-l+1), ..., \mathbf{s}_p(t-1))), l < t \in \mathbb{N} \tag{2.6}$$

Types of Strategies

We have now specified a strategy function for our problem with a vast number of possible strategies. To get an idea of what strategies might be useful, we will now introduce some of the well known basic strategy types. To keep things simple, we will assume a strategy function with one scalar each as parameter and return value. This list was taken from (Félegyházi et al., 2006) and has been extended to include generous tit-for-tat.

Let $\sigma_p : [0..1] \mapsto [0..1]$ be the strategy function of peer p.

Hawk: Start with defection in the first game, then always defect. $\sigma_p(in) = 0$. A peer only ever consumes services and never provides any service to other peers. In peer-to-peer networks without an incentives mechanism such as (Napster, 2010), approximately 80% of peers follow this strategy.

Dove: Start with cooperation, then always cooperate. $\sigma_p(in) = 1$. The equivalent in a file sharing network would be a peer that always allows uploads of its files to other peers. Few peers follow this strategy, and those who do are either ignorant about uploads or have idealistic motives (”free information”).

Tit for tat: Start with cooperation, then mimic the opponent's behaviour. $\sigma_p(in) = in$. This strategy is the best one know so far for the famous prisoner's dilemma game (see (Binmore, 1992) for an explanation) and it has also been successfully used in many other games. The peer assumes that the other peer cooperates, but if it does not then it gets punished until it starts cooperating again.

Generous tit for tat: start with cooperation, then mimic the opponent's behaviour. Occasionally, cooperate despite possible defection from the opponent. $\sigma_p(in) = in$ and $\sigma_p(in) = 1$. This is a modified tit for tat strategy that avoids deadlock situations in which both peers continually deny cooperation. In scenarios where cooperation is generally the more desirable outcome of a game, it might be worth to be cooperative once in a while to give the other peer a reason to cooperate.

Rationally acting peers are selfish and will try to maximise their payoff. However, finding this optimal strategy is not straightforward for the peers. This is because peers generally do not have full information about the current conditions in the network and about the strategies that the other peers have selected.

So instead of directly picking the optimal strategy, peers will have to repeatedly chose a strategy, observe the results and then chose a more suitable strategy. This way, the peer's strategy will converge towards the optimum, or at least a local maximum, under the assumption of static conditions on every other peer. Since other peers are constantly adapting as well and thus the conditions are changing, this constant evaluation is required anyway, not just for convergence towards the optimum.

Equilibria

An equilibrium in game theory is a balanced outcome of a game. It occurs when all actors play their best strategy. The most well know equilibrium is the Nash equilibrium, where each actor has complete information about the game (the actors and their strategies) and plays the optimal strategy, i.e. always makes the best possible move. Thus, in a Nash equilibrium, no actor can do better by unilaterally changing their strategy.

Felegyhazi's theorems (Félegyházi et al., 2006) show that if we assume all peers play tit-for-tat, we have a Nash equilibrium if either no peer provides anything or if all service links are part of a ring. Both possibilities are not very useful to us because they do not occur often in practice.

In our scenario however, we cannot expect peers to behave fully rationally because they have incomplete information about the current state of the peer-to-peer network and because the computational complexity of considering all options might be too high for the given time span for a decision on a strategy (*bounded rationality*). Additionally, we cannot guarantee the required properties of the peer-to-peer overlay network.

Instead, we are looking for an equilibrium that a given static peer-to-peer network will converge to, assuming that all peers follow the same strategy. This strategy then would be a guideline for peers: following it would assure a decent quality of service. Ideally, the equilibrium should be *pareto-optimal*, meaning that no peer could be better off with another peer being worse off.

Peers however will not necessarily follow any strategy voluntarily, even if all peers would be better off if they all did. Instead, the rational strategy that we have to assume peers will follow is trying to maximize their benefit without relying on the strategy chosen by others. Quite likely, this will not lead to any useful equilibrium. Thus, we need a structure where the very properties of the peer-to-peer network will *force* peers to

follow a strategy that will lead to a useful equilibrium. The only way to force peers into a particular behaviour is to make it the rational strategy to follow. This is where *mechanism design* comes into the picture. Mechanism design is concerned with designing game mechanisms for a desired outcome.

2.5.3 Mechanism Design

A survey of the use of mechanism design in computer science and networks is given by (Dash et al., 2003). Mechanism design, which is like game theory a part of micro economics, deals with how to design games with independent actors that exhibit desired system wide properties, such as efficiency, stability and fairness.

For this to work in a game with rational/selfish actors, the desired properties must emerge in an equilibrium of the game. Typically there is more than one equilibrium in a game, and not all of the equilibria are useful. Thus, the equilibrium should be likely to be reached and it also has to be a good equilibrium with regards to the desired properties. As rational actors will only play best strategies, the game has to be designed in a way that rational behaviour of the peers leads to this good equilibrium.

Ideally we want to design a game where the dominant strategy for the peers, i.e. the strategy that guarantees the maximum utility for the peer, is simply to truthfully reveal their complete private information, instead of a complicated strategy that tries to maximise a peer's utility. A less optimal but still desirable mechanism is one where truth-telling is an equilibrium strategy for agents (a so called *incentive-compatible* mechanism).

An example is the sealed bid second price auction, also called Vickrey auction. Here, each bidder for an item hands in one bid in a sealed envelope, and the highest bidder wins the item for the price of the second highest bidder. Using this mechanism, the rational strategy for the players is to truthfully reveal the value that the item has to them. This property is called *strategy-proof* and is especially important in the case of computer programs as actors, since they cannot act fully rational.

The *revelation principle* states that any mechanism can be transformed into an *incentive-compatible direct-revelation mechanism*. "Direct" here means that the peer's strategy space is limited to reporting its types. (Dash et al., 2003)

This means that we can indeed assume that there is a mechanism for our prob-

lem that finds a utility-maximizing equilibrium, given complete information and fully rational peers.

In our case, we would like to design a mechanism and therefore game that has the goal of maximising the global payoff, i.e. the accumulated payoff of all peers.

Maximising global utility is one of the well understood goals in mechanism design and is expressed using the *social choice function* $f : \Theta \times t \mapsto \vartheta$. Θ is the set of all possible sets of agent types and ϑ is the set of all possible outcomes.

$$f(\theta, t) = max_{o \in \vartheta} \sum_{p \in Peers} \sum_{s \in S_p} u_p(\theta_p, s, q_p(s, t)) \tag{2.7}$$

In words, we are trying to find the set of peer preferences that maximises the accumulated utility of all the peers in time slot t.

There is however a problem with classic mechanism design with regards to our problem, in that in our case the conditions are very dynamic. Peers join and leave, and they also change their preferences and thus their type while they are in the network.

In fact, there are two even more severe problems that we have with mechanism design. The first problem is that computers only have limited computational abilities, meaning that peers cannot be fully rational, simply because they do not have infinite resources. The second problem is that we have a completely decentralised scenario, meaning that we lack an entity that could run the mechanism. Instead, peers are both actors in the game and they need to self-organise the game and make sure that no player tries to circumvent the mechanism. We will look at these two problems now in more detail.

Mechanism Design and Computers

If we look at computers as actors with computer networks between them, game theory and mechanism design have to be adapted to the special conditions of that environment.

This area is called Algorithmic Mechanism Design (AMD) (Nisan and Ronen, 1999) or Computational Mechanism Design (CMD) (Parkes, 2001) in the literature.

Classic game theory makes a number of assumptions that do not hold when computers are the actors. It assumes that actors have the time and the computational abilities to consider all available information and make optimal decisions based on this information. This assumption obviously does not hold true when computers are the actors.

Thus, the actors in our case are *bounded rational*, they cannot always act fully rationally.

In addition, even if the algorithm manages to find the best move in a game, the network might not be able to execute it. There is no useable way to guarantee the bandwidth of a flow in today's Internet and so a strict tit-for-tat algorithm based on allocating bandwidth simply does not work. It is systematically impossible for peers to play the right moves because the nature of the Internet protocols will not let them.

Still, these problems could be partially overcome with a dominant strategy mechanism. Here, the peers do not need to consider possible strategies but it is in their best interest to follow the dominant one. There is still the "auctioneer" though, the central entity that runs the mechanism. Depending on the actual mechanism, its resource limitations will likely render the mechanism unusable with computers as actors.

Mechanism Design and Networks

Apart from computational bounds, communication is another limitation that we face. Computers are connected to each other with networks, and networks have delay, communication costs and limitations in bandwidth. So the free and unlimited communication that is assumed in traditional mechanism design does not hold true in computer networks. Yet this problem could be overcome in a direct-revelation mechanism, where peers only have to report their type to the mechanism.

The second and much more problematic point is that we are dealing with a completely decentralised system, where there is no entity that could perform the role of the mechanism ("auctioneer"). The peers (b̈idders)̈ themselves will have to perform this task. This means however that there is a conflict of interest on a peer between its own interests as a bidder and its function as part of the mechanism. Thus, the mechanism has to be resilient against attacks by the peers it is running on, so that the selfish peers cannot gain an advantage by modifying the mechanism in their interest.

Unfortunately, the revelation principle also is of little help in the distributed case, as direct-relevation mechanisms convert decentralised problems into centralised ones. This would require an auctioneer to handle the actual mechanism, which we do not have in a peer-to-peer network.

These problems were looked at by (Feigenbaum and Shenker, 2002) who called

this area Distributed Algorithmic Mechanism Design (DAMD), and by (Parkes, 2001) who calls it Distributed Mechanism Design (DMD). However, this is a very new research area, and there are by far more questions than answers. (Mahajan et al., 2004) found out that applying game theory in network settings does not necessarily lead to the desired results.

To the best of our knowledge, there is as of now no compelling example of a working distributed mechanism that fulfils our criteria.

2.5.4 Dead End?

The problems of mechanism design when applied in an environment of actors with bounded rationality and limited communication and the lack of results in this novel area effectively means that there is no way for us to construct a distributed strategy-proof mechanism that would solve our problem while fulfilling all of our requirements. The tool box that would assist with designing such a mechanism is still in early development itself, as the research area of distributed mechanism design is only in its infancy.

Others have made the same observation. According to Mahajan et al (Mahajan et al., 2004), game theory and mechanism design are not particularly helpful for designing systems. (Feigenbaum and Shenker, 2002) agree that ”because the Internet somewhat different to traditional game-theoretic contexts, the traditional solution concepts may not be sufficient.änd that there are no design concepts for games that do not have a dominant strategy solution. Furthermore the same authors stated earlier that there is no coherent framework for distributed algorithmic mechanism design, that many fundamental issues are unsolved and that most of these issues have never even been addressed. (Shenker and Feigenbaum, 2001)

This means that we will have to find an alternative approach to solve this problem. And while game theory and mechanism design do not help us to find a solution directly, they help us in structuring and understanding the problem, and we are able to build on top of this.

2.6 Problem Statement

We are looking for a lightweight and universal mechanism to incentivise file sharing in peer-to-peer networks.

We have specified a game that describes the problem of providing incentives for cooperation in peer-to-peer networks, where a peer-to-peer network consists of actors called peers and of a utility function that is based on the qualities of the services provided to the peer. Based on that we can describe the strategy space of peers and discuss equilibria of the system.

The challenge is to find an equilibrium that maximises the social choice function. Through Mechanism Design it is straightforward to design an algorithm that achieves this goal under the assumptions of unlimited computational time and unlimited and instant communication between the actors and the mechanism, but to the best of our knowledge there is no way to develop a distributed mechanism with these characteristics.

Therefore we need to apply an engineering approach to the problem. This means that we will need to build and test a system and evaluate it.

2.7 Related Work

In recent years there has been a large number of related work in the area of incentives mechanisms for peer-to-peer networks. Only a part of this work is carried out in academic circles, while a number of approaches were first implemented in a real world network before they found their way into an academic paper, if at all. In order to cover all the different approaches, we will have to look at both types of related work, academic papers and file sharing software.

Generally we distinguish three different major branches of incentives systems. The first branch are systems that are based on the reputation of peers. Often these reputations are exchanges among peers, which requires a trust infrastructure to be in place. The second branch are micro currency systems, where some form of internal currency, tokens or credits are used between peers to ”pay” for services. The last branch finally deals with systems where cooperation is required by peers for them to receive a service in return. The service quality they provide to others is immediately reflected back upon them and manifests in the service quality they receive. Finally we will look at other related work that does not fit into any of the aforementioned categories.

2.7.1 Early Peer-to-Peer Networks

For completeness, we will first mention some of the early peer-to-peer networks, none of which had an incentives mechanism to speak of. There were however some naive early attempts of encouraging users to share files.

The networks relied completely on the voluntary, altruistic cooperation of peers. Some early attempts on an incentives mechanism were inspired by dial-up bulletin board systems of the 1980ies and early 1990ies, before the Internet became prevalent. The idea is that there is a ratio of a peer's download in relation to the upload. We are not aware of any scientific work suggesting this method.

(Napster, 2010) was the first peer-to-peer network that became popular. The original Napster network started in autumn 1999 and was shut down for legal reasons in summer 2001. Its architecture was very simple. While the actual file download was decentralised, peers would find out about each other and the files they host through a centralised database. Each peer would connect to the Napster servers on startup and report which files it made available to others. Then it could query that database for particular file names and would get a list of hosts that supply the file, or contact a host directly and browse its files. There was absolutely no incentives mechanism in place, so peers could chose not to make any files available for download.

The original (Gnutella, 2010) software was different to Napster in that there was no central server and that it had a decentralised search functionality, where originally every single search request was forwarded to every other peer. The protocol and the prototype implementation were developed in early 2000. Over the years a great number of changes were made. The most important change being a new, more scalable search function. As with Napster, there is no incentives mechanism for cooperation in place at all.

The first client of the (EDonkey, 2010) file sharing network used a simple client side ratio scheme. The client would cap the download speed based on the maximum upload speed set. If the maximum upload speed was below 10 kB per second, the download would be limited to about 3 times the upload speed. Since this functionality was in the client software, patched versions of the client that did not have this limitation were made available by third parties quickly after the release of a new version of the software.

(Kazaa, 2010) uses a so called ”participation level”. The client monitors the ratio of upload and download bandwidth and limits the depth of the search function depending on the participation level. The higher the level, the deeper the search. The participation level is stored on the local client computer and can be easily tampered with. New peers start out with a medium participation level, so all a peer with a low level has to do to increase its level is to create a new identity.

2.7.2 Trust/Reputation Systems

In trust/reputation systems, the quality that a service obtains depends on its standing with the other peers in the network. This approach is often coupled with a trust infrastructure, which is typically reputation based itself. That infrastructure is used to distribute reputation across peers, so that peers can draw conclusions from other peers' experiences before they might have had a chance to deal with a particular other peer themselves.

Traditionally the problem of trust is discussed with a centralised environment in mind and is not very concerned with issues like scalability. Thus, it was not possible to simply use existing trust mechanism, but new ones had to be invented that are designed specifically for the massively distributed networks.

There are a few general properties of reputation based approaches. The first property is that some form of identity across sessions is required for the peers. Without such a static identity, peers cannot profit from the good reputation they gained in one session in the following session. Each peer will also need to maintain some form of a reputation database, which either requires a distributed implementation or which creates scalability issues in large networks with many peers. The third and final property is that it is generally beneficial for peers to exchange reputation ratings with each other. No peer exchanges services with all others peers, so it makes sense to exchange this information amongst peers. However, a trust infrastructure is required to do this, so that peers can have an informed opinion on the accuracy of reputation information gather from other peers.

Maintaining a reputation database, and possibly even exchanging reputation information between peers, together with the identity mechanism and the trust mechanism required for reputation database exchange are too complicated to be worthwhile and

create too much overhead for the relatively simple problem at hand.

(eMule, 2010) is a client for the eDonkey file sharing network network. It has a very simple credit based scheme. Each peer keeps track of what other peers upload to it. Other peers then get advanced in the download queue when they have a good upload history. There is no way for a peer to enforce this or even to find out about the actual advancement in the queue, and not all eDonkey network clients support this scheme anyway. There is also no mechanism in place that would force an eMule client to honour the accumulated reputation of another peer.

We will now discuss some of the research papers in this area. We note that to the best of our knowledge none of the academic approaches were ever used in a real peer-to-peer network. There is also little data of experimental results and simulations available, making it very difficult to compare reputation approaches with the ones we will present further on.

Kamvar et al introduce a participation metric for peers they call *EigenTrust*. In (Kamvar et al., 2003b) the goal is to discourage the upload of ”inauthentic filesb̈y means of a unique global trust value. This way the overall performance of the system is increased. In (Kamvar et al., 2003a) that this reputation vector can be used to determine the relative cooperation level of a peer.

Using bartering instead of a currency, coupled with a trust mechanism, has been suggested by (Chun et al., 2003). The trust mechanism relies on secure communication, which leads to peer IDs. Additionally, tickets which represent resources need to be saved, along with a tree of all issued tickets on each ticket issuer peer.

(Jurca and Faltings, 2005) have a service oriented computing perspective. They introduce a reputation mechanism that works on incentive-compatible service level agreements instead of social exclusion in the case of non-cooperation. It is assumed that the peers tell the truth and the authors can show that there is a Nash equilibrium for that particular case. The reputation of the peers is expected to be kept in a globally accessible database.

(Ngan et al., 2004) suggest a mechanism for a storage sharing system based on auditing. Each peer maintains a digitally signed usage record that others peers can check. A strong peer identity mechanism is assumed, as well as a public key infrastructure.

(Gupta and Somani, 2004) try to combine the advantages of reputation systems

and micro-economy systems. In their model, the reputation of a peer is a measure of its wealth. The mechanism ensures that serving a peer with a higher reputation will result in a higher increase of reputation of the providing peer. To achieve this, peers form trusted communities. This again requires an strong identity of peers as well as a reputation database on the peers. There is a lot of communication between peers of a trust group.

(Sun and Garcia-Molina, 2004) suggest a Selfish Link-based Incentive Mechanism (SLIC) for overlay peer networks. Each peer maintains a connection to only a few neighbours and assigns them a rating based on the quality of the queries to and from that neighbour. This way, a peer does not have to consider the whole network but only its neighbours. In turn, the peer has an interest in having a good rating with its neighbours because only highly rated neighbours get all the valuable queries. This appears to be a suitable incentives model for the query overlay network but does not work with the truly peer-to-peer service provisioning that follows a successful query, since this does not involve forwarding between a number of peers but simply a service between two peers.

2.7.3 Micro Currency Systems

A number of approaches to solve the problem of cooperation in peer-to-peer networks use a micro-economy between the peers. Peers would pay each other for services in a virtual currency, assuring that only peers that provided services in the past could themselves consume services. The main problem with this approach is that it typically requires a centralised agency that issues currency tokens. For the peers, currency tokens are expensive to save, exchange and secure. Besides, it requires effort to keep the currency stable in value. A credit system on the other hand requires strong identities for all peers, as well as an enforcement scheme for peers that are unwilling to pay.

(Odlyzko, 2000) provides an excellent and very extensive argument against micro-payment schemes. In networks where communication is relatively cheap, it often is the case that the cost of making a decision on whether to use a service is higher than the micro-payment associated with that service. This is especially true when the user has to be consulted for this decision (”mental transaction costs”), as opposed to having a set policy that the computer simply follows.

The situation is a bit different in mobile ad-hoc networks as those have an actual resource shortage, i.e. every resource in the end translates to battery power, which is scarce. Thus, having a ”strong” incentives scheme is more easily worth it and makes more sense than in peer-to-peer networks.

Mojo Nation is of particular interest, as it was the first peer-to-peer network that was based on a micro-economy. Peers received ’mojos’ from other peers for storing their data and spent them by storing data on other peers. A central authority would issue mojos, namely the Mojo Nation company itself. New peers would get a small number of mojos to get started, making it relatively easy to cheat the system. A lot of technical problems were encountered (Wilcox-O’Hearn, 2002) and eventually the company went out of money in 2001 and the network was shut down.

(Golle et al., 2001) introduce a micro-payment based approach in one of the first papers on the subject of incentives in file sharing networks. The approach assumes the existence of a central server that tracks all uploads and downloads of a peer. Two different metrics for peer behaviour are suggested: uploads per time and number of files offered.

Ppay is a mostly decentralised micro-payment system developed by Yang et al (Yang and Garcia-Molina, 2003). Most of the currency related functions are executed by the peers, but centralised brokers are still required to open and close accounts and for arbitration.

(Ham and Agha, 2005) suggest a credit system combined with an auditing system to prevent peers from altering the credit values. There are however no useful performance figures for the system to judge its performance.

In another paper, (Yang et al., 2005) look at the query function of a Gnutella like network. Since only queries allow a responding peer to provide services, they are a valuable resource and there is no reason for a peer to forward them to potentially competing neighbouring peers. Thus, the peer sells a right-to-respond (RTR) to queries to other peers by forwarding them in return for a reward. Like the similar reputation based approach that is targeting the search overlay network, this approach also does not work with arbitrary services or even with file sharing.

2.7.4 Tit-for-tat based Systems

(Sanghavi and Hajek, 2005) propose an extended Vickrey-Clarke-Groves mechanism that can deal with severely limited communication between the players and the centralised provisioning authority (auctioneer). Unfortunately their approach cannot deal with dynamic situations, where actors come and go constantly.

(Buragohain et al., 2003) contains a game theoretic model of the problem of cooperation in peer-to-peer networks that is similar to our approach. They introduce a global contribution of each peer, and a benefit matrix which contains the value that each peer's contribution has to each other peer. The probability that a peer replies to a request is directly related to the contribution value in the benefit matrix (modulo a probability function). The authors then derive the Nash equilibrium for their simple scenario and can show that their system will eventually operate at it. The main limitation of this paper is the benefit matrix, where it is not clear at all how a peer would derive the contribution scalar of other peers. Since this contribution value would have to include the contributions to others, the peer that makes the decision based on it would not know about it. It would require a costly reputation infrastructure to maintain an accurate benefit matrix.

We will revisit related work on tit-for-tat based systems in Section 3.4, where we will discuss the related work that is closest to our own.

2.8 Summary

In this chapter we have first described the constraints and requirements for a solution to our problem. Then we specified the basic model and the assumptions we make. We then provided a game theoretic analysis of the model where we devised a formal model of the game on service quality that takes place between peers. We discussed possible strategies of peers in that game and we attempted to solve the problem by using mechanism design.

After coming to the conclusion that mechanism design would not be able to help in designing a solution due to the constraints of the model we think that instead, an engineering focused approach is preferable.

The remainder of this chapter then reviewed the related work in the areas of early peer-to-peer networks, trust/reputation systems and tit-for-tat based systems.

Chapter 3

Principles

We are looking for an incentives mechanism for file sharing that is both lightweight and universal. The study of related work suggests that of all the lightweight mechanisms, those using tit-for-tat like strategies work best by far. Especially with BitTorrent being as successful as it is, the tit-for-tat principle has proven itself in real world applications. Although BitTorrent is lightweight, it is not universal. It only works well with large and popular files. It also requires seeds (i.e., peers that provide the file regardless). Our approach is to improve BitTorrent by *generalising* it. In order to make BitTorrent universal, it needs to work for various file sizes and popularities and it should not require peers to altruistically serve other peers, as is the case with seeders in BitTorrent.

We can interpret the two reciprocal downloads taking place in BitTorrent between two peers in a tit-for-tat supporting file sharing network as a two leg directional *ring*. The basic idea of our approach is to extend that ring if possible by inserting more peers and thus legs into it, and by allowing any resources on the peers to be exchanged for one another (as opposed to exchanging pieces of the same file in BitTorrent). Intuitively it is then more likely for an arbitrary download to be part of at least one tit-for-tat ring, which in turn makes it possible to apply a tit-for-tat like mechanism to these downloads, and thus all members of the ring will be able to get higher quality services by providing higher service qualities themselves.

This also makes the tit-for-tat approach much more universally applicable because rings make it possible for peers that do not have popular resources to participate in file sharing networks. Additionally, seeding peers for a particular resource can get remunerated for their efforts. The file sharing network does not need to rely on initial seeds to spread a file. However, for this to work we need to make sure that most or all services

are part of a ring at some point in time since tit-for-tat only works within rings.

We also need to find out about the existence of those rings that have formed. In BitTorrent, the existence of rings of length two is immediately obvious to a peer by looking at its neighbours: a ring is found if a providing neighbour is at the same time a requesting neighbour. This is not the case for longer rings because different peers are on the providing and requesting legs of the ring, and so a ring detection algorithm is required.

In fact, there are two separate problems in this. The first problem is to find the best tit-for-tat like algorithm for peer-to-peer networks, and the second problem is how to make sure that all or most services and peers are within one or more rings, allowing them to get a better quality on the services they use by providing services to others.

There has been some work on the first problem in the area of bulk content distribution algorithms, and therefore we will concentrate on the second problem in this chapter and look at a number of mechanisms that might be suitable to solve it. Rings need to be created by peers by carefully choosing the peer that will provide a requested service, and depending on the creation mechanism peers might require a way to discover existing rings. Additionally we have to consider the highly dynamic properties of the services that make up rings. Even peers themselves come and go constantly.

In the remainder of this chapter we will first introduce the peer-to-peer principle in more detail, and explain the BitTorrent algorithm. Then we look at how the tit-for-tat concept based on BitTorrent can work for rings and explain why the service density plays a role. Finally we will discuss ring detection mechanisms and investigate how peers can manage rings.

3.1 The Incentives Algorithm

The purpose of the incentives algorithm is to maximize the received service qualities by periodically setting the outgoing service qualities. In the special case of a file sharing network, the outgoing service qualities are defined by the choice of peers this peer uploads to and the upload speeds that are provided for such uploads.

Since we assume that peers have no long-term identity, a peer cannot keep knowledge of past behaviour of another peer. Therefore the reciprocation for a received service must be immediate or near immediate by supplying a service in return. Ideally

there is a correlation between the received services and the provided services so that providing better service qualities or more services will lead to an improved received quality or more received services. This concept is known as *tit-for-tat*, since a peer can expect to receive a quality of service that is correlated to the service quality that it provides to other peers.

3.1.1 Tit-for-Tat Principle

As mentioned earlier, tit-for-tat denotes a strategy that a peer can take over the course of a number of games. Let us assume that we have two peers that are providing a service to each other. Using a tit-for-tat like strategy, a peer responds to the other's strategy by behaving similarly in the following iteration of the repeated game. Cooperation is answered with cooperation, defection with defection. In the first round, the peer cooperates. See Figure 3.1 for an automaton depicting tit-for-tat (Binmore, 1992).

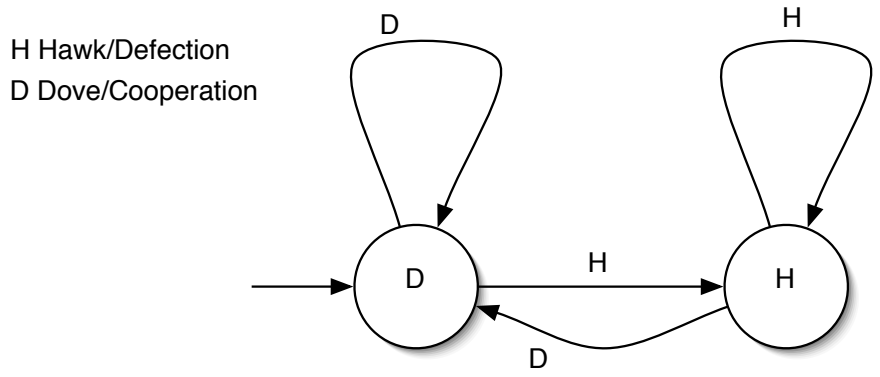

Figure 3.1: Tit-for-tat Automaton

This means that if the other peer is using a tit-for-tat like strategy, the only sure way for a peer to receive a good service quality is to provide a good service quality in return. But this can only work if those two peers actually do provide services to each other. In a file sharing network for example, if Peer **A** simply downloads a file from Peer **B** without in return uploading another file to Peer **B**, then there is no way for **A** to influence the speed of the download it gets. In fact, if Peer **B** acted rationally then it should not upload anything to Peer **A**.

Making this algorithm work in practice also requires an *optimistic* component: In order to get new tit-for-tat relationships going, peers need to start providing services to a peer and then to observe the reaction. Hopefully, providing the service will trigger

the provision of a service in return so that a tit-for-tat relationship can be established. Peers generally need to err on the side of generosity or else the whole system may end up in a deadlock situation because of the equilibrium of no peer providing anything to any other peer.

3.1.2 The BitTorrent Mechanism

Applying this principle to file sharing is very straightforward since service qualities equate to upload speeds. However, what services are actually being exchanged? Given large files it makes sense to split these up into smaller pieces and then treat each piece as an individual service: Each individual service takes less time to complete and so is more likely to complete. Since each peer downloading a file needs all of its pieces, there are also more peers to exchange services with, and more importantly those peers can be incentivised to provide missing pieces as well—contrary to seeds.

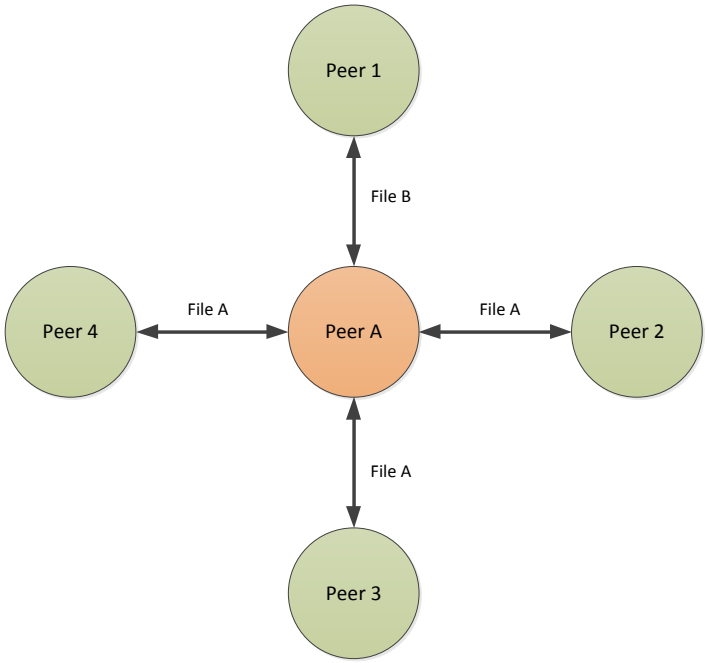

Figure 3.2: BitTorrent Download Relationships

This is in fact the BitTorrent model: Breaking files into pieces and then having peers downloading the pieces from each other. Figure 3.2 shows a peer that is uploading to and downloading from a number of other peers. It is exchanging parts of file **A** with three peers and parts of file **B** with another peer.

BitTorrent Architecture

Figure 3.3 shows the classic BitTorrent architecture, where there is one central node per file, called tracker, to which each peer connects at startup and which it then polls in regular intervals. The tracker tells it about other peers that are interested in the same file, i.e. other peers that have registered with it before. This is essentially the implementation of a search function that returns a list of peers that are interested in a given file. Peers then connect to those other peers that are also interested in the file in order to exchange pieces of the file with them.

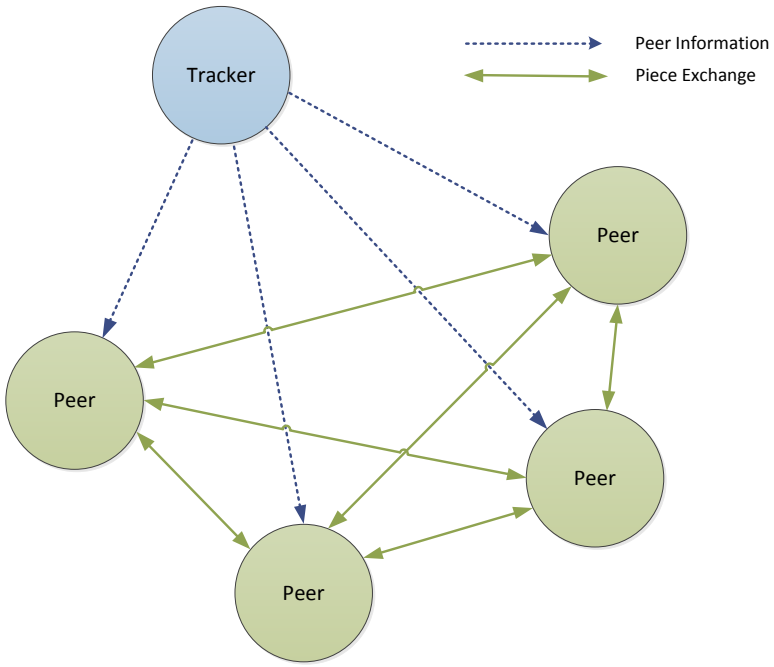

Figure 3.3: BitTorrent Architecture

Trackers are typically not participating peers but usually specialised nodes that often co-ordinate multiple files separately at the same time. The IP addresses of the trackers as well as checksums for the file and individual pieces of the file are contained in a metadata file that the peer has to get hold of before it can start the download, often through a web site. This file is called a torrent file.

In newer BitTorrent clients the file checksum is used for an alternative search method: It serves as an identifier in a distributed hash table among the peers that maps the checksum to the list of peers interest in the corresponding file. Typically, peers use both the tracker and the distributed hash table simultaneously in an effort to maximise

the number of peers known to be interested in the file.

Once a peer has received a list of other peers interested in a file, it starts the uploading and downloading of pieces of that file. A peer typically uploads to 4 or 5 other peers per file while maintaining an idle connection to many more peers that is only used to exchange information about newly acquired and thus offered pieces.

The individual speed of each upload is not controlled by the peer but solely determined by the network conditions. All the peer controls is the maximum bandwidths for all uploads and downloads combined. It does not try to control the bandwidth distribution of the uploading network flows but instead lets the network find the equilibrium.

BitTorrent Incentives Mechanism

In regular intervals of 10 seconds the peer ”chokes” the upload to the worst performing peer, i.e. the peer out of the set of peers it uploads to that it gets the slowest download from, and ”unchokes” another peer's upload. This peer is randomly chosen from the connected but idle neighbouring peers. This is called the ”optimistic unchoke”.

For the next three intervals this peer will not get choked, to give it the opportunity to recuperate the upload with a download, and for TCP to settle. After this time it joins the pool of active neighbouring peers that can be choked at any time if the download is not among the four fastest downloads.

The result of this is that eventually peers that offer the fastest upload will get the fastest download, while peers that do not upload anything will get a comparatively slow download. This strongly resembles the tit-for-tat mechanism, where cooperation is answered with cooperation and defection with defection. Peers need to provide upload bandwidth to assure a fast download.

The algorithm is not strictly tit-for-tat as (Jun and Ahamad, 2005) have documented, but it is a very workable solution regarding the fact that service qualities are not fully under the control of the providing peer—bottlenecks might exist somewhere along the path to the recipient and TCP's congestion control will slow down the flow automatically.

BitTorrent works very well in the scenario that it is designed for: a distributed version of FTP that allows interested parties to publish and spread large and popular files quickly with minimal cost to the party that is publishing it.

3.1.3 Generalising BitTorrent

Splitting files into pieces works very well with large and popular files because there are enough pieces and enough peers that are downloading at a given time to make it likely that a tit-for-tat service exchange can be initiated. It does not work however for small and/or rare files because there are not many reasonably sized pieces in small files. Because of that and because of the low popularity there are fewer peers who download the file at any given moment. Therefore, finding a tit-for-tat service with another peer exchange is unlikely. BitTorrent also does not solve the problem of providing an incentive for seeding files and relies on peers seeding a file because of an external incentive (for example, a company having an business interest in seeding the latest version of their product trial software).

Small files cannot be split into pieces that are large enough to efficiently work with a BitTorrent-like incentives mechanism. But when a file has only one piece then an incentives mechanism that relies on exchanging pieces of the same file cannot work. This is because each peer needs to have one piece that it can offer the other peer, but if there is only one piece in total then that is impossible.

Rare files might have enough pieces for the BitTorrent incentives mechanism to work in principle. The problem with rare file is however that they are not downloaded often enough to make it likely that there are two peers that wish to download any particular rare file at the same time. Therefore, in many cases there is no other peer that can be incentivised by exchanging pieces of a file.

In both cases BitTorrent does not work because there is nothing that a peer that is downloading a file can offer to the peers that already have the file. Effectively all peers that potentially have pieces to offer are seeds for the file and so they have no reason to offer it. Therefore we might be able to solve the problem of sharing rare and/or small files by creating more opportunities of exchanging pieces of files.

We can imagine two ways of achieving this: First, peers could form tit-for-tat relationships by uploading pieces of different files to each other. Second, if peers are not interested in each other's files but a third peer or even more peers exist with which a ring of uploads can be formed.

First Idea: Pieces across Files

Pieces of different files could be exchanged instead of pieces of the same file. Each peer offers all pieces of all available files in addition to already downloaded pieces of a currently requested file. Two small one-piece files could be exchanged. If a peer is the only one downloading a large but rare file then it might recuperate by uploading pieces of other files that it already has in return.

Figure 3.4 illustrates this idea. Instead of the peer uploading and downloading pieces of the same file to all other peers it is now able to exchange pieces of different files.

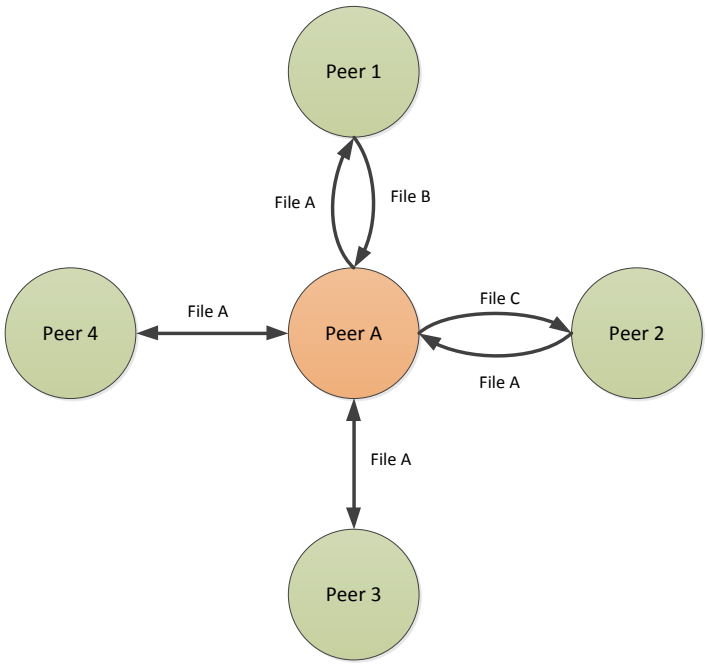

Figure 3.4: Download of Pieces across Files

In addition this would solve the problem of the initial seed. In this approach even peers that are only interested in a large and popular file could benefit from it: If they have something to offer then they can incentivise other peers to upload the initial piece of the file. They would not have to rely on another peer's optimistic unchoking combined with a fast enough upload speed to complete the first piece before the transfer gets choked again.

This idea increases the number of potential tit-for-tat relationships between peers, but on its own it is not enough as we shall see later in Section 5.6.3. Therefore we

propose a second mechanism to increase the number of potential tit-for-tat relationships further: incentives rings.

Second Idea: Incentives Rings

If a peer has no file to offer that any other peer that has a desired file is interested in then there may be a third peer that is interested in something the peer has to offer and at the same time can provide something that the second peer is interested in. By introducing three or potentially even more peers into the tit-for-tat relationship we can form an incentives ring. Tit-for-tat relationships in BitTorrent can be said to form an incentives ring of length two and we now suggest to extend that ring. This increases the probability of small and/or rare files being part of a tit-for-tat relationship. Whenever files are too rare or too small or whenever a peer wants to download the first piece of a file, so whenever there are no rings of length two, peers can look for longer rings.

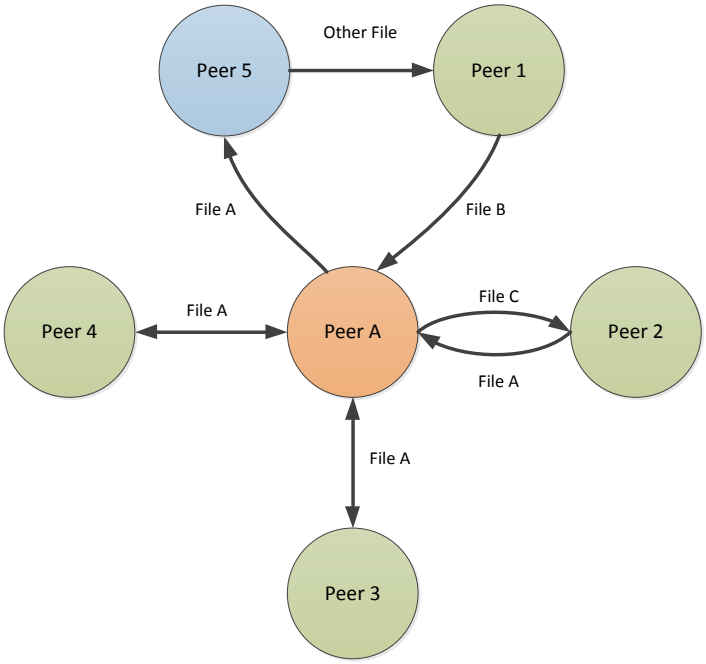

Figure 3.5: Incentives Rings

Figure 3.5 shows the same scenario as before, but this time with one ring of length three. In this case Peer **1** is not interested in any of the files offered by Peer **A** and so a Peer **5** is required for an incentives ring. Peer **A** is not concerned with the other file that is exchanged between Peer **5** and Peer **1**.

When using incentives rings nothing changes compared to BitTorrent with regards to the tit-for-tat game being played. The only difference is that in BitTorrent the providing and receiving peer are the same while with a ring they are not.

3.1.4 The Service Overlay Graph

When we talk about rings then we refer to cycles in the directed graph that is created by the peers as nodes and potential uploads, or more generally potentially provided services, as edges. We call this graph the *service overlay graph*: it is a graph of all the peers of the network that is on top of the actual network topology.

A peer can have a number of incoming and outgoing connections, and each may or may not be part of a ring. To visualise this, Figure 3.6 shows a peer and its immediate service overlay neighbourhood, in this case comprising of three rings of size two, three and four, and two additional peers that are not part of a ring.

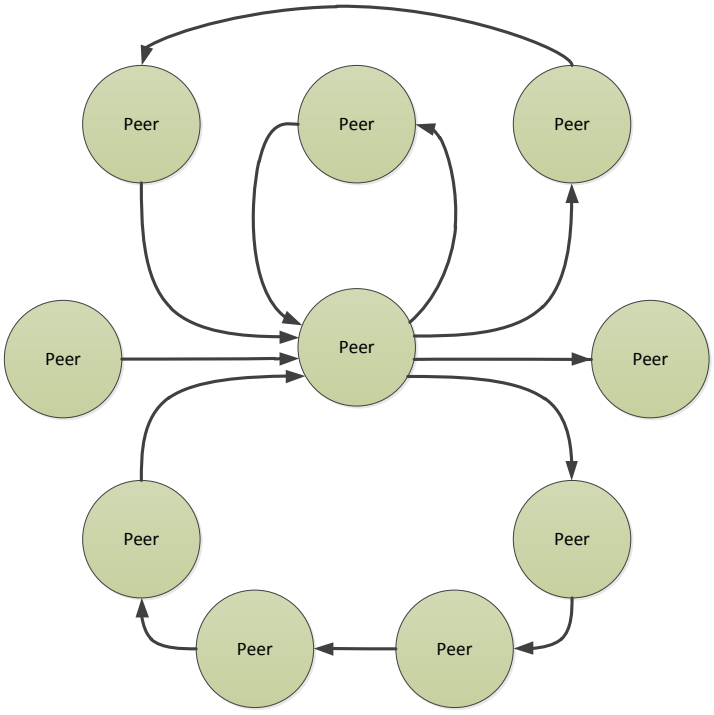

Figure 3.6: Peer in Service Overlay Graph

It is important to note that the number of potential rings, that is the number of cycles in the service overlay graph, correlates with the density of that graph. By density we mean the number of edges in the graph in relation to the number of *potential* edges, i.e. in relation to the number of edges in a full mesh graph. The more potential services

between peers, the more edges in the service overlay graph and the more incentives rings there are.

The denser the service overlay graph, the greater is the chance that two peers are interested in each other's services. Therefore it is important for the peers to try to maximise the density of the service overlay. This maximum number of edges in a directed graph is $e_{max}(n) = n(n-1)$, with n being the number of nodes in the graph.

With e being the number of edges and n being the number of nodes in the graph, the density d of a directed graph is then defined as

$$d(e,n) = \frac{e}{e_{max}(n)} = \frac{e}{n(n-1)} \tag{3.1}$$

Since the density is directly related to the number of services provided, the obvious way to increase the density is to simply increase the number of offered services. This is what BitTorrent does by first having separate and unconnected graphs for different files and then splitting files into pieces. The result is that almost any two nodes can form a ring of length two. Effectively BitTorrent uses one almost fully meshed service overlay graph per file and limits the ring length to two.

For small and rare files this does not work because there would be no other peers in a file's overlay. That is why we effectively combine the service overlay graphs of all files into one.

The only way for a peer to increase the density of the overlay graph is by increasing the number of files it either offers or requests at any one time. It is expected that the density of the graph will not be uniform because file interest is not uniform. There will be local clusters of nodes that are highly interested in each other's files, for example the users of FreeBSD who are all interested in downloading some of the latest FreeBSD packages.

3.1.5 Incentives Rings

With clusters in the service overlay graph, the question becomes how to make use of them in order to apply the tit-for-tat principle for a larger number of services. The main challenge is to make it possible to apply the tit-for-tat principle to services that are not part of a reciprocal exchange of services between two peers.

Our proposed solution is to extend the principle by allowing more than two peers

in a tit-for-tat relationship that form a ring of unidirectional services (Ackemann et al., 2002). Having a ring makes it easier for peers to join a tit-for-tat relationship, since they do not need to find another peer that is interested in their services directly. For tit-for-tat to work, it is enough for peers to establish a ring with 3, 4 or more legs that includes them.

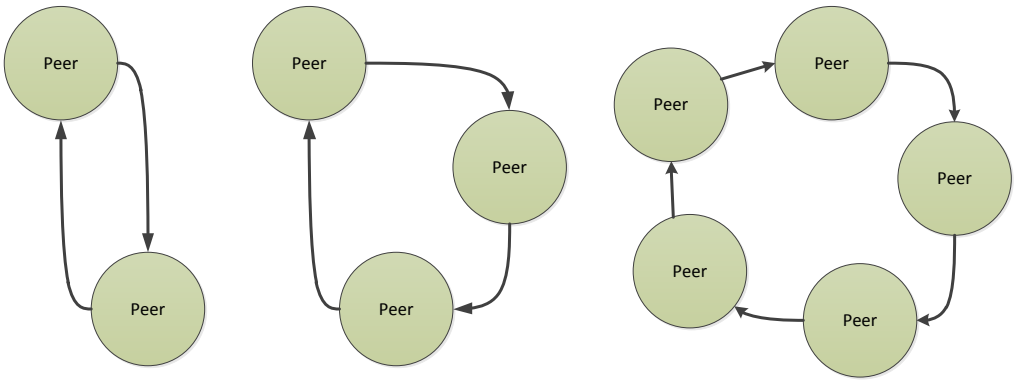

Figure 3.7: Rings of Length two, three and five

A tit-for-tat relationship between two peers manifests itself in two opposing edges between two nodes in the service overlay graph. This relationship can be interpreted as a unidirectional ring of length two. We can now extend this ring by allowing one additional peer into it (see Figure 3.7). Instead of a peer playing tit-for-tat with another peer it uploads to and downloads from, it uploads to one peer which in turn uploads to another peer which then uploads to this peer.

By using rings we can greatly increase the probability that a peer's outgoing service is linked in some way to one or more incoming services, thus allowing the peer to actively improve its *received* service quality by means of that control loop. This is the incentive we need for the network as a whole to work. The number of rings in a graph is related to the density of the graph however. The denser the graph, the more rings will appear.

For a peer, a ring is defined by a provider peer and a receiver peer and by the files of which pieces can be acquired through this ring. In order to be able to tell cause and effect, each neighbour can only be in one active ring at a time.

Figure 3.8 shows two rings of length three. One leg of each ring—the link between Peer **2** and Peer **4**—is shared with the other ring. If both rings were active then Peer **4**

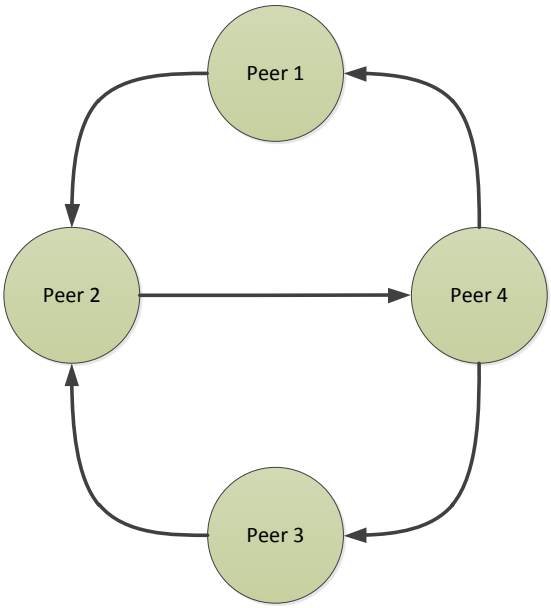

Figure 3.8: Rings - Cause And Effect

would not be able to tell which of the two services that it provides, one to Peer **1** and one to Peer **3** controls the service quality of the received service from Peer **2**. **4** would not be able to tell cause and effect and so would not be able to apply the incentives algorithm efficiently. Therefore Peer **4** will only accept one of the two rings to become active.

Peer **2** in this scenario knows by virtue of having two incoming services for one outgoing service that somewhere else in those rings there must be a peer that is in the situation of Peer **4**. So it knows that that peer will not accept a request to activate the ring and therefore it knows that it would be a wasted effort to try. Ergo, peers have an interest to have each neighbour only associated with one active ring.

Dynamic Aspects of Rings

There are of course challenges in detecting and managing those rings. In this section we will identify those challenges. We will talk about ring detection in much more detail in Section 3.2 and about managing rings in Section 3.3.

The first challenge is that of ring detection. Before rings can be used they need to be actually found. This is a problem because peers only know about their direct neighbours, which only allows for trivial rings of length 2 to be found. Therefore a

distributed algorithm is required that allows peers to find rings of length 3 or more. Three basic approaches come to mind:

The classic approach would be to introduce a server that collects request information from the peers and that then finds rings using a graph cycle detection algorithm. If there was a centralised search function in the network then that could be used as it would have all the necessary information already. However, a peer-to-peer network typically has no servers and it would be difficult to see how one could be incentivised. Additionally, existing BitTorrent trackers and distributed hash tables (DHTs) could not be used any more and would have to be replaced.

Another approach would be not to create rings but to only detect them. Each peer uploads to a set of randomly chosen neighbours and looks for a correlation between any of the outgoing upload speeds over time and the received download speeds from other peers. If there is a correlation then a ring has likely been found. While this approach might work in highly homogenous networks with few different files and thus a high probability of an interest between two randomly chosen peers, it will not be efficient in more general networks because of the signal-to-noise ratio. All peers of a ring would need to upload to just the right neighbours for long enough for the detection algorithm to identify the ring, all the while the downloads are proceeding with no incentives mechanism controlling them.

Finally the most promising approach is to actively look for rings. For this peers need to exchange tokens with their neighbours containing details about their other neighbours, thereby extending the knowledge each peer has about the overlay graph. Eventually a peer will have enough knowledge to identify a ring, and then it can proceed to inform the other peers in the ring.

The other challenge besides ring detection is that of ring management. Once a ring has been identified, peers need to find group consensus on whether and when to start it. And once the ring is started, peers need to keep the tit-for-tat scheme running, regularly checking the performance, and throttling it when appropriate. When one segment of the ring breaks then all peers of the ring need to realise it and shut down the remaining segments as well, or modify the ring if possible.

3.2 Ring Detection

Now that we have a good idea of what the peers and the overlay network look like once they are in place, we have to look at how to actually find the rings in the overlay network.

First it should be noted that a ring detection algorithm is only required for rings of length greater than two. Peers know about rings of length two implicitly because they registered their interest in each other's files.

Obviously peers could apply a detection algorithm for cycles in graphs if they had complete, accurate and up-to-date information about the whole network, but it is expensive in terms of network bandwidth and CPU usage to distribute this information across the network. Additionally, collecting information about the complete service overlay and processing that information does not scale with the number of peers.

Even if they would get information about peers from third peers, they would not be able to trust that information, as those other peers have their own agenda and motivation. They may be interested in providing others with false information about the network in order to influence their strategic decisions in such a way as to gain an advantage. The only way to avoid this would be to increase redundancy and get the same information from different peers, which would yet again multiply the information transmitted between peers.

3.2.1 Ring Brokers and Ring Detection Through Learning

BitTorrent has tracker nodes, at least one per file, that broker information among peers about who is interested in that file. Trackers can deal with multiple files at once. Such trackers would already have all the information necessary to construct an overlay graph based on those files, and they could find some of rings. It is not guaranteed however that one tracker serves more than one torrent, and in recent times DHTs are replacing trackers as well. One of the peers could take over the role as a ring information broker in exchange for downloads by peers but it is hard to see how this would work and how the induced complexity would be worthwhile. Therefore this does not seem like a promising route to take.

Another possibility that we considered was for peers to learn about rings by computing correlation coefficients between provided are received downloads. Peers would

need to regularly collect correlation information for this algorithm. Each peer would remember the incoming and outgoing bandwidth of all flows for the last few intervals and then compute the correlation coefficient for each permutation of incoming and outgoing flows, resulting in a correlation matrix.

The peer then takes this correlation matrix and applies a ring detection algorithm that tells it how to modify the quality of the provided services in order to optimise the quality of the received services. The peer will identify which outgoing services influence incoming services and increase their quality at the expense of other outgoing services which do not seem to have a strong correlation with any of the incoming services. Additionally, peers will want to start providing new services in order to be able to discover even stronger correlations between a newly started service and one or more incoming services.

This approach is elegant in that it does not require any additional information to be passed around, but it does require a relatively dense service overlay graph so that there is a sufficiently high probability of rings to emerge by chance. Initial simulation tests that were conducted by us indicated that the signal to noise ratio is simply too high to actually detect rings using this way. Also, in order to be able to detect a signal in the correlation coefficient and thereby to detect a ring, peers need to be able to quickly and precisely adapt the bandwidth of services that they provide: Firstly to react to changes in the incoming bandwidth for already identified rings and secondly to actively probe suspected rings by modifying the upload bandwidth and monitor for changes of the download bandwidth of the flows associated with the ring. However, this is difficult to do consistently in the Internet as the peers have no control over the bandwidth of the flow beyond their own uplink.

It is very hard to see how in the light of all this uncertainty regarding rings, combined with the dynamic nature of the overlay graphs where rings come and go constantly, any kind of incentives mechanism could be successfully employed.

3.2.2 Random Walker Tokens

A third way of ring detection is to use an active, distributed algorithm. This is the approach that we decided to focus on in the remainder of this works.

Peers create and send out tokens to their uplink and downlink neighbours, they

forward received tokens and they detect rings by using the path information of received tokens in a way similar to link-state routing protocols. Rings can be detected quickly and reliably this way without having to interpret service qualities. This approach increases the network bandwidth usage however because of the number of tokens being sent.

A random walker algorithm seems best suited for this case: Instead of forwarding tokens to all neighbours and subsequently flooding the network, a token should be sent and forwarded to only one or at most a few randomly selected neighbours. This rate as well as the token creation frequency can always be adapted if not enough rings are found. When too many tokens arrive then peers can also simply drop them.

A generally similar approach has been taken by (Anagnostakis and Greenwald, 2004), however they send whole sub-trees instead of path tokens.

Peers construct rings by maintaining a list of paths to nearby peers using the path information from received tokens. These tokens are used to create two path tables, one for each direction. These tables contain the known peers and the known paths to them. Whenever a new token arrives, all peers in the path that the token took are checked in the table of the other link—if the token arrived through an upload neighbour then the download path table is checked and vice versa—and when a match is found then the peer has found a new ring.

It then informs the other peers of the ring by sending a ring found token message to the upload neighbour. Received ring-found token messages need to be forwarded by peers of course, so that eventually the token will get back to the originating peer. When the originating peer sees the message again then it can conclude that all other members of the ring still exist and still consider their part of the ring to exist and are willing to use the ring. For all peers to be certain about the intent of the other peers of the ring the message needs to be passed around the ring a second time. Sending and forwarding ring search tokens and ring found tokens is in a peer's interest because other peers will only cooperate in identified rings.

Tokens consist of an ID, a TTL field, and of the path that they have travelled so far. The TTL field contains the number of remaining hops to other peers before the potential path gets too long and the token gets deleted.

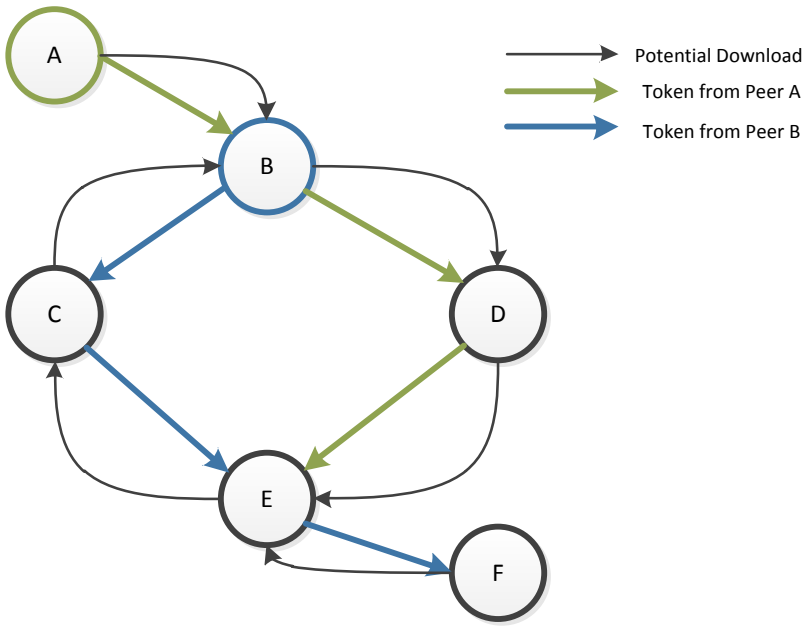

Figure 3.9: Peer with Token System

Figure 3.9 illustrates the basic idea. There are two tokens shown in the service overlay graph, a green one that travels on downlinks and a blue one that travels on uplinks. The green one originated on Peer **A** while the blue one originated on Peer **B**. After three hops the green token is at Peer **E** while the blue token arrived at Peer **F**.

Now consider Peer **E**. When the blue token passed through, it contained the path [**B**, **C**]. From this information the peer learns that there is an uplink path to Peer **C** via Peer **B**.

When the green token arrives from a downlink neighbour, it contains the path [**A**, **B**, **D**]. Peer **E** now checks for each of the path elements whether they are in the uplink path table. Peer **A** is not and neither is Peer **D**, but there is a known uplink path to Peer **B**. Since Peer **E** now has both an uplink path as well as a downlink path to Peer **B**, it has discovered a ring.

Each peer needs to generate new tokens and needs to process and forward received tokens. New tokens are periodically generated and sent out to one or more random neighbours with a TTL of half the desired maximum ring length. Received tokens are processed and then they are forwarded (and possibly duplicated) if desired, if the TTL is still non-zero. Processing tokens consists of using their path information and the path

table to find rings and of appending this peer's ID to the path and reducing the TTL field before forwarding the token.

In Section 4.11.2 we will describe a possible implementation of this algorithm in detail.

3.3 Ring Management

This section discusses the handling of already detected rings by peers. This includes the group consensus algorithm on starting and stopping them, as well as the incentives algorithm.

3.3.1 Ring Life Cycle

When we look at the emerging rings as entities then we can investigate their life cycle. Figure 3.10 shows the progression of states of rings. We consider a ring to have reached a new state when all member peers have processed the transition.

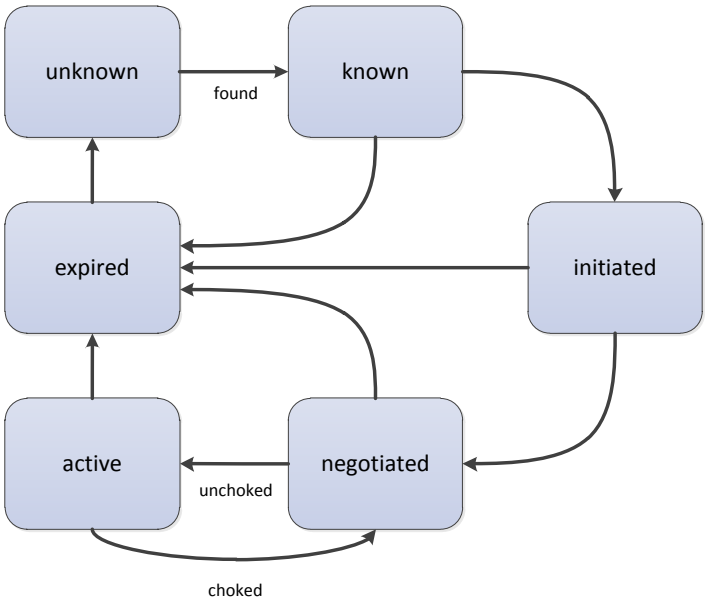

Figure 3.10: Ring Life Cycle

Initially a ring begins to exist but is unknown to the peers. Eventually, one peer detects a potential ring. This peer now needs to inform the other peers about the ring. Once all peers that are part of the ring know about it and also know that all other peers know about it, the ring enters the known state. From any point after a ring is known, it

might get expired if one leg disappears. It might take a while until all peers know that a ring that is not active has expired as they would need to try to initiate or unchoke it.

At some point all peers might come to the consensus that they are interested in actually using that potential ring. If that is the case then the ring enters the initiated state. This is the state after a ring initiation token has passed once around the ring. It is immediately followed by the second pass and the ring enters the negotiated state, again by consensus of all peers. This corresponds to the choked state in BitTorrent. TCP/IP data transfer connections are established but no data is actually transferred yet.

From here, the ring can move easily back and forth between the negotiated state and the active state (which corresponds to the unchoked state in BitTorrent), depending on how the participating peers choose to play the tit-for-tat game.

Rings that are stuck for too long in one state may be considered expired by its members and may get dropped. If the peers are cooperative then they might let their neighbouring ring members know that they have dropped the ring. Alternatively a keep-alive token may get sent to confirm that the ring still exists.

3.3.2 Peer View

From a peer's point of view there are two goals in managing rings: The first one is to make sure that the most promising rings are chosen for activation and the second one is to try to maximize the utility of active rings.

The purpose of this section is to explore and highlight the decisions that peers need to make. We will not suggest concrete ways of making these decisions in this section because different peers may have entirely different strategies they follow. Therefore this section will leave questions unanswered because there is not necessarily one correct way of doing this. However, in the following Chapter 4, where we introduce one possible implementation of a peer, we will revisit these questions and propose one set of answers that works.

Choosing Rings

Each peer knows about a potentially large number of rings, either by having found them itself or by being informed about them by a neighbour. These rings might be short or long, they might have been found recently or a while ago (and so be unlikely to still exist). They might provide a file that is particularly rare or of special interest to the

peer. In other words, each peers gives each ring a different desirability based on its properties.

Based on these priorities and the current state of active and negotiated rings, the peer will need to decide which rings to initiate itself at which time and whether to forward ring initiation requests by other peers.

A balance needs to be found between the number of active/negotiated rings and the ring initiation rate. On the one hand it is not in the peer's interest to agree to an initiation request for a ring and then not to activate that ring before it expires. On the other hand a peer needs to avoid a situation where one ring has collapsed and there is no other ring ready to be unchoked. In BitTorrent, and also rings of length 2 in our mechanism, no group consensus is necessary because by knowing which pieces another peer is looking for we already know if a ring can reach the choked (negotiated) state.

Each peer needs to maintain a set of priority queues for each state a ring can have. A ring's position on that queue will determine whether and when it reaches the next state on this peer.

For rings in the known state the most important properties is the ring length and the rarity of the pieces on offer by the previous neighbour in the ring (i.e. the neighbour that uploads to this peer). The precise algorithm is ultimately down to the actual implementation of the system and can differ between different peers.

Maintaining Rings

Once a ring reaches the negotiated state, the peer needs to make sure that the ring will be used to maximum effect. The peer can do this by using an incentives mechanism, that is by unchoking it and setting it to the active state, monitoring whether a service is provided in return.

Each peer maintains a set of rings that are active, i.e. an upload is taking place, and another set of rings that are negotiated, i.e. the upload is throttled.

At regular intervals the peer checks the downloads it receives. It unchokes the rings that belong to the highest bandwidth downloads and it chokes all other rings. The checking interval needs to be sufficiently small so that the incentives mechanism works even in longer rings: If a peer chokes an upload then it is essential that its download is choked as well within one or at worst two intervals. If that is not the case then freeload-

ing becomes viable. The longer the ring, the longer it takes for a drop in upload quality to propagate around it. At the same time there is an issue with keeping rings going once they are established: Detecting and setting up a ring represents an investment.

An alternative solution for this problem of immediate feedback is to have peers that choke a ring send a message announcing this fact around the ring. However, this is only in their interest when they have choked the ring as a reaction to a drop of the quality that they receive on the incoming link of the ring.

A peer might even chose to immediately choke rings that have been choked on the incoming link and unchoke rings that have been unchoked on the incoming link. It is up to an individual peer's utility function to settle on a tradeoff between these two.

As with BitTorrent there also needs to be a certain percentage of "optimistic unchokes", i.e. of rings where the peer takes the initiative of uploading to see if the ring can provide a better download than the currently active ones.

Unlike BitTorrent, the number of concurrent rings cannot be fixed. For example, there should be a minimum and maximum number of active rings, to allow a peer to activate a ring when a ring activation request passes through without having to immediately choke another ring.

The longer the ring, the more likely it is to get choked as the constant probabilities of individual peers to choke the ring need to be added up. To counter this effect, peers need to be more lenient to longer rings. This in turn means that the incentives mechanism becomes weaker for longer rings.

However, in the worst case this means that peers are paying over the odds to get the pieces they want. This might still be acceptable, as long as any upload reaches the peer. Especially when the alternative is not getting the pieces at all, which would likely be the case for small and rare files. Again, a peer's actual behaviour depends on its individual utility function.

Since a peer has to assume that longer rings are less stable, peers should always prefer shorter rings, all other things being equal.

Ring Synchronisation

For the tit-for-tat mechanism to be effective, the quality of service that a peer receives on a ring needs to follow the quality of service that it provides as closely as possible.

The longer the ring, the more intervals it takes for the unchoking of ring segments to propagate through the whole ring. But if there is a substantial delay between action and response then peers may try to play the system in order to maximize their utility. This is not in the interest of the other peers in the ring. Therefore peers should react to changes as quickly as possible. The easiest way of doing this is to have peers monitoring the set of incoming flows that are associated with known rings constantly, look out for signs that rings have been unchoked on an uplink and act immediately by unchoking the downlink leg of the same ring immediately. That way, all segments of a ring become unchoked rapidly and the incentives mechanism can work.

3.4 Related Work

3.4.1 Ring-based Incentives Systems

A number of papers have picked up on the idea of using rings for incentives mechanisms.

Reciprocity and Barter in Peer-to-Peer Systems

(Menasché et al., 2010) suggest a tit-for-tat incentives mechanism that implicitly uses rings. Instead of having one-way downloads around the ring, it is based on bartering content between the members of the ring. Peers download content that they are not interested in themselves but that another member of the ring is interested in. They download it from one neighbour in a tit-for-tat exchange with the intent of using it in another tit-for-tat exchange with another neighbour. The authors call this indirect reciprocity: *I help you and someone helps me.* In effect, two-directional rings are constructed by shifting the desired content in both directions to the receiver. This allows to maintain a simple two-node tit-for-tat strategy on all links.

The disadvantage of this approach is an up to twofold increase in required bandwidth, because the peers need to download and forward content that they are not interested in themselves. Popular content can be worth caching on the peers, in which case this number can be lower. The authors can prove that there is an upper bound of two on the loss of efficiency.

The authors do not address the problem of discovering these rings, because since the peers only use direct reciprocity, they do not need to know about rings. It is sufficient

that the rings exist. The evaluation is done in a 200 node network with two types of artificial content distributions, where all content is of the same size.

Nash Equilibria of Packet Forwarding Strategies in Wireless Ad Hoc Networks

(Félegyházi et al., 2006) introduce a model for wireless ad-hoc networks that is closely related. The basic problem is slightly different in wireless networks, where the most interesting service is simply forwarding each other's packets. This is because unlike in peer-to-peer networks, there is no basic infrastructure that can be used. Instead, the peers need an incentive just to forward packets from others.

The basic game theoretic model of the problem is very similar to ours, simply because we have derived our analysis of the peer-to-peer cooperation problem from this analysis of the wireless cooperation problem. The payoff function and the strategy space are very different though for the two different problems.

Then the authors introduce the dependency graph, which is derived from the routes the packets take and which correlates to the service overlay graph in our scenario. The automaton as a model for each node in the dependency graph bears some resemblance to our peer model. It is simpler in that it only consists of an input component, a strategy function component and an output component and does not discriminate between different packets or flows and their source and destination.

The following analytical results succeed in finding a Nash equilibrium of all peers cooperating for the given scenario, but only under very severely limiting assumptions which render the analytically derived equilibrium useless for real-life scenarios: All peers have to have a dependency loop with all of the sources for which they forward packets, and all peers have to play a tit-for-tat strategy.

Exchange-Based Incentive Mechanisms for Peer-to-Peer File Sharing

(Anagnostakis and Greenwald, 2004) suggest an incentives mechanism that is based on exchange rings. As an extension of the direct reciprocation scheme of BitTorrent, peers would form rings where each peer uploads to one neighbour and downloads from another neighbour in such a way that the downloads form a ring.

In order to construct rings, each neighbour would construct a "request tree" of depth 5 which contains all other peers requesting a service and their request tree. This

tree then gets pruned and attached to any request the peer sends out. This way, each peer can recognize rings of a size up to the depth of the tree.

Once a possible ring is established, a token is sent through the ring to check that everyone is still willing to serve. The transfer rate is also negotiated this way as the minimum of all proposed transfer rates.

There are a number of severe limitations in the model and the simulation scenario that are used in this paper. Some are mentioned by the authors themselves: the heterogeneity of the capabilities of peers and the costs for a peer of sending the complete request tree with every request. Others limitations are the small size of 200 peers of the network used for the simulation and the unrealistically high request rate of files, which leads to an equally unrealistically high probability of rings.

3.4.2 BitTorrent-like Systems

In the area of file sharing, BitTorrent (Cohen, 2003) is currently by far the most popular protocol that uses a tit-for-tat strategy. Because our work is based on BitTorrent, we have provided a detailed description of the BitTorrent mechanism earlier in Section 3.1.2.

However, BitTorrent is far from being perfect. There are a number of papers on analysing and improving BitTorrent-like algorithms for bulk content distribution (Qiu and Srikant, 2004) (Koo et al., 2005) (Sherwood et al., 2004) (Ganesan and Seshadri, 2005) and there are also observations by (Andrade et al., 2005) that existing BitTorrent communities use additional methods to force people to share because cooperation levels achieved by the standard mechanism are insufficient.

Another major problem of BitTorrent is that it is limited to a fairly specific scenario: bulk file dissemination of popular content. The resource that is shared also needs to be partitionable (such as a file or a computation). This is because BitTorrent relies on peers being interested in each other's offers simultaneously, and that works best if peers are offering parts of the same resource that they are all interested in.

BitTorrent is also un-optimised in that it will always use all available outgoing bandwidth and only try to optimise the incoming bandwidth. Based on this observation, (Jun and Ahamad, 2005) conclude that BitTorrent is in fact inviting free-loaders when compared to the theoretical optimal tit-for-tat mechanism.

A further problem is that it is only incentives-compatible when there is at least one complete copy of the file shared between the peers. At the beginning, at least one altruistic peer that has a complete copy of the file, a so called “seeder”, needs to join and upload pieces of it to the other peers without getting anything in return. This peer must have an incentive for its actions that is external to the peer-to-peer network.

3.5 Summary

We have proposed an incentives mechanism for file sharing that supports the sharing of small and rare files, while maintaining a lightweight system that neither requires persistent state between sessions nor external incentives.

For this purpose we have suggested a generalised BitTorrent like system that allows the exchange of pieces of different files and that detects and maintains incentives rings where pieces are exchanged in a ring with three or more participating peers.

We are proposing a random walker token mechanism to detect cycles in the service overlay graph. In regular intervals tokens that contain a unique ID and the token path are sent to a random neighbour, and received tokens are forwarded to another neighbour. This allows peers to find cycles by comparing the token path with a neighbourhood tree constructed from the paths of previous tokens.

For operating the rings we propose simple group consensus algorithms for starting and stopping the rings, and a BitTorrent-like incentives mechanism for the tit-for-tat strategy implementation. The peers need to maintain a list of rings in various states ranging from known to active, and they have to make decision of when to transition rings to a different state.

We have discussed related work and found that the proposed principles are novel.

Chapter 4

Design

In this chapter we will design an actual system from the principles that we have defined in the previous chapter. There can be different implementations of the principles on different peers, as long as the interaction between peers remains the same. Therefore, our proposed design is a proof of concept. It shows that the principles can actually be implemented.

The system that we design consist of nodes, of services between those nodes (in our case limited to file downloads) and of rings that are made up of some of these services. Each node has a network connection by which it can communicate with all other nodes. A service is the provisioning of some kind of resource over time by one node to another node. We assume only one kind of service, namely that of file sharing. File sharing provides both the content of a file as well as the upload bandwidth over time. It takes place over the nodes's network connection.

We assume that there is also a file search facility in the network, either run by BitTorrent-like trackers for motivations outside the peer-to-peer network or by a distributed hash table (DHT) on the actual nodes, or by a combination of both. We assume that this search facility can be queried in order to receive a list of peers that are interested in any particular file. One could argue that a DHT would need an incentives mechanism to function as well but we regard that as a separate problem that is not part of this thesis.

4.1 Nodes: Seeds and Peers

A node is an entity that controls a number of hardware and software resources. It runs on a computer that is separate from all other nodes. The computer has storage space to

keep a number of files, and it has a network connection with which it can communicate with other nodes and exchange files with them.

Some nodes only provide services and never receive them. These nodes are called *seeds* in BitTorrent as they are used to provide the initial copy of a file to the peers, and we have adapted that term. Other nodes can both provide and receive services, and we call these nodes *peers*. All nodes are either seeds or peers. Seeds are not particularly interesting to us as they are not participating in the incentives mechanism. Instead, we will focus on peers.

Seeds may or may not provide services to peers, and a peer can do nothing about it. Peers register their interest in a file with the seed as they would do with any other peer, but there is no further interaction. Therefore, peers will take what they can take from seeds, but they will not consider seeds any further in their model of the network.

We will look at peers in detail later in this Chapter from Section 4.5 onwards, but first we need to focus on the other building blocks of the system: files, services and rings. This will set the scene for the actions of the peers.

4.2 Files and Pieces

A file has a name and a size and content. Because the file name might differ on different nodes, a hash function value that is derived from the file content is being used for identification purposes. A hash function such as MD5 (Rivest, 1992) or similar is sufficient for this purpose. A file consists of one or more *pieces* of the same size. This piece size is determined by the original seed and then fixed. Each piece has either finished downloading or it has not. It can only be downloaded from one peer at once. Different pieces of the same file can be downloaded from different peers at once. This is similar to the file and piece model of BitTorrent.

Figure 4.1 illustrates this. It shows a file that is on a peer (which is not shown explicitly in the figure). Some pieces are already downloaded while others are not, and two pieces are in the process of being downloaded from two other nodes.

Each peer offers and requests a certain set of files. Once a requested file has finished downloading, the file is being offered by that peer. This is rational for the peer as offering additional files increases the probability of rings.

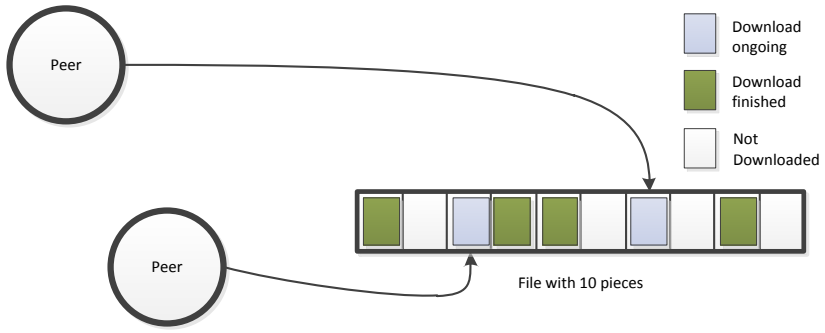

Figure 4.1: File with Pieces and two Downloads

4.3 Services

A service $s(p_{from}, p_{to}, t)$ is provisioning a resource from one peer p_{from} to another peer p_{to} at a time t with a measurable quality $q_s(s,t) \in [0..1]$, where 0 means that the peer receives no service at all and 1 means that the peer receives the best possible quality of service. In the case of a file sharing service, that quality is the download speed.

A service might have an end to it, for example a file that is eventually downloaded. If the resource that is associated with the service is of a more general kind, such as disk storage space, then this might not be true. If it does have an end then the completion ratio can be queried.

A service can also be virtual: a potential service may not have been discovered yet, or a potential has been discovered but not acted upon. These states are meaningful because in the second case a ring with that service can be found, but not so in the first case.

4.3.1 Service States

A service can be in two different states.

choked the service is generally set up and ready to go, but the provided service quality $q_s(s,t) = 0$. This is the initial state of a service.

unchoked the service is being provided at a best effort quality $q_s(s,t) = [0..1]$. The incentives mechanism takes no active control over the service quality. A number of these services can run at any time, in which case they would share the available resources.

One can think of other states such as a quality-controlled state where the providing peer decides on the precise quality of a service are possible but not relevant in the file sharing scenario and so we will not consider it in this thesis.

4.3.2 Service Life Cycle

A service starts out in a choked state, as soon as both peers agree on commencing the service. It then gets unchoked by the providing peer. After this point, the service can go through any number of provider-induced choking/unchoking cycles, until the service is finished or terminated. A service can be terminated by either peer.

A service can go from choked to unchoked and back at any time. Due to the necessity of measuring the flow speed it may take the receiving peer a short amount of time to realize that a service has changed status. Therefore the state of a service is not the same on both ends at all times.

4.4 Rings

A ring is a cycle in the directed graph of peers and of potential or actual services. Rings even exist when they have not been discovered by peers, they are simply a property of having peers and having potential services between them. Once they have been discovered they can be used by the incentives algorithm on each of the participating peers.

4.4.1 Ring Formation

Rings appear and disappear by virtue of the underlying interest between peers. If all peers of a ring happen to offer and request the right services then the ring forms. If one of the services in a ring stops being requested or stops being offered then the ring disbands. Rings can go entirely unnoticed by peers for the entirety of their lifetime.

Peers can facilitate the formation of rings by increasing the number of services they offer and request, and by letting as many other peers know as possible.

4.4.2 Ring Life Cycle

At some point a node may become aware of the existence of the ring by using a ring detection mechanism. It will then propagate a ring initiation message to the next node of the ring, which will forward the message until it gets back to the sender. At this stage all nodes are aware of the ring and it could be used for the tit-for-tat algorithm. For the

nodes it now becomes a matter of group consensus whether or not to start the ring, and when.

Starting a ring effectively means that all nodes agree to start their uploads within the next checking interval. Now all the nodes play tit-for-tat against the ring, with the same rules as in a ring of 2. When a node is not satisfied with the service quality then it can either lower the quality (not available for file sharing) or choke it completely. When the quality picks up again, the ring will be unchoked. As long as all nodes agree on the ring it keeps running. Only when a node decides to terminate the ring will it stop completely and either drop back to the know but not used state or, if the service terminated successfully, disappear entirely.

4.5 Peers

After having introduced the other elements of the system, we can now revisit the peers and explain the processes in the system as actions of those peers.

We assume that a peer cannot know what services other peers consume or provide other than the services the peer is engaged in itself. The only parameters a peer knows about is what services it provides and receives from other peers it knows, and the quality of those services. This includes peers to which a service relationship has already been established, but also potential service partners the peer knows about. A peer might also keep a record of previous values of these parameters, as that might help in predicting future behaviour. However, since there is no concept of persistent peer identity these records cannot be kept between sessions.

There are a number of tasks a peer has to perform. Each peer needs to be able to provide uploads to and receive downloads from other peers. It needs to be able to search for files. It needs a token distribution mechanism for the communication with other peers and for ring discovery. The peer needs some ring detection logic that processes those tokens and it needs to have an incentives mechanism that manages the uploads.

4.5.1 Tasks of a Peer

Now that we have defined the important elements of our system, it is time to focus on the processes in the network. Since nodes are the only acting elements in the system, it is sufficient to look at their processes to model the complete system. Seeds simply

provide services randomly and do nothing else, and so we can narrow this down further to model only peers in detail.

There are a number of different tasks that peers must be able to perform. We can group them into seven distinct categories. Each category can be implemented in a separate module. We discuss each of these modules in turn. Figure 4.2 gives a graphical representation of the most important links between these modules.

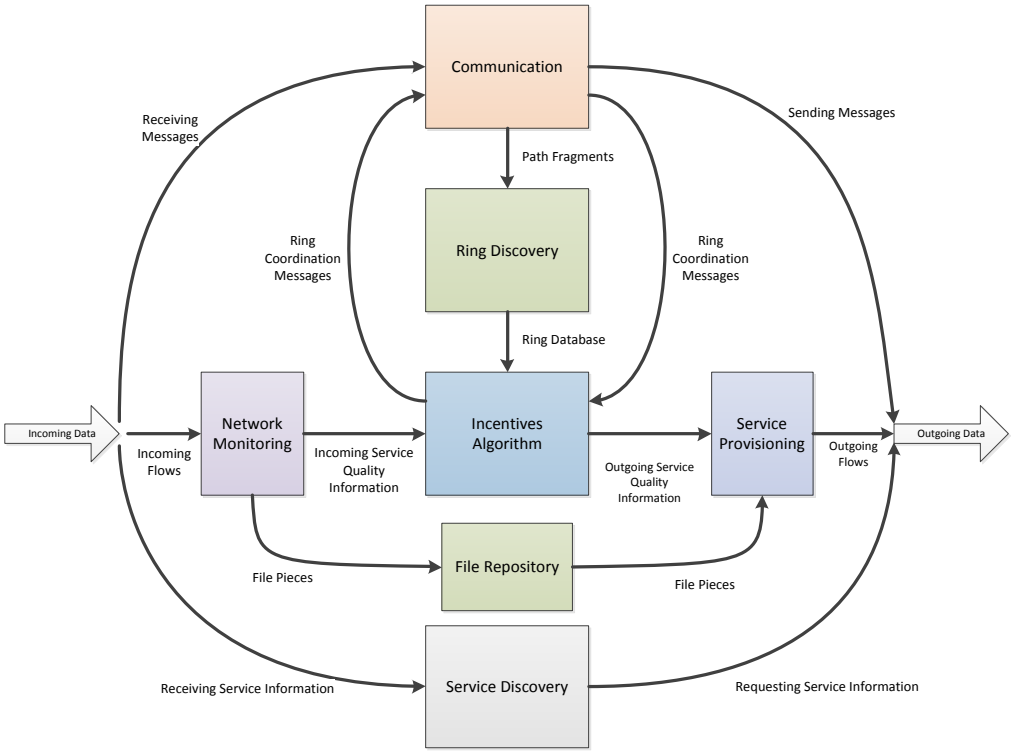

Figure 4.2: Peer Overview

Network Monitoring Peers need to gather information about the current state of the network uplink, or more specifically of the flows that it sends and receives. They need to know how much bandwidth is used for the communication with other peers (messages and tokens) and how much downlink bandwidth each of the received services, or rather its flow, gets. The module provides this information to the **Incentives Algorithm**.

File Repository This is where the actual files and pieces are stored. It takes pieces from incoming flows and provides pieces to the **Service Provisioning**.

Service Provisioning Peers need to be able to provide services to other peers in a controlled fashion: they need to choose which other peers to upload to and which not. For some service types other than file sharing they might also want to control the quality of services. The module gets the service quality information from the **Incentives Algorithm**.

Communication Peers must be able to communicate with any other peer about files and pieces on offer and for coordinating rings. For this communication some kind of messaging infrastructure has to be in place. This module creates and forwards ring search tokens, and it tells the **Ring Discovery** about path fragments found in incoming ring search tokens. It also sends and receives ring coordination messages for the **Incentives Algorithm**.

Service Discovery A peer must be able to find out about other peers that can offer services that the peer requests. This could be done by forwarding the query to a central search instance or by using a distributed directory that is shared between all peers, or by a combination of both. In either case this module needs to communicate with other nodes. Because service discovery is not part of this dissertation we cannot make statements about its implementation and we consider it separate from other **Communication** between peers. We assume that the service discovery module is able to return a list of peers that have a given file.

Ring Discovery The knowledge of paths to other peers that is received by the **Communication** module in RingSearchTokens needs to be collected and processed for rings. Path information is therefore forwarded to this module and the resulting ring database is made available to the **Incentives Algorithm**.

Incentives Algorithm Once a ring is active, a tit-for-tat algorithm needs to be applied in order to stimulate cooperation. Based on the incoming service quality information by the **Network Monitoring** module, the ring database information by the **Ring Discovery** module as well as the current status of ring negotiation with other peers, which is handled by the **Communication** module, this module decides on the outgoing service quality and tells the **Service Provisioning** module.

In the remainder of this chapter we will discuss each of these modules in more

detail.

4.5.2 Peers and the Network

Generally we define a service as one peer's resource usage by another peer over time, coupled with a well know function to perform such as a file download or a computation following a particular algorithm.

The service may also consume resources on the peer that is using the service, in addition to the resources consumed on the providing peer. A typical example is a file download, since it uses the receiving peer's downlink bandwidth as well as the sending peer's uplink bandwidth. However, this is also true for computational services, where the data to process needs to be transferred to the other peer.

For each service or group of services there is a limit on the achievable quality because resources are finite. In the case of computational resources the providing peer can allocate the CPU time to the different services and it is in full control of the distribution of the resource. However, that is not the case with a download, where the limited bandwidth of the *receiving* peer might be the bottleneck, reducing the service quality below what the sending peer would be willing to provide. For example, if the receiving peer of a file download is connected with a very slow 64 kBit/s link to the Internet then that is the upper limit of the service quality, even if the sending peer would be willing to provide 300 kBit/s.

Furthermore, it is essential that the quality of service can be evaluated in one way or another. This is trivial to do for a file download, where it is simply the download speed, but more difficult for more complex services, such as a lookup service. In the latter case the quality would have to be determined by ”soft” factors such as the time it takes for the answer to arrive and of course the quantity and accuracy (does the peer still have the service on offer?) of the search results.

The specific quality metric for a service has to be convertible into a generic service quality metric, in order to be to able to compare different qualities of different types of services.

In the remainder of this chapter we will assume for simplicity that there is only one service type: file sharing. The consequence is that there is only one quality limit on the combination of all services, namely the peer's bandwidth.

4.6 Network Monitoring

The network monitoring process needs to be able to monitor network flows, both their originating peer as well as their bandwidth used. More specifically the monitoring is actually only interested in the bandwidth used for the download of actually requested files and not of the total bandwidth that includes metadata or any other traffic that might occur.

In fact, the peer is not interested in absolute number of bytes per second but rather in a more abstract service quality scalar, ranging from 0 for no download of payload data at all to 1 for using all of the downlink bandwidth of the peer. The sum of all service qualities on a peer cannot exceed 1.

4.7 File Repository

The file repository is responsible for storing the actual content on a peer. The content is organised in pieces, which can be obtained from uploads or through other means outside the system such as memory sticks. Knowing which pieces of a file are available and which pieces are requested is also important for the ring discovery mechanism. The file repository provides pieces of files to the service provisioning module, which we will describe next.

4.8 Service Provisioning

The service provisioning process makes sure that all services are run at the quality assigned to them by the Incentives Algorithm (see Section 4.12).

If a service quality was previously at 0, which means that no service was actually provided while it would have been possible to do so (i.e. there was demand by the other peer), then the service will be started. Likewise, if the new service quality is 0 while the previous quality was above 0, then the service will be stopped or terminated.

4.9 Communication

Peers communicate with each other using messages. For identification purposes peers have 64-bit identity numbers that they choose randomly at the beginning of each session.

Peers are neighbours in the service overlay graph when one of them is interested

in a service that the other offers. Each peer has two sets of neighbours: Those that can provide a service and those that can receive a service. Peers only communicate with direct neighbours in the service overlay graph.

There are three classes of messages. The first class contains four different message types and deals with exchanging piece information with neighbours. The second class (3 message types) transmits the tokens for the ring discovery algorithm. Finally the third class deals with communication between peers of already established rings. There are two message types in this class.

InitialPiecesMessage A bit array of which pieces of a file are requested (0) and which are available (1) on the sending peer. This also doubles up as the initial contact message for new neighbours: A peer searches for a file and then contacts a number of the resulting peers with this message.

OfferPieceMessage A particular piece of a known file is now available from the sender when it was not before. The sender finished downloading that piece from a third peer.

RequestPieceMessage A particular piece of a file is now requested by sender when it was not before. This can happen for example when the sender has lost a piece since the InitialPiecesMessage.

RemoveFileInterestMessage A notification that the sender does not request or offer the file any more. This message will be sent by peers that either are not interested in acquiring a file or peers that have deleted their copy of a file.

IncomingRingSearchTokenMessage This message contains a token that is received from neighbouring peers that a peer is uploading to and is potentially sent to neighbouring peers that a peer is downloading from.

OutgoingRingSearchTokenMessage This message travels the opposite way of the previous one. It contains a token that is received from neighbouring peers that a peer is downloading from and is potentially sent to neighbouring peers that a peer is uploading to.

RingFoundTokenMessage This message is sent around a newly found ring in order to make its existence known to all peers that are part of it.

RingNegotiationMessage This message is sent as part of the negotiations to start rings. It travels around the ring twice if all participating peers are willing to start the ring.

RingChokeMessage This message is sent by peers whenever they choke a ring. When a peers receives such a message from the upstream neighbour of a ring then it chokes the ring as well and passes the message on to the downstream neighbour of the ring.

4.10 Service Discovery

The Service discovery process maps a file name to a list of peers that offer that particular file. This list may be a subset of all peers in the network that offer the file.

We will not consider concrete implementations of the service discovery process of peers in detail in this thesis. Our model of service discovery is a mapping from a file (identified by its hash value) to a set of peers that are interested in that file. Those peers might have the file either completely or partially, and if they do not have it completely then we assume that they are interested in obtaining the missing parts.

4.11 Ring Discovery

Ring discovery is the first of the two core functionalities of a peer in our model, with the second—the incentives algorithm—following in the next section. Rings are discovered by propagating token messages and by looking at the paths that the tokens have travelled along. Each ring does this independently but peers inform the other members of a ring once they have identified said ring. As rings are the foundation of the incentives mechanism, peers have an incentive to share information about the ring with all others peers of that ring.

4.11.1 Token Messaging

The messaging system is also used for the exchange of ring detection tokens and ring initialisation tokens. The main difference to the other messages is that tokens can travel more than one hop.

All tokens have a TTL (time to live) field that allows peers to discard tokens that have travelled too far to be useful. The initial TTL is set by the peer that generated the token to different values depending on its type. Tokens can also contain a path in the service overlay graph. A path is simply an ordered set of peer IDs.

RingSearchTokens contain a unique token ID and the path that the token has travelled so far as a sequence of peer IDs. This path is then used to identify rings. These tokens also contain a flag that tells whether this token gets forwarded via uplink or downlink neighbours. The TTL is set to half the maximum ring size.

RingFoundTokens also contain a token ID as well as the path of a ring as a sequence of peer IDs. They are sent by the discovering peer around a newly found ring in order to establish it. They are always sent towards the next downlink neighbour in the ring. The TTL is set to twice the ring size so that the token can travel around the ring twice before being dropped.

4.11.2 Ring Identification

The ring identification algorithm is an implementation of the Random Walker Tokens approach that we have discussed in Section 3.2.2.

Ring Search Tokens

At a regular interval each peer creates a new RingSearchToken with an initial TTL value of half the desired maximum ring size and sends it to at least one randomly chosen neighbour. Half the desired maximum ring size is enough to find many (but not all) rings of that size because rings are composed of two token paths, one from each direction. The interval is a tradeoff between the network traffic created and the number of rings found, and is not fixed and up to the peer to decide, but times around one token every second have worked well during the evaluation phase. The length of the interval could also change with the number of known rings: if not enough rings are known then the token generation rate should increase. Vice versa, if there are already enough known rings that could be initiated then there is little point in finding even more rings.

Tokens are sent in both uplink and downlink directions. If a peer is both uplink and downlink neighbour, which is the case when both peers are currently downloading the file, then the sender decides whether the generated token should be one that travels uplink-wise or downlink-wise through the ring. Which direction the token travels

through the ring makes a difference because all rings are directional and this decision will determine which path a token can take after the first hop.

Peers will receive RingSearchTokens from their neighbours. Using the path within each token, which tells what peers this token has passed through on the way from the originating peer to the current peer, peers can create a database of shortest known paths to other peers within the service overlay graph. Since RingSearchTokens are directional there are two tables on each peer, one for each service provisioning direction that a token can arrive from: either a potential uploading neighbour or a potential downloading neighbour.

Once a token has been processed its TTL will be reduced by 1. Unless the TTL reaches 0, the token will be forwarded to one or more neighbours (depending on the strategy of the peer). Because direction is important in rings, tokens received from an uplink neighbour will always be forwarded to a downlink neighbour, and vice versa.

The Path Database

The two path tables that make up the database contain the shortest known paths to all other known peers. Because rings are directional there is one table for each direction.

Every time a token is received the peer compares the token path and the path database to see if a new ring has been discovered. This is the case when the peer knows about a path to one other peer in both directions. If that is the case then the combined paths comprise a ring.

So for each peer in the token's path (which is part of the token payload), the peer checks whether a path to that peer in the opposite direction of the token exists in the corresponding path table. If it does exist then both paths can be combined to form a ring.

In order to test that the ring still exists, and also to make sure that all members of the ring know about it, the peer then initiates a RingFoundToken and sends it to its downlink neighbouring peer. This neighbour then does the same if it is interested in the ring, and so on. Once the token has been sent successfully around the ring twice, all members know that all other members are interested in it, and it enters the set of rings that can be used by the incentives algorithm, as detailed in the following section.

Algorithm 4.1 discoverRings(receivedIncomingTokens, receivedOutgoingTokens)

```
sendToken(new Token(new Path()),getRandomNeighbour());
for all token ∈ receivedIncomingTokens do
  path ← token.getPath();
  rememberIncomingPaths(path);
  for all peer ∈ path do
    if outgoingPaths.get(peer).size() > 0 then
      addRing(this ⊕ outgoingPaths.get(peer).getShortest() ⊕ peer ⊕
      path.subPath(peer));
    end if
  end for
end for
for all token ∈ receivedOutgoingTokens do
  path ← token.getPath();
  rememberOutgoingPaths(path);
  for all peer ∈ path do
    if incomingPaths.get(peer).size() > 0 then
      addRing(this ⊕ path.reversedSubPath(peer) ⊕ peer ⊕
      incomingPaths.get(peer).getShortest());
    end if
  end for
end for
for all token ∈ receivedIncomingTokens ∪ receivedOutgoingTokens do
  if decideToForwardToken(token) and token.getRemainingTTL() > 0 then
    token.decreaseTTL();
    token.getPath().add(this);
    sendToken(token, getRandomNeighbour(token.getDirection()));
  else
    token.destroy();
  end if
end for
```

See Algorithm 4.1 for a pseudo-code representation of the core ring detection algorithm. This algorithm runs once every interval. First a new token is generated and sent out to a neighbour. Then the path information is extracted from all received tokens and new rings are checked for. Finally, the received tokens are forwarded if desired and if there is TTL (time to live) left.

The path database can be implemented in various ways, but the tables need to support a mapping from a peer to a set of paths to that peer, and they also need to support the removal of all paths that contain a particular peer. One possible implementation which we have used in the simulation in Chapter 5 is to use hash tables that map from peer IDs to list of paths ordered by length.

Algorithm 4.2 rememberIncomingPaths(path)

```
for all peers ∈ path do
    subPath ← path.subPath(peer);
    paths ← incomingPaths.get(peer);
    if paths.size() < pathsToRememberPerNode then
        paths.add(subPath);
    else
        longestKnownPath = paths.getLongest();
        if subPath.size() ≤ longestKnownPath.size() then
            paths.remove(longestKnownPath);
            paths.add(subPath);
        end if
    end if
end for
```

Based on this model, Algorithm 4.2 shows how to update the path database at the arrival of a token. We only look at the algorithm that deals with tokens that arrived from an uplink neighbour, but a similar implementation for tokens arriving from a downlink neighbour is required as well.

Paths in the database get deleted when the first hop peer ceases to be a neighbour. When path information is outdated and a ring is falsely discovered based on that outdated information then the RingFoundToken message will not successfully pass through the ring twice and so the ring will not get established. If that happens then the path to a more distant peer that fits the path in the ring will get deleted as well.

Ring Detection Example

To illustrate the algorithm let us consider the following example which centres around a peer that we will call Peer **X**, at the moment in time when it receives a token that will allow it to find a ring.

See Figure 4.3 for a graph of the part of the overlay service graph that is know to Peer **X**. It shows the other peers that Peer **X** knows about and the potential downloads between them. This knowledge is gathered by processing the five tokens that have reached the peer in this scenario: three tokens with a downlink direction that have been received from potential uplink neighbours and two tokens with an uplink direction that have been received from potential downlink neighbours.

Let us now consider the steps taken by Peer **X** in this example when the marked token is received from downlink neighbour Peer **D**. The first step is to update the path table from the information contained in the token. This token's path is $[\mathbf{A}, \mathbf{B}, \mathbf{D}]$ when it

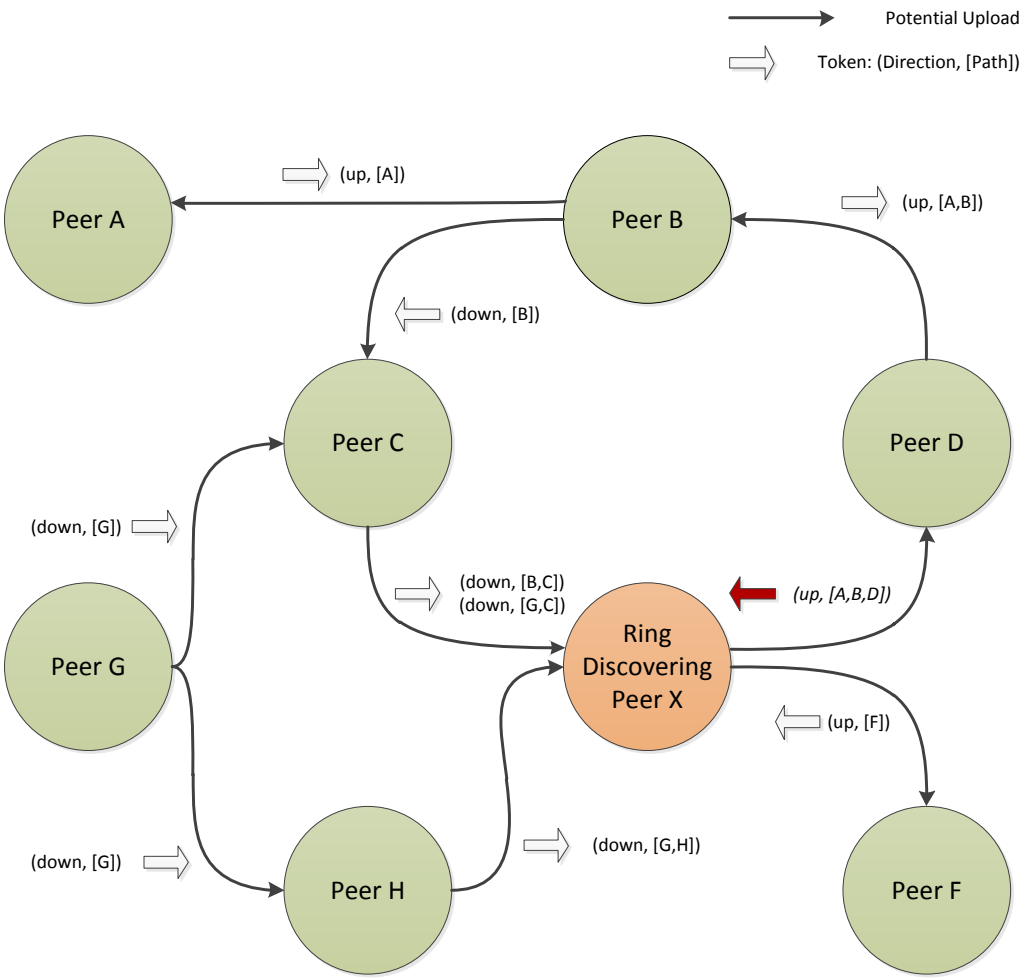

Figure 4.3: Ring Detection Example

arrives. From this it can be concluded that Peer **D** must be a direct downlink neighbour as this is a token that is sent to uplink neighbours. Of course this is something that is known already. However, the token path also contains two paths to other, more distant neighbours: [**D**] is a downlink path to Peer **B** and [**D**, **B**] is a downlink path to Peer **A**. Since both peers are new as downlink neighbours they are added to the downlink path table (see Table 4.2), together with the newly discovered paths.

In the second step the peer scans the uplink path table (see Table 4.1) for any of the peers in this token's path. Peer **D** has no entry in uplink path table because there is no known uplink path to it. There is however a path for Peer **B**: [**C**]. This means that a ring has been found as there is a known path to Peer **B** in both uplink and downlink directions.

Peer	Paths
B	[C]
C	[]
G	[C], [H]
H	[]

Table 4.1: Uplink Path Table

Peer	Paths
F	[]
J	[F]
D	[]
B	*[D]*
A	*[D,B]*

Table 4.2: Downlink Path Table

All that is left to be done is to construct the ring by concatenation of the downlink path, the common peer, the uplink path and this peer: $[\mathbf{D}, \mathbf{B}, \mathbf{C}, \mathbf{X}]$. Then, a RingFoundTokenMessage is sent to downlink neighbour Peer **D**. If the ring still exists then it will shortly thereafter be received from Peer **C**, which will trigger the second time that it is sent to Peer **D**.

4.12 The Incentives Algorithm

Once a peer knows about a number of rings, the incentives algorithm can come into action. The actual incentives algorithm chooses which rings should get an upload slot and which should not, based on the download performance of those rings.

Since each peer might have a different utility function, we cannot predict the choices that individual peers will make. The concrete algorithm that we are presenting is therefore only a proof of concept, a suggestion to show that our approach works. Any individual peer may run a different algorithm as long as it keeps the inter-peer communication compatible.

However, mechanisms need to be in place to actually execute the choices made by the incentives algorithm: Each peer must try to make sure that there are always enough rings running. Peers must find a consensus on when to start rings. The actual unchoking of a ring then needs to be synchronised among the participating peers. Also, peers should be able to explicitly communicate to the other peers that they choke the ring. The other peers can tell from the lack of service provisioning, but an explicit notice will lead to a faster reaction and will thus improve the incentives algorithm. This might not be in the short-term interest of the peer as it will stop the download, but it might be beneficial to it over the lifetime of the ring. Again, this is a choice to be made by individual peers.

4.12.1 The Core Incentives Algorithm

Algorithm 4.3 shows a pseudo-code implementation of the core incentives algorithm. It is executed once per interval.

Algorithm 4.3 incentivesAlgorithm()

```
for all ring ∈ getActiveRings() do
  if currentTime() − optimisticUnchokeProtectionPeriod >= getUnchokeTime(ring)
  then
    removeChokeProtection(ring);
  end if
end for
for all token ∈ getExpiredRingNegotiationTokens(getCurrentTime()) do
  removeRing(token.getRing());
end for
for all token ∈ getRingSearchTokens() do
  if token ∈ getExpiredTokens() then
    deleteToken(token);
  end if
end for
for all ring ∈ getIdleNegotiatedRings() do
  if idleCounter(ring) > negotiatedRingsIdleTimeout then
    dropRing(ring);
  end if
end for
for all ring ∈ getIdleActiveRings() do
  if idleCounter(ring) > activeRingsIdleTimeout then
    chokeRing(ring);
  end if
end for
if negotiatedRings.size() < maxNegotiatedRings then
  initiateRing(chooseRingToInitiate());
end if
chosenRings ← chooseRingsToActivate();
for all ring ∈ getActiveRings() do
  if ring ∈ chosenRings then
    chokeRing(ring);
  end if
end for
for all ring ∈ chosenRings do
  if ring ∈ getActiveRings() then
    unchokeRing(ring);
  end if
end for
```

The algorithm consists of eight steps. First it removes the choke protection on earlier optimistic unchokes. Optimistically unchoked rings are protected from being choked for a few intervals. Then it removes expired ring negotiation tokens and their

rings. Expired means that the peer did not receive back the token within a timeout period.

Next, it drops idle negotiated rings. Negotiated rings that have not been used for a very long time are discarded as they likely do not exist any more. Then it chokes idle active rings. Active rings where the peer has not received any service for a few intervals are set to the negotiated state.

If the threshold of negotiated rings has not been reached then it initiates the negotiation of additional rings. One or more new rings are chosen by the peer from its pool of identified rings. We will look at possible selection criteria in Section 4.12.2. Then it determines the list of rings that should be active during the interval. This is the very core of the incentives algorithm and we will look at it in detail in Section 4.12.3.

Then it chokes active rings that are not in the active rings list. These are the rings that have dropped in performance and that should not be run any longer. And finally it unchokes negotiated rings that are in the active rings list. These are the rings that have performed better than before and that should subsequently be supported.

Ring Negotiation Messages

In addition to these regularly schedules steps the peer also needs to process RingNegotiationTokens as soon as they arrive. Algorithm 4.4 shows the function that is triggered by incoming negotiation messages. If the message is for a ring that is neither initiated, negotiated nor active, and if the peer agrees to initiate it, then it forwards the message to the following peer in the ring. If it has done so before and the ring is in the initiated state, and if the peer still wants to negotiate the ring then it forwards the message once more (unless it started the initiation, in which case the TTL will be 0). As this algorithm is part of the ring initiation, we will talk about it in more detail in Section 4.12.2.

Ring Choke Messages

Finally, a peer needs to act immediately on incoming RingChokeMessages. Algorithm 4.5 shows the function that deals with those messages. It simply checks if the ring is active. If that is the case then it chokes the ring and forwards the message to the next peer in the ring. The purpose of these messages is that when a ring gets choked then all peers choke it as near simultaneously as possible, so that a non-cooperating peer that caused the initial choking gets an immediate feedback. This is necessary to

Algorithm 4.4 receiveRingNegotiationMessage(message)

```
ring ← message.getToken().getRing();
if isInitiatedRing(ring) = false and isNegotiatedRing(ring) = false and isActiveRing(ring) =
false then
    if chooseWhetherToInitiateRing(ring) = true then
        addInitiatedRing(ring);
        if message.getTTL() > 0 then
            message.reduceTTL();
            forwardRingNegotiationMessage(ring.getNextNeighbour(), message);
        end if
    end if
end if
if isInitiatedRing(ring) = true and getSeenMessages().contains(message) then
    if chooseWhetherToNegotiateRing(ring) = true then
        removeInitiatedRing(ring);
        addNegotiatedRing(ring);
        if message.getTTL() > 0 then
            message.reduceTTL();
            forwardRingNegotiationMessage(ring.getNextNeighbour(), message);
        end if
    end if
end if
```

enforce the incentives mechanism.

Algorithm 4.5 receiveRingChokeMessage(message)

```
ring ← message.getToken().getRing();
if isActiveRing(ring) = true then
    chokeRing(ring);
    if message.getTTL() > 0 then
        message.reduceTTL();
        forwardRingChokeMessage(ring.getNextNeighbour(), message);
    end if
end if
```

Without RingChokeMessages peers would need to rely on the received service qualities to detect if a ring is choked on the previous peer of the ring. As they are only checked periodically this means that it can take a while until all peers of a ring have realised this. The last peer to realise it is the one that provides a service to the peer that originally caused the problem, and so peers that do not provide a service would get the best service in return. This is clearly not in the interest of the other peers of a ring and so they have an incentive to send RingChokeMessages whenever they choke a ring.

4.12.2 Ring Initiation

The ring initiation is concerned with the process of getting all peers in a ring to agree on starting services in a ring simultaneously. This requires a simple distributed consensus algorithm and the infrastructure for peers to communicate with each other by means of messages.

Algorithm

The ring management algorithm needs to make sure that there are always enough rings for the tit-for-tat algorithm to unchoke. That means that rings need to be brought from the known state to the negotiated state. So the purpose of the ring initiation algorithm is to find group consensus on the intent to start a ring within the next few intervals. We assume that the ring discovery algorithm maintains a set of discovered but otherwise un-used rings. The incentives algorithm can query that set of rings.

One peer now generates a RingNegotiationMessage. This message needs to be passed around the ring twice, first in the initiation phase and then in the negotiation phase. In the initiation phase a peer agrees to set the ring to negotiated simply by passing on the message. The second phase serves as the confirmation phase. When the message is received the second time then a peer knows that all other peers forwarded the message and therefore agree to initiate it.

The setting maxNegotiatedRings sets the target number of negotiated rings. Once during each interval the peer checks the target number, and if it has not been reached yet then the peer will initiate a new ring. If a ring initiation request is received by the peer and the limit is reached then it is dropped, else the request is processed and then forwarded.

If a ring is choked then it will leave the active state and return to the negotiated state. If there are more negotiated rings than maxNegotiatedRings then rings will get choked until maxNegotiatedRings is reached.

Rings can leave the negotiated state by being promoted to active, by reaching the continuous idle time interval count specified by negotiatedRingsIdleTimeout, or when the ring is being removed altogether (for example because the file is completed).

All of these measures try to ensure that there is always a constant number of rings that are ready to be unchoked so that the incentives algorithm can work. There is a

delicate balance in this number. If it is too high then rings will never get unchoked and so the whole incentives mechanism will not work. If it is too low then there is little risk for a peer of a download getting choked and so the whole incentives mechanism will not work very well. Each peer may have a different cutoff point at what is still acceptable. Therefore, the maximum number of negotiated rings is a parameter of a peer's utility function.

Selection Functions

When following the ring initiation algorithm, there are three decisions to be made by peers. The algorithm used by these functions depends on the utility functions of the individual peers and might differ from one peer to another.

chooseRingToInitiate() The peer needs to chose a known ring that it would like to elevate to negotiated state. A simple implementation would choose the shortest known ring that passed both the **chooseWhetherToInitiateRing()** and the **chooseWhetherToNegotiateRing()** functions.

chooseWhetherToInitiateRing() This function is called when a new RingNegotiationMessage is encountered. As a minimum it should check that any of the two neighbouring peers of the associated ring are not part of an existing ring that is already being initiated and that the ring passes the **chooseWhetherToNegotiateRing()** function.

chooseWhetherToNegotiateRing() Finally this function is called when a known message has been received a second time. The peer now knows that all other peers of the ring agreed to set the ring to negotiated. The minimum checks for the peer are that it needs to make sure that none of the neighbours of the ring are part of a negotiated or an active ring yet and that the limit maxNegotiatedRings has not been reached yet.

4.12.3 Ring Selection

For each interval the peer has to decide which rings to allow downloads from (i.e., set to active) and which rings to choke (i.e., set to negotiated). The purpose of the ring selection algorithm is to choose the set of rings that are active during the following interval.

There is only a limited number of slots because fewer active rings means more bandwidth per ring, which means on average a better download speed in return. Also, fewer network flows create less overhead and leave more of the available bandwidth of the peer for the transmission of file pieces. But keeping more rings running safeguards against not using the bandwidth optimally because a collapsing ring has less of an impact. Using the number same number of rings per file as BitTorrent (five) seems like a reasonable starting point for the evaluation.

The **chooseRingsToActivate()** algorithm splits the available slots for active rings (five per file in BitTorrent) into two groups. One group is reserved for optimistic unchokes (one in five in BitTorrent) whereas the other group is reserved for the best performing rings.

For the latter group the peer takes those rings from the set of currently active and negotiated rings with the fastest download speed until all slots are filled up.

For the first group the peer then adds the *choke protected* rings to the list of rings to activate. These are the rings that have been optimistically unchoked recently and that require a chance to increase in speed.

Then it adds new optimistic unchokes to fill up the available slots for those. It randomly picks already negotiated rings that are not yet active and and starts the associated downloads. It adds those rings to the list of *choke protected* rings, where they remain for a few intervals (three in BitTorrent).

4.13 Summary

In this chapter we have introduced a system that implements the ideas that we have outlined in Chapter 3. We have looked at the difference of seeds and peers. We have explained how BitTorrent cuts files into pieces why we have adapted this model to our system. We defined services between peers and how they are adapted to the download of pieces in file sharing. Then we have investigated rings as seen from the perspective of individual peers. Finally we discussed the processes that are happening on the peers. First on a overview level and then in detail, focussing on the two core algorithms of ring detection and ring selection.

Chapter 5

Evaluation

After having laid out the principles and the algorithms for our incentives mechanism, the next step is to study its performance, both in absolute terms and in relation to a similar system. In this chapter we will first discuss the goal of the evaluation. We discuss evaluation methods and why we chose to perform a simulation. This is followed by a detailed discussion of the simulation that we have developed. Finally we will present the scenarios that we chose to simulate and the results of the simulation runs.

5.1 The Goal of the Evaluation

The first question that we would like to evaluate is whether the incentives mechanism that we have proposed works as intended. Does it indeed improve the download performance of peers if they also offer to upload files rather than only downloading them?

If that is the case then the second question is how the algorithm compares to existing file sharing systems. To answer this question we will compare it to BitTorrent as this is the most popular tit-for-tat incentives mechanism in use today.

The third and final question that we would like to address is that of the performance limitations of our mechanism. Up to which network size will it work reasonably well and when will it break?

We will now review each of these questions in more detail.

5.1.1 Fit for purpose?

File sharing systems, and it fact all peer-to-peer systems where the peers exchange some kind of service, work the better the more people participate. For this reason it is of crucial importance that peers have a strong incentive to join the network, as only

then the network will grow to a size that makes it efficient. The more peers have joined the network, the more interesting it becomes for other peers to join.

A peer will join the network when there is a sufficiently high probability that it will successfully finish downloading the files that it is interested in within a reasonable time frame. However, since the urgency and importance of files will be different from one peer to the next, and even from one file to another (i.e. different peers have different utility functions), we cannot know for sure what constitutes a reasonable time for individual peers. The best we can do is to consider the ratio of requested files that have been downloaded successfully after a period of time.

It is not enough that peers join the network but they also need to be incentivised to share their content. Therefore we need to show that peers fare better when they cooperate and provide resources themselves, as opposed to only passively receiving services from other peers.

5.1.2 Comparison with BitTorrent

To show how well our algorithm works we will compare it to BitTorrent, which is currently the most popular file sharing system, in a Gnutella scenario. This scenario contains both large and popular files, which BitTorrent is designed for, and small and/or rare files, which our algorithm is designed for.

Small Files

BitTorrent does not perform well when sharing small files. When files become too small, there will either be only one normal-sized piece or alternatively there will be multiple small pieces. The BitTorrent algorithm does not work if there is only one piece as there is nothing to exchange it for. But if there are many small pieces then that increases the overhead. Another problem for BitTorrent is that the smaller the file, the fewer peers are in the process of uploading and downloading it at any given time and therefore the fewer potential tit-for-tat partners exist.

Rare Files

BitTorrent does not perform well when sharing rare files either. If a file is rare then that means that there are few other peers downloading it at the same time. For very rare files there may even be no other peers at all that are downloading it. As with small files: if there are no pieces to exchange then the BitTorrent incentives mechanism does not

work. Additionally, if there are no seeds for a file—and no rational peer would seed a file—then, in fact, there need to be enough peers also downloading the file at the same time to share all the pieces of the file between them. Otherwise downloads will stall before finishing, waiting for more peers to appear that offer the missing pieces.

5.1.3 Performance Limitations

We also need to investigate at what point the algorithm breaks. We are mostly interested in the scalability our our algorithm. Does it work for large sets of peers as well as for small sets? The larger the network, the sparser populated the service overlay graph will be, under the assumption of a constant number of offered and requested files per peer. This makes rings less likely and so it remains to be shown how well our algorithm scales.

5.2 Choosing the Methodology

Next we need to choose the right toolset for this evaluation: The methodology to use for the evaluation and how to choose the right tools for it. After considering different options such as a theoretical analysis, experiments in the real world and a simulation, we decided to build and run a simulation for this purpose.

5.2.1 Evaluation Method

Following the advice by (Wohlin et al., 2000), there are three basic evaluation methods that could be applied to this case.

Theoretical analysis Developing a detailed model of the proposed system and then deriving conclusions on its performance from that model alone.

Experiment Implementing a real-life implementation of the system, distributing it amongst thousands of people, encouraging its use and then use monitoring, surveys or case studies to evaluate its performance.

Simulation Implementing a simulation environment and then running experiments within that simulation and monitoring the outcome.

Ideally we would be able to show analytically that our system works. Unfortunately, because of the scale and the complexity of the system, this seems impossible

to do. It would also not be clear how to model the problem in the first place. While there are ways to model the game theoretic aspect of the system on one hand as well as the network constraints such as limited bandwidth and delayed communication on the other hand, there is no clear way how to combine the two and still come to meaningful conclusions about the functioning of the system. Any theoretical model would also require a significant simplification of the system, which may cast doubts on its validity.

Another evaluation technique would be to build the system for real use and then to monitor it in action. The sheer scale of a peer-to-peer network however makes this an impossible task. (PlanetLab, 2010) would be too small and persuading thousands of people to run the system for testing purposes is not practical either. Additionally it would be very difficult to set up experiments that yield meaningful results in such a truly distributed setup with a lot of random interference. We would likely be limited to observation only.

Therefore, the most promising evaluation technique for our problem appears to be a simulation. A simulation allows us to set up precise conditions while not compromising on the complexity and scale of the system under evaluation. It does compromise however on the direct applicability of the results as the simulation will introduce a bias through its network performance approximation.

5.2.2 Choosing the Simulator

Finding the right simulator for our purpose is very important for performance reasons. We want to be able to simulate a large number of peers in order to achieve a realistic peer-to-peer network scenario.

We also need to simulate the network infrastructure between the peers. Classical network simulators operate directly on an IP packet level. As a result they are very precise but they significantly slow down the simulation. We do not really need this level of fidelity and would rather invest the computational resources in simulating more nodes. For our purposes a network simulation on the TCP flow level seems sufficient: There are flows between peers that have a certain speed that is essentially determined by the TCP congestion control algorithm, which tries to share the available bandwidth equally between flows.

Since we do not need a detailed model of the network on a packet level, popular

network simulators such as (ns-2, 2010) or Omnet++ (Varga and Hornig, 2008) are not really useful in our case. Instead, we need a lightweight simulation tool that knows of nodes and communication protocols between them. Ideally it is also designed with the peer-to-peer paradigm in mind. We have chosen (Peersim, 2010) as the simulation tool that fits our requirements best: It is written in Java, it is lightweight and fast and allows both cycle driven and event based simulations. The minor performance hit by using Java as opposed to a compiled language such as C++ is outweighed by shorter development times and lower probabilities of undetected errors in the code. A detailed explanation of Peersim will follow in Section 5.5.1.

5.3 The Simulation Model

The key elements to model are the network, the nodes and the incentives algorithms on the peers. For the peers we can directly use the models that we have developed in Chapter 4. The notable exception is the search functionality, which we have omitted in that chapter and that we assume to exist. During our simulations we will use a simple centralised database implementation for this purpose.

We are using the event-based simulation class of Peersim. An event-based simulation is the natural choice for our scenario because of the asynchronous nature of individual network nodes and because of the time it takes for messages between nodes to traverse the network.

The simulation works as an extension of the Peersim network model. There are a number of *nodes* where each node runs a set of *protocols* such as the file search protocol or the ring detection protocol. We will discuss these protocols in more detail in Section 5.5.3.

5.3.1 The Network Model

Because the simulation is concerned with the interaction of nodes and not with the effects of the network, we choose to model the network as a simple star topology. There is a single exchange in the centre and each node has a direct uplink and downlink to it. All network effects like delay, congestion and packet loss are ignored.

For each node, the simulation keeps track of its incoming and outgoing bandwidth, as well as the flows that exist to other nodes in the network. There is no bandwidth

limitation within the central exchange and so there is no bandwidth limitation between nodes other than their connection to the network.

In regular intervals the bandwidth use will be recalculated for all flows in the network. First, the bandwidth required for tokens is subtracted from the available bandwidth. Then a simple TCP approximation algorithm is used that is akin to the one used in the Narses simulator (Giuli and Baker, 2002). The actual flow bandwidth is the minimum of the outgoing bandwidth on the sender divided by the number of outgoing flows and the incoming bandwidth of the receiver divided by the number of incoming flows. We do not model TCP slow start and so a flow reaches its maximum speed immediately.

One main goal of the simulator design is to break its functionality up into components, so that they are re-usable for different simulation scenarios. The most obvious benefit of this strategy is that the fact that BitTorrent is a special case of our algorithm is directly reflected in the simulator code: with two exceptions it uses the same classes as our algorithm does. Those exceptions are that it does not use a ring finding algorithm and that it overloads some of the decision-making methods on which neighbours to upload to in the incentives algorithm class.

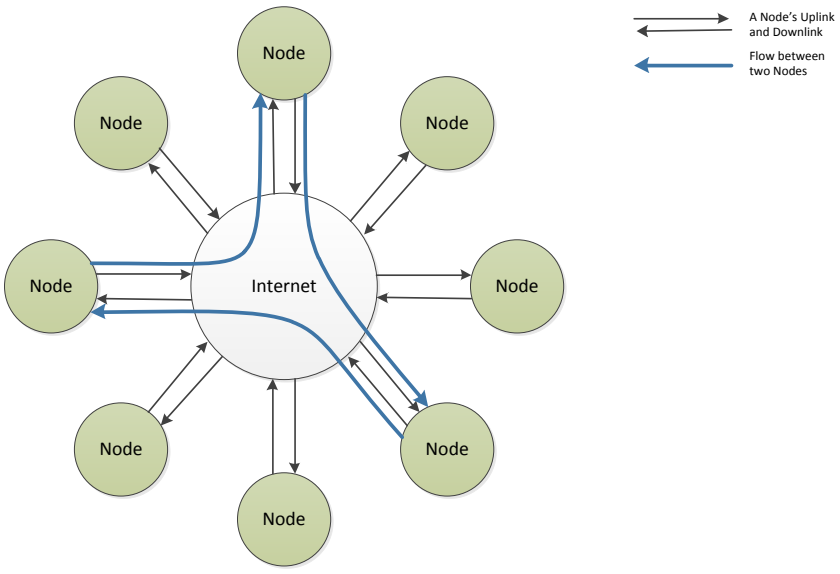

Figure 5.1: Network Model

See Figure 5.1 for a graphical representation of the model. The Internet is modelled as an unblocking network switch in centre. Each peer has an uplink and a down-

link to the Internet. Flows between peers are routed through the Internet and are limited in quality only by the limitations of the peer's connections. The implicit assumption is that there is no bandwidth limitation in the backbone of the Internet. Three flows between different peers that make up a ring are shown.

Implications of the Network Model

We feel that this strict simplification is necessary to be able to correctly model and evaluate the correct functioning of our algorithms in an environment that is completely controlled by the peers. Only when we understand its performance in such a setting can we proceed towards more realistic network models, or perhaps a real-world implementation.

This simplification comes at a price, naturally. There are two main limitations introduced by this model. The first limitation is that peers cannot distinguish problems in the network from non-cooperation of peers, which translates to a positive feedback loop, which in turn worsens their effect. The second limitation is that we assume potential synchronicity of peers due to the granularity of time and near-instant transmission of messages, meaning that we will not be able model the effects on the algorithms caused by these network characteristics at all. We do not consider this to be a major problem though due to the fact that peers will need to collect information for a considerable amount of time before they are able to act on it. This means that small network delays will likely not influence the decisions peers take to a large extent.

5.3.2 The Nodes

In addition to the network connection that consists of an incoming and an outgoing bandwidth per time unit, nodes are modelled as having an incentives strategy and a maximum number of concurrent services that they provide. These settings can be different for each node of the simulation.

Additionally a number of system-wide variables are set that are the same for all nodes. These include the maximum ring length to search for, the number of paths to remember to each other node, the generation frequency of RingSearchTokens, the ratio of optimistically unchoked rings and the various timeouts for idle rings and choked rings.

For each node a number of functions are modelled. Nodes have the ability to mes-

sage each other. They can provide services to each other, in our case file sharing services that consume bandwidth. Nodes can maintain a list of files that they are interested in, and they can search for them, i.e. find out which other nodes offer those files. Nodes can discover rings using a distributed algorithm. Furthermore nodes can apply an incentives mechanism to facilitate the goal of downloading the files that they are interested in, which manifests in their ability to choose which other nodes to upload to at any given time.

5.3.3 The Incentives Algorithms

We will simulate two different incentives algorithms: The classic BitTorrent algorithm with a slightly more aggressive choking and unchoking than described in the original paper—a modification that is in wide use—and our algorithm, which is a generalisation of it.

In the original BitTorrent paper a peer replaces the slowest receiving peer with the fastest providing peer that does not receive anything yet. In the more aggressive variant more than one peer will be exchanged if the incoming flow measurements indicate it. In other words, a peer chooses those neighbours for uploads that provide the best service, regardless of whether they already receive a service from the peer or not. If they happen to do so then provisioning of the service simply continues.

On the level of the incentives algorithm, there are only two decisions that differ between the BitTorrent algorithm and ours: The question of which ring to choose for initiation next and the question of which rings to activate during the next interval. Everything else is exactly the same. BitTorrent is limited to a certain number of two-leg rings per file whereas our algorithm is not. For the simulation this similarity means that one algorithm class can inherit most methods from the other, with mostly only the choice methods being overloaded.

5.4 Variables in the Simulation

In this section, we will discuss the input variables of the simulation. Some of the variables are simple scalars such as the number of nodes, whereas others are functions such as the file size distribution.

(Stutzbach et al., 2007) were kind enough to make some of their Gnutella trace

log files from 2005 available to us. These files contain a snapshot of the list of files and their attributes that the reachable peers of the Gnutella network offered at a certain point in time. From this raw data we were able to retrieve a number of key probability distributions for the simulation.

The most obvious variable from the trace is that of the actual size of the network. In those trace files the number of hosts that could be successfully queried is between 367,090 and 459,148. Between 99,727,354 and 101,430,198 unique files were offered, on average around 260 files per host. While for reasons of limited resources we cannot actually run a simulation with that number of nodes, it still gives us an idea of the scale and how far the simulations are off.

Distribution of Offered Files

Figure 5.2 shows the distribution of the number of files offered by the individual hosts that were queried. The graph only contains those peers that actually offered files. Of the 381281 peers in the snapshot, 56,611 offered no file at all, 7,017 offered one file, 5,147 two files and the highest number of files offered by a peer was 61,416.

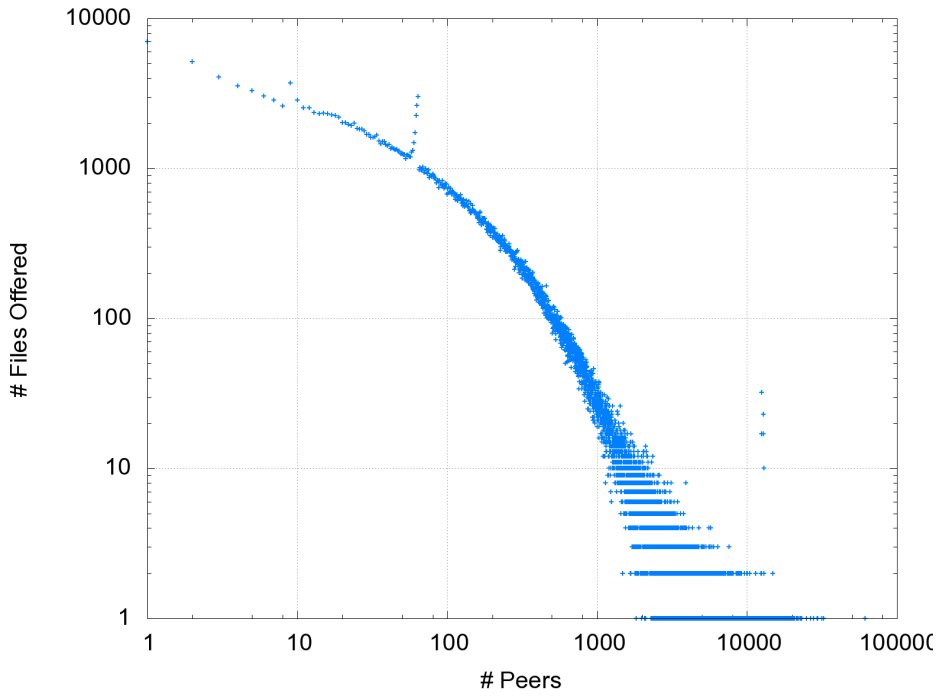

Figure 5.2: Number of Files Offered in Gnutella

Because there is no incentives mechanism in Gnutella, we can take and use the

distribution of the number of files that are made available by the peers as the lowest limit. The files offered in Gnutella are offered despite the fact that there is no advantage for peers to do so. If peers had an incentive to offer files then those numbers would be very likely much higher.

File Popularity Distribution

We can also derive a realistic file popularity distribution from this data. It shows how many copies of files exist within the network. It is immediately obvious that the vast number of files are offered by only one or by very few peers.

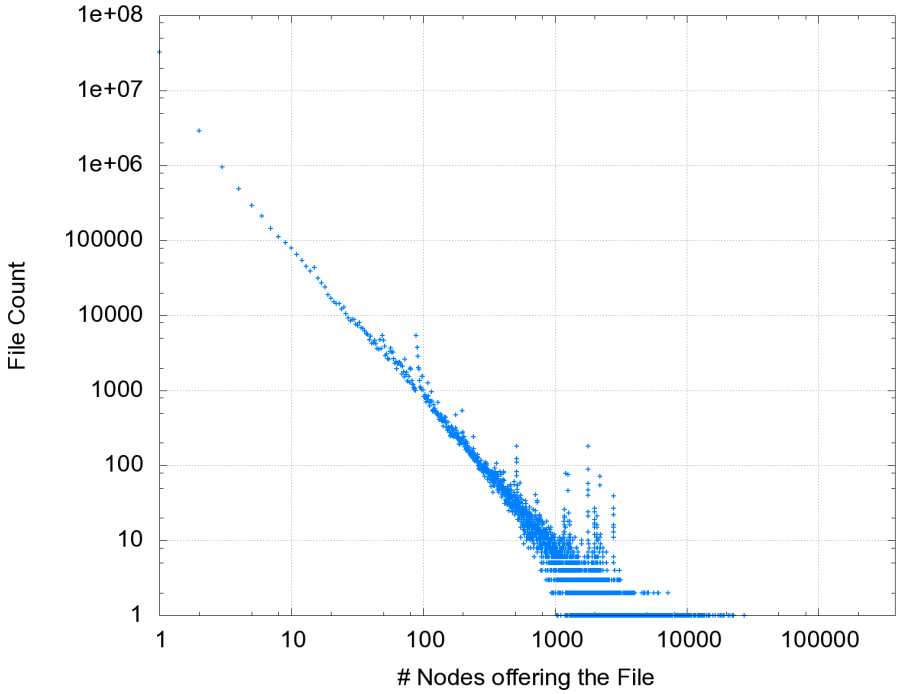

Figure 5.3: Popularity Distribution of Files in Gnutella

As we can see in the log-log graph in Figure 5.3, the file popularity distribution closely follows a power-law distribution with $\alpha \approx -2$. The most popular file was offered by 27,234 out of 381,281 peers, or about 7.14% of peers. 32,610,000 files were unique and were only offered by one of the peers. In total there were 38,684,039 unique files within the snapshot.

File Size Distribution

We can also derive a file size distribution from the data. This is shown in Figure 5.4. We will not try to interpret this rather chaotic distribution other than noting that some

of the peaks can likely be explained by the typical file sizes of songs, films and similar. It is important though that this graph shows that a significant number of small files are shared by the peers.

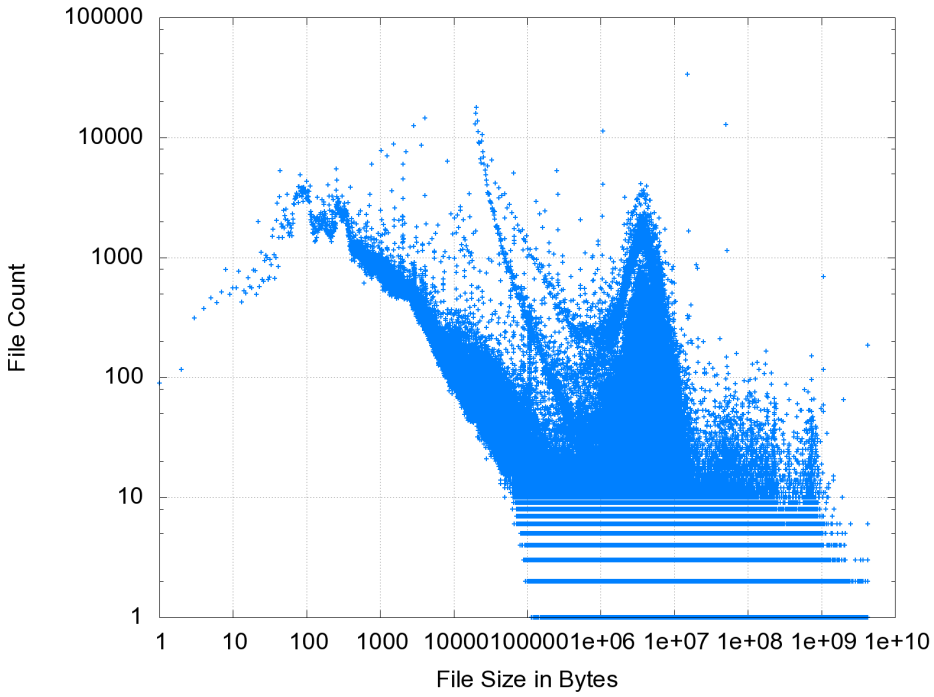

Figure 5.4: Size Distribution of Files in Gnutella

Both the file popularity as well as the file size distribution are likely independent of Gnutella particularities, as they are determined by the type of content that peers are interested in irrespective of the actual sharing mechanism.

Other Distributions

For one key distribution there is to our knowledge no real-world data available: the distribution of the file request probability amongst peers. We decided to use the most conservative assumption for this variable, which is that each peer is requesting only one file. We know for certain that each peer requests at least one file because if a peer did not then it would not be part of the peer-to-peer network (assuming that peers act rationally).

There are two key distributions that we need to modify between simulation runs: The strategy distribution among peers and the file request frequency. We model 'peers' and 'leechers'. 'peers' are cooperative nodes that try to improve their performance by

uploading pieces. 'leechers' are uncooperative nodes that never upload anything but only download.

The file request frequency is a variable that determines the frequency of new file requests in the network. As files finish downloading during the simulation, new requests are required to keep the total number of requested files in the network constant. Because each simulation will differ based on the other variables in how quickly files are downloaded, the file request frequency needs to be calibrated for each simulation scenario independently.

5.5 Simulation Implementation

In this section we will discuss the technical details of the simulation. We will introduce the simulator *Peersim* that we used, and explain how we have built a simulation of our incentives algorithm and of the BitTorrent incentives algorithm using it. We will also discuss the additional tools that we have created for generating distributions and *events files*, and for parsing the simulation output and creating graphs from it.

5.5.1 Overview

Peersim uses a model where there is a number of nodes, and protocols running on those nodes. There is one instance of each protocol on each node. In addition there is a system wide set of observer classes, typically one per protocol.

The Peersim simulation is configured using a configuration file. This file contains the basics such as the number of nodes, but also the protocols of each node and configuration parameters for those protocols. In addition it contains a set of observer classes. A protocol in Peersim is a class where each instance is associated with one node. Typically it is either called periodically or it responds to events. Often it communicates with another instance of the same protocol that is running on another peer. A typical example would be the *RingProtocol*, where instances communicate with each other in order to find rings. Protocols are explained in more detail in Section 5.5.3. An observer in Peersim is a class that is called in a fixed interval and that is usually used to observe the status of an aspect of the simulation by querying all instances of a protocol about their current state and then processing and printing that information for further processing after the simulation run. An example in our simulation would be the *Net-*

workObserver class, which regularly queries the *NetworkProtocol* instances about the amount of bandwidth used and which then prints an aggregate of all the bandwidth used within the simulated network.

At the beginning of the simulation, one instance of a node and of each protocol class is generated and initialised and then cloned. One instance of each observer class is also created and initialised. This means that we cannot configure individual nodes using Peersim's configuration file, but only those variables that are the same on all nodes. This is not enough because in our model different nodes offer and request different files, and they might have a different incoming and outgoing bandwidth. Our solution is to create an events file for each simulation with a per-node configuration and node specific events, and process it using an events file reader observer class.

Once these files are in place the simulator is started, it executes and eventually produces results, then prints them and quits. There is no interaction once the simulator is running. The output contains information about the finished download of pieces and aggregate status information of the nodes and the network. In the last step this output is then processed by a series of Perl and Shell scripts that generate graphs from it.

Because the seed for the pseudo-random number generators is specified in the configuration files, all simulations can be re-run with the exact same outcome. The simulator implementation is entirely deterministic.

Peersim Configuration File

The *Peersim* simulation tool uses a configuration file that is passed to it as a command line argument. These configuration files contain few general settings, for example the time the simulation ends and whether peers are to be shuffled, but they also contain the Java class names of the components and that way they also define all the *protocols* used in the simulation. In addition they contain configuration parameters for those protocols such as the maximum ring size or the search query interval. We will discuss the options in detail in Section 5.5.3.

Among the parameters in the Peersim configuration file is the location of the events file. This file contains all the events for the entire simulation run. The events include the initialisation of peers, as well as file requests and file offers by those peers. The events file itself is generated by an application that we called EventCreator, which takes a

number of variables and probability distributions and generates the events file based on them.

5.5.2 The Events File

The events file contains the simulation events over time. Each event is defined by the time that it happens, the node that it happens on and the type of the event. Depending on the event type it may also contain additional event type parameters. Events of time -1 are executed before the simulation starts.

File Format

Events files are organised as one command per line, with arguments following the command. Some of the information such as the file size and the piece size needs to be specified a number of times in different locations. While this increases the size of the events file and is a potential source for inconsistency, it allows for a faster parsing of the file and requires less memory. Since events files are always auto-generated, we feel this is a worthwhile trade-off.

```
# defines a three node ring
# format: time node command arguments
# initNode upBw downBw strategy
-1 0 initNode 262144 32768 std
-1 1 initNode 262144 32768 std
-1 2 initNode 262144 32768 leech
# offersFile filename size piecesize
0 0 offersFile fileA 2097152 262144
0 1 offersFile fileB 2097152 262144
0 2 offersFile fileC 2097152 262144
# requiresFile filename popularity size piecesize
1 0 requiresFile fileC 0.3333 2097152 262144
1 1 requiresFile fileA 0.3333 2097152 262144
1 2 requiresFile fileB 0.3333 2097152 262144
```

Figure 5.5: Sample Events File

Figure 5.5 shows a simple events file that describes a scenario with three nodes that could form a ring, based on the files they offer and request. However, one of the nodes is a freeloader as denoted by the leech strategy and so the ring would soon collapse.

The following event types exist:

initNode This event initialises a node. It sets its uplink and downlink bandwidth, and the strategy it uses.

offersFile This event causes a node to offer a particular file. This means that it announces to the tracker that it has all pieces of the file available. This is the state that a requested file will eventually reach when all pieces have been successfully downloaded. The parameters for this event are the name of the file, the file size and the piece size.

requiresFile This event causes a node to try to fetch all pieces of a new file. It announces to the tracker that it does not have any of the pieces but is interested in all of them.

dropFile This event causes the node to stop offering or requesting pieces of the file.

Creating Events Files

Events files are generated by a purpose-built tool called EventCreator. Its input consists of a number of variables such as the number of peers and a number of probability distributions for parameters like file popularity or the bandwidth of a node.

A design limitation of the EventCreator tool is that possible correlations between variables that were present in for example the original Gnutella trace files are ignored. For example, it might be the case that the file popularity distribution is different on peers that offer many files when compared to peers that offer few files.

One important decision when creating events files is to set the length of a simulation time unit to a sensible value. The coarser the time granularity of the simulation, the quicker its execution. However, setting it too coarse will introduce errors because nothing can happen in a time span shorter than one time unit. This is especially relevant for the time it takes for messages to travel between peers. For our case we settled on a time unit of 125ms, which allows communication even in rings of length 7 well within one interval of the incentives mechanism.

Random seed The seed for for the pseudo random number generator.

Files offered distribution A probability distribution of the number of files offered by a peer.

All requests at beginning flag A boolean value of whether requiresFile events should occur during he whole duration of the simulation based on the file request frequency distribution, or only at the beginning of the simulation.

Files requested distribution A probability distribution of the number of files requested by a peer.

File request frequency distribution A probability distribution of the likeliness of a new file request for a node in any one time unit.

File popularity distribution A probability distribution of the popularity of files, i.e. the percentage of peers that offer a file.

File size distribution A probability distribution of the size of files.

Peer strategy distribution A probability distribution of the strategies of peers. For example, 90% of peers could be co-operative (”std”) while 10% might be freeloaders (”leech”).

Incoming and outgoing bandwidth distributions A probability distribution of the incoming and outgoing bandwidths of peers (as made available to the peer-to-peer software).

Incoming and outgoing bandwidth distributions for seeds A probability distribution of the incoming and outgoing bandwidths of seeds.

Time limit the number of time units that events should be generated for. Determined by the planned length of the simulation as no events will happen after it.

Peers per seed The number of peers that need to be interested in a file for a seeding node to be created. If less peers are interested in a file then no seed will be available. More interested peers will lead to more available seeds.

Initialisation period The node initialisation phase at the beginning of the simulation is drawn out over this many time units.

Number of peers The number of peers in the simulation.

Piece size The piece size for all files.

5.5.3 Peersim Protocols

Each node in the simulation implements a number of Peersim protocols. Those are the Java classes of Peersim for nodes to communicate with each other. The classes are called NetworkProtocol, MessageProtocol, ServiceProtocol, SearchProtocol, RingProtocol, and IncentivesProtocol.

Some of these protocols have an observer class associated with them. The observers are called NetworkObserver, SearchObserver and RingObserver.

We will now explain the function and the configuration options of all of these classes in detail.

NetworkProtocol

The NetworkProtocol class deals with the data flows between nodes. On each node it keeps a list of incoming and outgoing flows, as well as the bandwidth each of them currently consumes. In regular intervals it recalculates the bandwidth distribution of the flows. One special flow for each node are the messages to other nodes. When distributing the bandwidth it takes precedence before all other flows. This allows us to model the effect of messaging on the bandwidth of flows without making the simulation overly complicated by simulating communication problems between nodes.

MessageProtocol

The MessageProtocol class provides a messaging mechanism for nodes. Nodes can communicate using messages, which then use up some of the network bandwidth. The messages that are exchanged between nodes deal with communicating interest in a file, with exchanging information about the file pieces that are required or available on nodes, and with exchanging ring token messages for the ring detection algorithm.

The only configuration option is the *message delay*, i.e. how long it takes for a message to get from one node to another.

ServiceProtocol

The ServiceProtocol class maintains a list of provided and received services on a node. The only service implemented in the simulation is that of uploading pieces of a file. This type of service communicates with the NetworkProtocol class to register its bandwidth needs.

SearchProtocol

The SearchProtocol class is responsible for finding other peers that are also interested in the files that its node is interested in. It does so by issuing search queries to the SearchObserver class, which hosts the simulated equivalent of a *tracker*. In addition the class keeps track of which nodes offer and require which pieces of the files (including its own node).

The configuration option for this protocol is the *search frequency*, i.e. how often it searches for new nodes that are interested in the same files.

RingProtocol

The *RingProtocol* class deals with finding the rings that are then used by the incentives protocol. The basic version only finds rings of length 2, as required by the BitTorrent protocol implementation. This can be done by simply comparing potentially providing and potentially receiving peers.

Our algorithm on the other hand requires a much more sophisticated ring detection mechanism that is able to find much longer rings, as explained in Chapter 4. This is done in the derived class DoubleRingProtocol, where we implement the ring token mechanism that was detailed in Chapter 3. The prefix 'double' was chosen early during development because tokens can travel in both directions, whereas before they would only travel one way.

The basic version has one configuration option only: Whether to look for rings that comprise of different files. This is true for our algorithm but it is not true for the BitTorrent algorithm where only different pieces of the same file can be exchanged.

The DoubleRingProtocol has additional configuration options. The *maximum ring length* to search for can be set (this set the TTL of RingSearchTokens), as well as how many paths to remember to every other known node, but also the ratios of creating and dropping RingSearchTokens and for how long to wait for a RingFoundToken to return.

IncentivesProtocol

The last of the protocol classes to be detailed, called IncentivesProtocol, is implementing the ultimately most important aspect of the node: the incentives mechanism. In effect this means that this class decides when to initialise rings in order to use them, whether to forward incoming ring initalisation requests, and whether or not to actually

start and stop services (file uploads) to other nodes.

IncentivesProtocol itself is an abstract class, with FullIncentivesProtocol which implements our algorithm being derived from it, and BitTorrentIncentivesProtocol in turn being derived from FullIncentivesProtocol. Besides the initialisation and other house keeping, only two methods are overloaded in the latter class: chooseRingToInitiate() and chooseRingsToActivate().

The configuration options of FullIncentivesProtocol set the maximum number of negotiated and active rings, the ratio of rings that have been optimistically unchoked, how long those are protected from being choked again, the period how long a choked node is not considered for unchoking, and various timeouts for tokens and idle rings. Additionally BitTorrentIncentivesProtocol requires to configure how many files can be active at once and how many active rings per file are allowed.

Observers

In addition to these protocols there are also three observer classes, each of which is associated with one protocol. The most important class is the SearchObserver, as it parses the events file and also contains the search facility and the file directory in the simulation. As such, one configuration option requires to set the name of the events file and another option sets the maximum number of search results to return for any query.

The two other observer classes are the NetworkObserver and the RingObserver, both of which are creating simulation output and both of which have no configuration other than their invocation interval.

5.5.4 Simulator Output

The simulator creates a number of different types of output. Many different kinds of telemetry can be printed about the current state of the nodes, how many rings are in which state, how many rings nodes know about, the load of the network and similar.

A sample of the two essential types of output lines is shown in Figure 5.6. The first line notes when a node becomes interested in a new file and the second line notes when a piece of a file has been successfully downloaded, and whether that piece was seeded by a seeding node or whether it was downloaded as part of a ring. Both lines also contain further information about the nodes and the file, to be used by the post-processing scripts.

```
# node interested in new file
node 281 strategy leech interested in f11215 size 2406537\
popularity 0.006000 at time 112

# finished pieces
download finished. piece 95 of file f1353814 size 37947832\
piecesize 262144 from 2812 to 9885 start 15600 stop 15680\
speed 3276 duration 80 popularity 0.000200 receiver std ringsize 4
```

Figure 5.6: Sample Simulator Output

5.6 Simulation Scenarios and Results

In this section we will discuss the actual simulations that we have run. First we will review the issue of seeds in the simulations. Then we will explain the general simulation settings that were used. Finally we will review the concrete simulations and present and discuss the results.

Most of the basic settings are the same in all simulation scenarios. This will make the results of the different incentives mechanisms as comparable as possible. After all, the peers will remain the same irrespective of the overlay network in all but one aspects: when there is a reason to share then more peers will share. Therefore the main parameter that will differ is which files the nodes offer and request.

5.6.1 Seeding Dilemma

Most of the simulation scenarios include seeding nodes for some of the files in the system. This is necessary for BitTorrent to work at all, as there is no incentive for peers that already have a complete file to offer pieces of it to other peers. If there is no seed then the downloads will never start.

Instead, there need to be dedicated seeding nodes that provide uploads for no benefit within the system. We assume that there is some kind of external motivation for the seeds, or that they simply do not act rationally in a game theoretic sense. This behaviour can often be observed in real-world BitTorrent, where peers keep seeding a file for a while when they have finished downloading it. This is often simply because many BitTorrent implementations do not stop uploading once the download has finished and the owner forgot to terminate the download. It is unlike the peers in our model, which

always act completely rationally and so they would not upload if there was nothing they wanted to download.

Seed Influence

The problem is that seeds skew the results of simulations because with seeds not all of the network traffic is incentives driven. The main quality metric for the simulations will be the percentage of completed requested files, and that does not consider whether individual pieces of those files came from seeds or from other peers. Most likely it is a combination of both. However, this is not a problem when we compare the two approaches in the same setting, as the same amount of seeding will be performed in both simulation runs. To combat the influence of seeds on the results we have throttled down the bandwidth of seeds so that they provide the data if required without overshadowing peer-to-peer downloads.

Seeds and Rare Files

The rarer a file, the more impact the presence of a seed has on the download performance of that file. The dilemma is that we cannot expect seeds to exist in the real world network, but without seeds BitTorrent does not work. Rarer files are less likely to have a seed. Therefore, to allow us both to show that our algorithm outperforms BitTorrent for rare files where there is a seed, and also on rare files where there is no seed, we have set an arbitrary seed cutoff point based on popularity. For files with a popularity below the cutoff point there will be no seed in the simulation.

Seeds and Small Files

In BitTorrent there is no point in seeding files that are smaller than one piece, because once the piece has been downloaded, the whole file has been downloaded. This means that BitTorrent cannot incentivise their download at all, irrespective of the actual size of pieces. Even the smallest files needs to be cut into at least two pieces to make the algorithm work at all. With our incentives algorithm however, two small files can be exchanged for each other. There is no need to cut them up into tiny pieces.

If small files were seeded in the simulation then our algorithm would have to compete with seeds and that would tell us nothing. It is much more relevant to see how many small files can be downloaded successfully with our algorithm while none of them can be downloaded with the BitTorrent algorithm. The actual piece size that

we use is set somewhat arbitrarily, but that does not matter much since this dilemma is independent of the piece size.

5.6.2 Scenario Configuration Parameters

In this section we will describe the actual parameters of the simulation and what they influence. Some of these are used to generate the *events file* while others are parameters in the *simulation configuration file*.

We will put them in two groups. One group concerns the properties of nodes whereas the other group is about the files, their distribution and the distribution of interest in them. First up are the settings that predominantly concern the nodes and more technical aspects of the simulation:

Random seed The seed value for the pseudo-random number generator that generates the seeds for all the pseudo-random number generators in the simulation (RngPack, 2010).

Number of nodes This is the number of nodes in the network. All nodes will be part of the network for the whole simulation, even though they may not offer or request files at certain times. This is equivalent to leaving the network when no rings can be formed.

Bandwidth probability distribution How the incoming and outgoing bandwidth properties are distributed among nodes. Some nodes might have a faster connection than others.

Node strategy distribution Different nodes may have different strategies that they follow. There are three strategies that have been modelled for this thesis: *peers*, *seeds* and *leechers*. Leechers are freeloading peers that do not ever contribute.

Time limit The duration of one simulation run. To achieve a reasonable granularity with respect to the network simulation we are using a granularity of 125 ms.

Number of concurrently provided services on a node The maximum number of services a node provides simultaneously. This is five per file for BitTorrent.

Initalisation period The time period at the beginning of the simulation during which the peers get entered into the system, to minimize side effects of adding them all at once. New file requests only start after this period.

Piece size The size of file pieces. We assume one size for all files. This size is a trade-off between the overhead a piece generates versus the likeliness to download a complete piece within a session without the ring getting choked. 256 kB is a sensible size that is also popular in real-world BitTorrent.

Following are the settings regarding the files and their distribution. Most of these settings directly affect the service overlay graph and so the existence of rings.

File size probability distribution The size distribution of the files offered and requested by nodes. This should be independent of the actual network, except for very large files that cannot be downloaded within the limited time that passes within one run of the simulation.

File popularity probability distribution The probability distribution of the popularity of files that are being offered. This depends on both the number of nodes as well as the *Number of offered files probability distribution*. The popularity distribution for requested files will be set differently for different simulations.

Number of offered files probability distribution This distribution defines how many files are offered by nodes at the beginning of the simulation. The actual files are then chosen using the previously described file popularity probability distribution.

Number of requested files probability distribution This is the probability distribution that is used to set the request situation per peer at the beginning of the simulation. It is also used to choose a peer for subsequent file requests during the course of a simulation run. Each peer gets assigned one randomly chosen probability at the beginning and keeps it for the whole time of the simulation.

File request frequency probability distribution This is the request probability distribution. It is used to determine the number of new file requests per simulation

time unit. The purpose of this variable is mainly to keep the total number of requested files in the simulation at a constant level. Therefore this variable needs to be close to the frequency of finished downloads. As a consequence of chance, peers may not request any file at some point in the simulation.

Number of peers per seed How many peers offering a file it takes for a seed for that file to be generated. This determines the popularity cutoff point, under which there are no seeds available. The popularity cutoff point was discussed in the previous Section 5.6.1.

Common Settings

The following settings will remain constant throughout all simulation runs.

Bandwidth probability distribution The bandwidth is 2 MBit/s symmetric for all peers and 128 kBit/s symmetric for seeds.

Time limit 116,400 units, where one unit is 125ms. This consists of slightly over one hour (30,000 units) for startup plus three hours of simulated time.

Number of concurrently provided services on a node Five concurrent services provided per node, the same as BitTorrent.

Initalisation period Nodes are introduced within the first 1,000 time units, but no measurements are taken until time unit 30,000.

Piece size a uniform 256 kByte for all files.

Maximum Ring Size the maximum ring size is set to 7.

Number of offered files probability distribution This distribution is taken directly from the Gnutella trace data as shown in Section 5.4.

Number of requested files probability distribution Each peer requests one file.

File popularity probability distribution This distribution is also taken directly from the Gnutella trace data.

File size probability distribution Also taken from Gnutella. Very large files cannot be successfully downloaded within the time frame of the simulation, but those are comparatively rare.

File request frequency probability distribution There are enough new requests during the simulation to replace the finished downloads, so that there is on average one request per peer in the system.

5.6.3 Gnutella Scenario

The goal of this simulation is to show that our algorithm outperforms the BitTorrent algorithm for small and rare files. For this we create a Gnutella-like scenario by using the real Gnutella data mentioned in Section 5.4. A new scenario based on those distributions is generated for every simulation run. More specifically the following variables are set:

Number of nodes 1,000 nodes.

Node strategy distribution All nodes are cooperative.

Number of peers per seed One seed is generated for every five peers requesting the file.

The output of the simulation results in three interesting graphs. The first graph shows for both algorithms the number of file downloads that have finished over time. The second graphs shows the ratio of requested rare files that having finished downloading and the third graph shows the same with respect to the file size.

All Files

The first question is whether our algorithm manages to download more files than BitTorrent. Therefore the metric is simply the number of files that have been downloaded successfully. A file is considered downloaded when all its pieces have been successfully downloaded.

Figure 5.7 shows the results. The x axis outlines the simulation time and the y axis outlines the number of downloaded files. Our algorithm manages to download significantly more files than our BitTorrent implementation does. The algorithms seem to settle at more than a threefold increase of downloads when our algorithm is used.

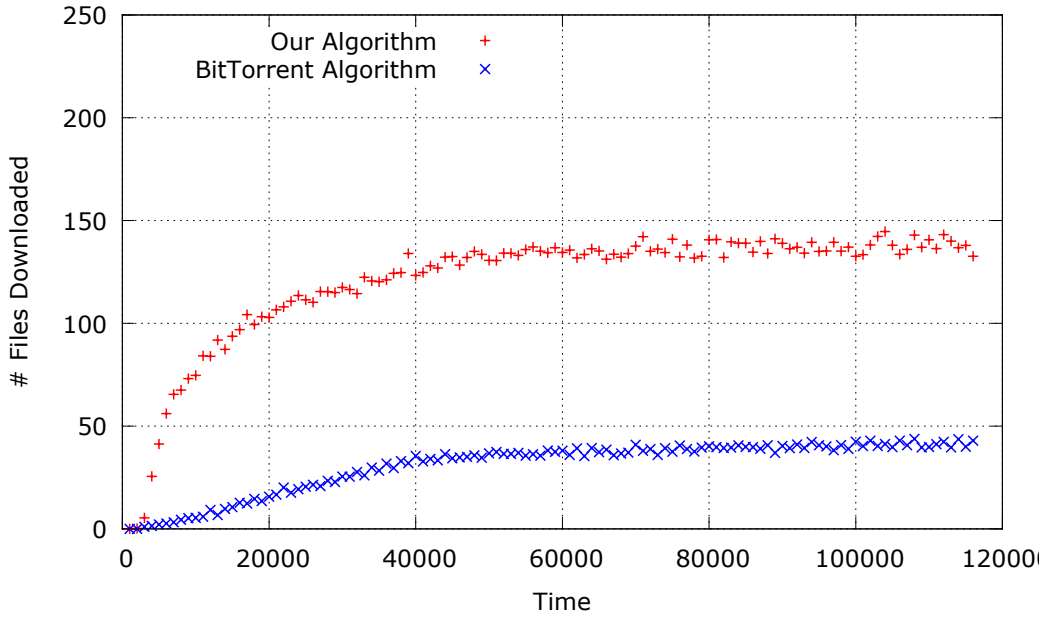

Figure 5.7: Gnutella Scenario: Finished Files over Time

This confirms our hypothesis that far more files can be downloaded when our algorithm is applied, when compared to BitTorrent alone. We can also say that a saturation point is quickly reached for both, after which the average file download rate is static.

Rare Files

Next we will compare the performance of the algorithms for rare files, that is files that are offered by few peers or very often only one peer.

The metric we use to judge our algorithm is the ratio of successfully downloaded files over the requested ones. We break this down either by file size or by file popularity. An alternative metric would be to consider the average time between requesting a file and finishing the download, but we think that this metric would be misleading because the prime objective is to acquire the file at all, not to acquire it particularly quickly. In many cases waiting for a download to start takes longer than the actual download itself.

The results are shown in Figure 5.8. The x axis of this graph outlines the popularity of files on a logarithmic scale where a value of 1 would mean that every peer has the file. The y axis outlines the system wide ratio of requested files of that popularity that have been successfully downloaded during the course of the simulation. A value of 1 would mean that all requested files of that popularity on all peers have finished downloading. At 0.005 is the *seed cutoff point* (see Section 5.6.1), i.e. the lowest file popularity for

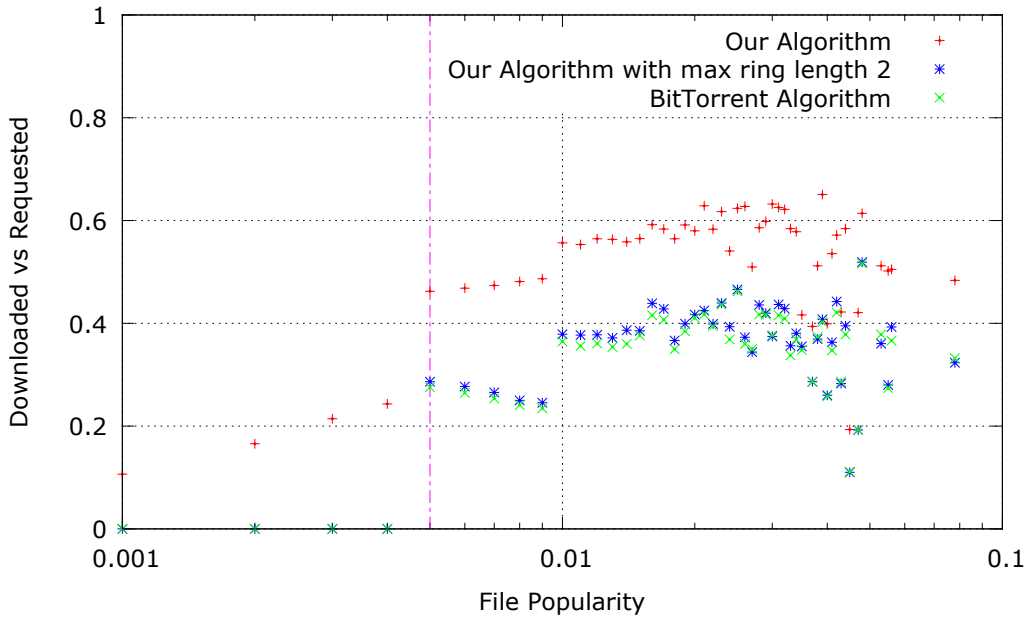

Figure 5.8: Gnutella: Ratio of Finished Rare Files

which files still have a seed. This is signified by a vertical line in the graph.

The graph contains three distributions of the ratio of finished downloads for files of different popularities. The rarest files are shared by only one peer out of the 1000 peers in the network, and the most popular file is shared by 8% of peers. Our algorithm performs better in this scenario for files of all popularities.

Our algorithm is able to successfully download files for which there are no seeds (popularity 0.005 and below). Unless there is a seed, BitTorrent is not able to incentivise the upload of files that are so rare that there is only one peer downloading them, but this graph shows that our algorithm is capable of achieving that. In addition, rare files that do have a seed—those around the 0.01 popularity mark—show a much better performance with our algorithm.

The graph confirms the findings of the previous graph in that more than three times as many files are successfully downloaded, and it shows that this difference in numbers is due to rare files. Between 20% to 30% of the requested files that are so rare that there is no seed were downloaded successfully.

The third curve in this graph shows our algorithm with a limit on the ring length of two. This effectively means that only the ”pieces across files” aspect of our algorithm gets used. The curve is very close to the BitTorrent curve. This shows that a maximum

ring length of two is not sufficient to improve the download performance of rare files significantly. In other words, tit-for-tat *rings* are essential for our algorithm to work, it is not enough to extend the BitTorrent tit-for-tat algorithm to use pieces of different files.

Small Files

The final graph that results from this simulation in Figure 5.9 shows how well files of different sizes fare. The x axis of this graph outlines the file size on a logarithmic scale while the y axis again outlines the ratio of completed vs. requested files. Again a value of 1 would mean that all files of that size interval have been downloaded successfully.

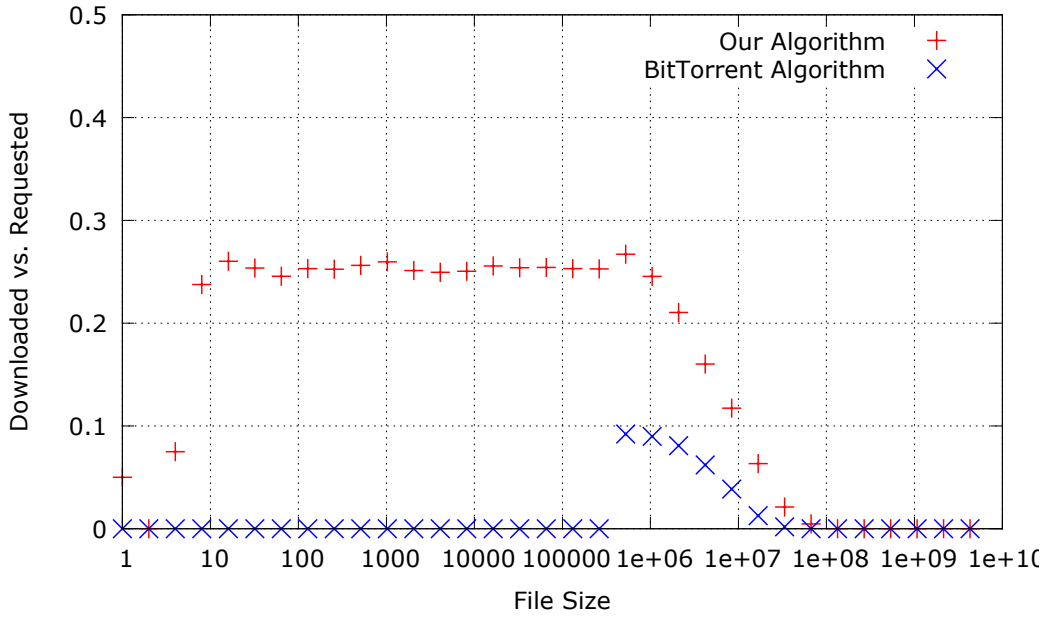

Figure 5.9: Gnutella Scenario: Ratio of Finished Small Files

This graph confirms the results of the previous two graphs in that consistently more files are being downloaded using our algorithm. The BitTorrent algorithm cannot incentivise the download of files that are smaller than one piece, which in this case is 262,144 bytes in size. In our algorithm however, two small files consisting of only one piece can be exchanged for one another. This becomes apparent in the graph, where the download performance is largely independent of file size. There is a sharp drop for both curves for large files, but that is due to the limited simulation time. The larger the file, the less likely it is to finish within that time limit.

The key result shown in this graph is that our algorithm is successful in incentivising the sharing of small files.

Ring Sizes

To round this experiment up we will analyse the ring sizes used in this simulation in Table 5.1. It contains for our algorithm the distribution of ring sizes among pieces that were successfully downloaded from other peers and not from seeds.

ring size	2	3	4	5	6	7
percentage	4.36%	39.05%	33.06%	14.63%	4.09%	0.43%

Table 5.1: Gnutella Scenario: Ring Length Distribution for Downloaded Pieces

The table shows that many more long rings than short rings have been found by our ring detection algorithm and used by our incentives algorithm, while the BitTorrent algorithm can only make use of a fraction of the rings—those of length two.

5.6.4 Effectiveness of the Incentives Mechanism

In this section we will discuss whether the tit-for-tat properties of our incentives mechanism work. We will show the effectiveness of the tit-for-tat algorithm with a scenario that is similar to the Gnutella one except that there are no seeds. Pieces of files can only be downloaded from other peers. Furthermore, half of the peers are now non-cooperative. They will go through all the same motions as the cooperative peers with respect to detecting and negotiating rings, but they will never actually start an upload.

To summarise the configuration parameters:

Number of nodes 1,000 nodes.

Node strategy distribution 50% of nodes are cooperative and 50% are non-cooperative.

Number of peers per seed There are no seeds.

Figure 5.10 shows the results. The x axis outlines the time and the y axis outlines the accumulated number of pieces downloaded up to that point. As we can see cooperative peers are able to download many more pieces than non-cooperative peers. This shows that it is hugely beneficial for peers to offer services. As with BitTorrent,

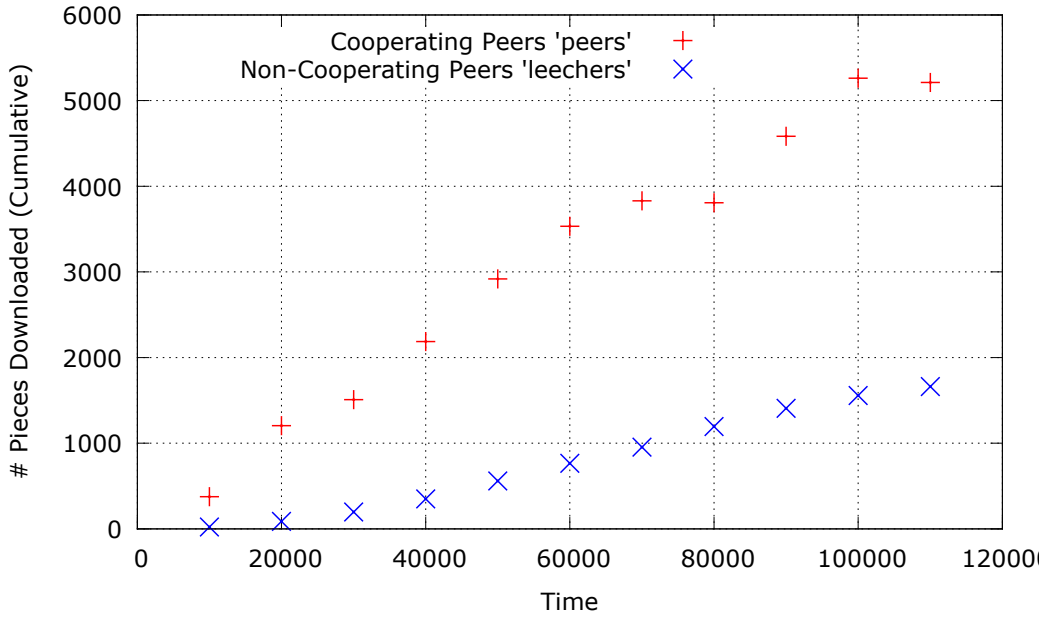

Figure 5.10: Cooperation Scenario: Downloaded Pieces over Time (Cumulative)

due to the optimistic unchoke mechanism, peers that never upload anything will also eventually succeed in downloading files, but downloads are much quicker if peers do cooperate by uploading pieces themselves.

In order to facilitate the tit-for-tat properties of our algorithm we had to shorten the optimistic unchoke grace period in the simulation. Using the BitTorrent settings of 30 seconds the peers would have been too kind to non-cooperative peers in combination with our simple TCP flow model where there is no spooling up. The exact timing is something that would need to be calibrated in a real-world implementation. We see no reason why this would differ from BitTorrent however.

5.6.5 Performance Limitations

In order to better understand how well our algorithm scales with respect to network size, we ran the Gnutella simulation again in larger networks of 5,000 and 10,000 peers. In order to avoid a saturation effect, the simulation only samples a one hour period instead of the previous three hour period. This results in a lower download ratio. Again, a new events file was generated for each simulation run.

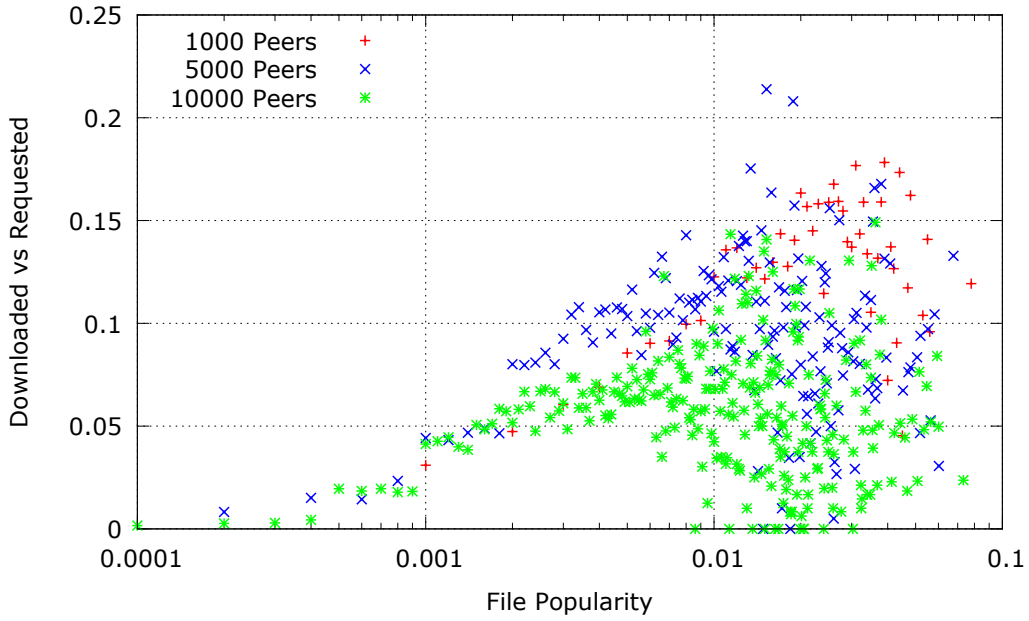

Figure 5.11: Network Size Comparison: File Popularity

Rare Files

Figure 5.11 shows the file popularity distribution graph. The x and y axes again outline the popularity of files and the ratio of successfully downloaded files, similar to Figure 5.8.

We can see that for all three network sizes the distributions are very similar indeed. With our algorithm, the probability of successfully downloading a file scales well with the network size and depends almost entirely on its rarity.

Small Files

Figure 5.12 shows the file size graph. Again, similar to Figure 5.9, the x axis of this graph outlines the file size on a logarithmic scale while the y axis outlines the ratio of completed vs. requested files. We can see that our algorithm scales well for small files with respect to the network size.

As before the file size does not seem to be an important factor on the probability of downloading a file for our algorithm, as all three distributions show almost a straight line. The drop off that starts around the one megabyte mark is again due to the limited time frame of the simulation.

This graph shows however that the overall probability of downloading files is much lower in larger networks. While about one in three requested files in the small

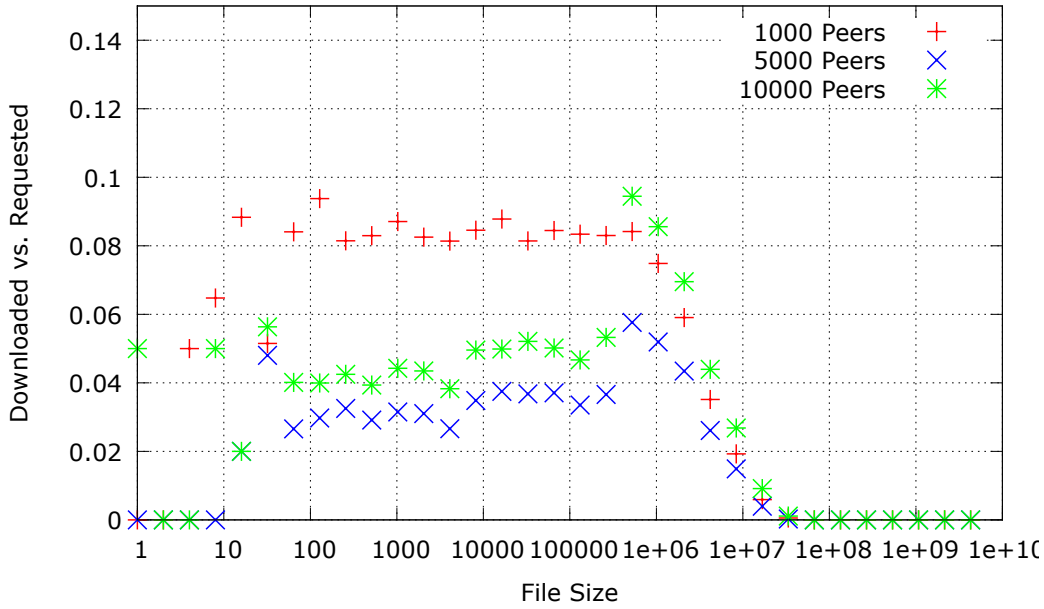

Figure 5.12: Network Size Comparison: File Size

network were successfully downloaded, that value drops to about one in fifty for the large network. As the previous graph showed, our algorithm works worse for rarer files. The reason for this is the large number of very rare files in larger networks. Most of the files that are offered in Gnutella are, in fact, unique. When most of the requested files are very rare then fewer of them complete their download.

This highlights one implicit assumption that we have made throughout the evaluation: only files that are offered by one of the peers in the simulation get requested. Even though 232,959 unique files are offered in the simulation with 1,000 nodes alone, these are very few when compared to the 100,000,000 or more different files offered in the actual Gnutella network.

However, the rarity of a file alone cannot be the problem by itself. After all, a file of which there is just one copy in a 1,000 node network downloads much faster than a file with only one copy does in a 10,000 node network. This means that the increased sparsity of the service overlay graph must play a crucial role: With more nodes and rarer files, the probability of edges in the service overlay graph simply drops under the threshold where enough rings exist.

The solution to this problem seems to be the semantic clustering of peers as proposed by (Handurukande et al., 2004). When peers are grouped based on their interests

then a sufficient number of rings should exist. There is no reason to assume that an individual peer cannot be part of more than one of those kinds of swarms, so these swarms could effectively be considered as areas of high density in the service overlay graph.

If we assume that the file popularity distribution in any such swarm is always close to a power-law distribution then our algorithm will work, as long as peers can be reasonably certain that at least one peer in the swarm actually has the file that they are looking for.

5.7 Summary

In this chapter we have presented the evaluation of our thesis. We first stated the goals of the evaluation. Is the system fit for purpose? How does it compare to other solutions? What are its limitations? Then we argued why to perform a simulation instead of using one of the other available evaluation methodologies. After justifying our choice of a simulation framework we presented the model of our simulation, which is heavily based on our system design in the previous Chapter 4. Once we understood the variables in the simulation we then explained implementation details. In the final and most important section of the chapter we presented the actual simulations that took place and discussed their results: Our algorithm outperforms BitTorrent for small and rare files, it provides an incentive to share files, but it requires some kind of clustering to work well with very rare files in large networks.

Chapter 6

Conclusions and Future Work

6.1 Actual Contribution

In this thesis we have used rings in the service overlay graph to incentivise the sharing of rare and of small files in peer-to-peer networks. We have analysed the underlying problem of cooperation using game theory and realised that instead of playing a game against another peer, a peer might play against a ring instead, opening up more possibilities for sharing. We have designed distributed ring detection and ring management algorithms to discover and manage rings.

We have designed a peer-to-peer system that contains those algorithms and all the other components that constitute a peer. This included as the largest contribution the design of message-based communication system for use by the peers, but also contains stubs for network monitoring and service provisioning.

In the evaluation we have outlined a number of simulation scenarios to show that the approach is feasible, and we have implemented the system with a simulation framework to test those scenarios. We have added a simulation of the BitTorrent algorithm—which is, in fact, a specialisation of our own algorithm.

We managed to show that the algorithm works as expected, that it outperforms BitTorrent and that it works sufficiently well for one class of peer-to-peer networks. We were also able to show that the algorithm provides an incentive to share.

6.2 Conclusions

The good news is that we were able to achieve what we set out to do. The bad news is that our solution appears to require a fairly narrow set of conditions to work well;

maybe more so than we would have liked. However, this was a very long road that took us here with plenty of dead ends, so any positive result is good.

Quite a few others have taken up the idea of considering rings in the service overlay graph since we first published the idea in (Ackemann et al., 2002), but none of them have found a perfect solution. Finding out that the approach works in relatively small networks is maybe more than we could have hoped for.

It is notable that although there are a great many papers being published on peer-to-peer networks, both from a game-theoretic side as well as a more practical side, very few suggest radically new networks and many seem to settle on incremental improvements to BitTorrent.

Our system could be implemented for real relatively easily. It could be written by extending an existing open source BitTorrent client to contain the ring detection and ring management modules. The search infrastructure already exists and any semantic clustering is perhaps best achieved by non-technical means such as online communities for special interests. The system could be marketed as an incremental new version with improved sharing features that still support normal torrents as well, much like the DHT extension to BitTorrent systems that emerged in the last few years.

One example for these communities is in the area of user generated content such as for flight simulation games: Many web sites with collections of freeware extensions for flight simulator software exist, but over the years they tend to come and go, often taking their content with them. Then people start asking in forums whether anybody still has some of that content. Sometimes they are lucky and sometimes they are not. With our system, the other enthusiasts would have a rational reason for sharing those files, improving the chances for finding and being able to download them significantly.

On a technical note, it remains to be seen how much of a constraint bandwidth really is, but there will always be bandwidth constrained niche environments such as low power mobile devices. In any case, maybe a bigger problem than bandwidth is to incentivise peers to actually share all the data that they have accumulated. Even the fastest network is of little use when peers do not consider sharing most of their files.

6.3 Future Work

The immediate future work would be to explore the algorithms in more depth. Many of the parameters of peers have not been fully explored and have been set to conservative values, as that was sufficient to demonstrate that our approach works. Given a lot of time and computing power these settings could be optimised and this would likely make a large difference in performance. However, there are always limits to what a simulation can tell.

With regards to the algorithm itself, the next step would be to consider one of the many BitTorrent optimisations and extensions that have been published in recent years for our system. The basic game is still the same as with BitTorrent, except that the pieces can come from different files and one plays against a ring, not a peer, so there is some more delay in the reaction of the opponent. Any extensions to the core tit-for-tat mechanism in BitTorrent that are not delay-sensitive may work.

The question of scalability is another avenue to pursue. Perhaps there are ways to overcome the sparsity of larger graphs, for example by introducing some kind of clustering algorithm. It would probably be enough to have dense regions in the service overlay graph. Maybe the search functionality could be tied in with the overlay, so that it is more likely to get search results from peers that are close in the service overlay graph instead of getting a random sample of peers that have the file. This is obviously especially true for comparatively popular files, where the search functionality has to truncate in the reply the list of peers that have the file.

While working on the problem, it quickly became apparent that there are other decentralised networks that are facing similar problems. The two most prominent examples are mobile ad-hoc networks and computational grids. While the first are heavily bandwidth constrained, the latter have CPU times as a different kind of service to file sharing. It might be worth investigating if our algorithms work under those conditions.

We hope that in the long term our proposed system to incentivise the sharing of small and rare will allow for fully decentralized, user-driven, all-encompassing file repositories to emerge. These would be much more resilient than the existing predominantly centralized servers and could achieve a real long-term availability of a wide range of files completely independent of the continued existence of individual organisations.

Bibliography

Ackemann, T., Gold, R., Mascolo, C., and Emmerich, W. (2002). Incentives in Peer-to-Peer and Grid Networks. Technical Report RN/02/24, UCL.

Adar, E. and Huberman, B. A. (2000). Free riding on Gnutella. *First Monday*, 5(10).

Anagnostakis, K. G. and Greenwald, M. B. (2004). Exchange-Based Incentive Mechanisms for Peer-to-Peer File Sharing. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, ICDCS '04, pages 524–533, Washington, DC, USA. IEEE Computer Society.

Andrade, N., Mowbray, M., Lima, A., Wagner, G., and Ripeanu, M. (2005). Influences on cooperation in BitTorrent communities. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, P2PECON '05, pages 111–115, New York, NY, USA. ACM.

BBC iPlayer (2010). BBC iPlayer. http://www.bbc.co.uk/iplayer/.

Binmore, K. (1992). *Fun And Games*. D.C.Heath and Company.

Buragohain, C., Agrawal, D., and Suri, S. (2003). A Game Theoretic Framework for Incentives in P2P Systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, P2P '03, pages 48–, Washington, DC, USA. IEEE Computer Society.

Chun, B., Fu, Y., and Vahdat, A. (2003). Bootstrapping a Distributed Computational Economy with Peer-to-Peer Bartering. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*.

Cohen, B. (2003). Incentives Build Robustness in BitTorrent. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*.

Dash, R., Jennings, N., and Parkes, D. (2003). Computational Mechanism Design: A Call to Arms. *Intelligent Systems, IEEE*, 18(6):40 – 47.

EDonkey (2010). The EDonkey Network. http://en.wikipedia.org/wiki/EDonkey_Network.

eMule (2010). The eMule Project. http://www.emule-project.net.

Fedora (2010). Fedora Project Bittorrent Tracker. http://torrent.fedoraproject.org/.

Feigenbaum, J. and Shenker, S. (2002). Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13. ACM Press, New York.

Félegyházi, M., Hubaux, J.-P., and Buttyán, L. (2006). Nash Equilibria of Packet Forwarding Strategies in Wireless Ad Hoc Networks. *IEEE Trans. Mob. Comput.*, 5(5):463–476.

Ganesan, P. and Seshadri, M. (2005). On Cooperative Content Distribution and the Price of Barter. In *Proceedings for the 25th IEEE International Conference on Distributed Computing Systems. ICDCS 2005*, pages 81 –90.

Giuli, T. and Baker, M. (2002). Narses: A Scalable, Flow-Based Network Simulator. *Technical Report, Department of Computer Science, Stanford University*.

Gnutella (2010). Gnutella Peer-to-Peer Network. http://en.wikipedia.org/wiki/Gnutella.

Golle, P., Leyton-Brown, K., and Mironov, I. (2001). Incentives for Sharing in Peer-to-Peer Networks. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, pages 264–267, New York, NY, USA. ACM.

Gupta, R. and Somani, A. (2004). Reputation Management Framework and Its Use as Currency in Large-Scale Peer-to-Peer Networks. In *Peer-to-Peer Computing, 2004. P2P 2004. Fourth International Conference on*, pages 124 – 132.

Ham, M. and Agha, G. (2005). ARA: A Robust Audit to Prevent Free-Riding in P2P Networks. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 125 – 132.

Handurukande, S. B., Kermarrec, A.-M., Fessant, F. L., and Massoulié, L. (2004). Exploiting semantic clustering in the eDonkey P2P network. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, EW 11, New York, NY, USA. ACM.

Jun, S. and Ahamad, M. (2005). Incentives in BitTorrent Induce Free Riding. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems*, P2PECON '05, pages 116–121, New York, NY, USA. ACM.

Jurca, R. and Faltings, B. (2005). Reputation-based pricing of P2P services. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, P2PECON '05, pages 144–149, New York, NY, USA. ACM.

Kamvar, S., Schlosser, M., and Garcia-Molina, H. (2003a). Incentives for Combatting Freeriding on P2P Networks. In *Proc of International Conference on Parallel and Distributed Computing (Euro-Par 2003)*.

Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003b). The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA. ACM.

Kazaa (2010). Kazaa Peer-to-Peer Network. http://www.kazaa.com.

Koo, S., Lee, C., and Kannan, K. (2005). A Resource-trading Mechanism for Efficient Distribution of Large-volume Contents on Peer-to-Peer Networks. In *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, pages 428 – 433.

Mahajan, R., Rodrig, M., Wetherall, D., and Zahorjan, J. (2004). Experiences Applying Game Theory to System Design. In *PINS '04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 183–190. ACM Press.

Maymounkov, P. and Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*.

Menasché, D. S., Massoulié, L., and Towsley, D. (2010). Reciprocity and Barter in Peer-to-Peer Systems. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1505–1513, Piscataway, NJ, USA. IEEE Press.

Napster (2010). Napster Peer-to-Peer Network. http://en.wikipedia.org/wiki/Napster.

Ngan, T., Nandi, A., Singh, A., Wallach, D., and Druschel, P. (2004). On designing incentives-compatible peer-to-peer systems. In *Proc. FuDiCo 2004*.

Nisan, N. and Ronen, A. (1999). Algorithmic Mechanism Design. In *Proc. 31st ACM Symp. on Theory of Computing*, pages 129–140.

ns-2 (2010). The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

Odlyzko, A. (2000). The history of communications and its implications for the Internet. Manuscript.

Parker, A. (2005). P2P - Past, Present and Future. Invited talk at The Future of Peer-to-Peer Computing & Communication Platforms Conference, London, 12 December 2005.

Parkes, D. (2001). *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania.

Peersim (2010). Peersim, a flow-based network simulator. http://peersim.sourceforge.net/.

PlanetLab (2010). PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

Qiu, D. and Srikant, R. (2004). Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. *SIGCOMM Comput. Commun. Rev.*, 34:367–378.

Ranganathan, K., Ripeanu, M., Sarin, A., and Foster, I. (2003). To Share or not to Share: An Analysis of Incentives to Contribute in Collaborative File Sharing Environments. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A Scalable Content-Addressable Network. *SIGCOMM Comput. Commun. Rev.*, 31:161–172.

Rivest, R. (1992). The MD5 Message-Digest Algorithm. RFC 1321 (Informational).

RngPack (2010). RngPack: High-Quality Random Numbers for Java. http://www.honeylocust.com/RngPack/.

Rowstron, A. I. T. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350.

Sanghavi, S. and Hajek, B. (2005). A New Mechanism for the Free-rider Problem. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, P2PECON '05, pages 122–127, New York, NY, USA. ACM.

Shenker, S. and Feigenbaum, J. (2001). Fundamental Open Problems in Distributed Mechanism Design. Presentation to DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design.

Sherwood, R., Braud, R., and Bhattacharjee, B. (2004). Slurpie: A Cooperative Bulk Data Transfer Protocol. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 941 – 951 vol.2.

Skype (2010). Skype Internet Telephony. http://www.skype.com/.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *SIGCOMM Comput. Commun. Rev.*, 31:149–160.

Stutzbach, D., Zhao, S., and Rejaie, R. (2007). Characterizing Files in the Modern Gnutella Network. *Multimedia Syst.*, 13(1):35–50.

Sun, Q. and Garcia-Molina, H. (2004). SLIC: A Selfish Link-Based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*.

Varga, A. and Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10.

Wilcox-O'Hearn, B. (2002). Experiences Deploying a Large-Scale Emergent Network. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 104–110, London, UK. Springer-Verlag.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

Yang, B., Condie, T., Kamvar, S., and Garcia-Molina, H. (2005). Non-Cooperation in Competitive P2P Networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, pages 91–100, Washington, DC, USA. IEEE Computer Society.

Yang, B. and Garcia-Molina, H. (2003). PPay: Micropayments for PeertoPeer Systems. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 300–310, New York, NY, USA. ACM.

Zhao, B. Y., Kubiatowicz, J., and Joseph, A. D. (2002). Tapestry: A Fault-tolerant Wide-area Application Infrastructure. *Computer Communication Review*, 32(1):81.