The University of Birmingham
School Of Computer Science
MSc In Advanced Computer Science

First Semester Mini Project
A Study On Peer-to-Peer Systems

Charalampos Margiolakiotis
Supervisor: Dr Georgios Theodoropoulos

January 15, 2007

Abstract

At the same time when "the end of the personal computer" was a much over-hyped concept, Peer-to-Peer (P2P) computing made its appearance. It is essentially collaborative networking where a PC is not anymore just a "dump" terminal at the end of a client-server connection but an organic -yet not trusted- part of a network of peers. The evident success of file sharing networks such as Napster, later followed by Gnutella and BitTorrent among others, triggered a new wave of research on this new computing discipline.

The first two generations of P2P networks, based on an unstructured architecture, are the ones used by the widely known file sharing applications since 1999. More recently a third generation of P2P networks introduced a structured approach using Distributed Hash Tables (DHT) and has become the main focus of current research.

The goal of this project is to look into these systems, critically review the different approaches undertaken and produce a comparative analysis. After presenting the necessary background information, the different possible classifications are analysed. The structure of the network is the axis around which we categorise the various systems as structured or unstructured. The operation of these systems is presented and their advantages/disadvantages are indicated. We then carry on to discuss current challenges in research that need to be faced in order for peer-to-peer to deliver its promise. Finally an overview of available simulators is presented.

Keywords

peer-to-peer; p2p; overlay networks; DHT ; p2p simulators; distributed computing

Acknowledgements

I would like to thank my supervisor Dr. Georgios Theodoropoulos for his guidance and valuable support throughout this project. I am also thankful to Robert Minson for answering my questions and to Theodoros Tsokos for moral support and proof reading my report.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It was Napster in 1999 that introduced to the public a new approach of distributed computing known as peer-to-peer networking. Immediately attracting millions of users, such systems came in the forefront of a range of technologies that were going to change the way we see the Internet. This new approach in networking, very soon got the attention of the internet community. Not unexpectedly, its apparent success fuelled academic research which gave birth to new generations of systems but also new challenges that need to be faced. Nowadays, the term peer-to-peer is for distributed computing what the term "Web 2.0" has become for internet applications.

But besides being a buzz-word, peer-to-peer's promise has provoked considerable activity in the academic community and also the industry. Many new applications have appeared and Internet Service Providers report large proportions of their overall traffic being caused by peer-to-peer implementations for file sharing. The academic world has responded to this newly found network architecture with a vast amount of publications since the beginning of the millennium. The initial wave was "re-active" in that it mainly examined the existing systems. Following was a flourish of new ideas that either improved extensively current ideas or proposed completely new approaches.

It is the purpose of this report to critically explore the extensive amount of knowledge that has been produced in the last few years, survey the different systems that have been introduced and investigate challenges faced in different aspects of current research. The remaining part of this report is organised as follows:

- **Chapter 2** provides background information, defines peer-to-peer systems, presents their promise and explores application areas.

- **Chapter 3** attempts to taxonomise existing peer-to-peer networks in order for a more formal investigation to be possible.
- **Chapter 4** analyses the first two generations of peer-to-peer systems and explores their advantages, disadvantages and future challenges.
- **Chapter 5** focuses on third generation systems which are in the centre of current research. Three representative systems are analytically presented and current challenges are discussed.
- **Chapter 6** describes how simulations are an important part of research on peer-to-peer and overviews existing tools.
- **Chapter 7** concludes and suggests how this project could be extended.
- In the **Appendix** there can be found a brief report on JXTA, a generic peer-to-peer platform that was examined during this project with regard to whether it can be easily amended to use a different set of algorithms.

Chapter 2

Background

It was only very recently, around 2000, that people started becoming aware of peer-to-peer technologies. It was companies -or rather phenomena- like Napster that brought peer-to-peer in the public's eye. Napster started providing its services in September 1999 and by 2001 it was eventually shut down by the music industry. By then its network consisted of 50 million users (Barkai, 2001). The characteristics of this technology, its popularity, potential and its effect on the field of distributed computing has lead many to call it a "disruptive technology" (Christensen, 1997).

Nowadays, many internet service providers report that 50%-75% of overall internet traffic is caused by peer-to-peer implementations, mainly file sharing applications (Steinmetz and Wehrle, 2005). In fact, such networks could not exist without peer-to-peer technology, not only because of legal issues but mainly because of the cost they would have using traditional technologies. The stupendous requirements in terms of resources and effort render existing solutions based on the client server architecture incapable of meeting such scalability or at least *very* costly in order to do so. A very illustrative example is given by Chawathe et al. (2003) : 100,000 peers connected at 56kbps outperform a single server farm connected by 2 OC-48 links in terms of aggregate bandwidth. Given the growth of the Internet this will only become more evident in the future.

2.1 Definition

According to the Oxford English Dictionary a peer is "*a person of the same age, status or ability as another specified person*". Based on this definition we can have a -rather simplistic- view of what is meant by a the term "peer-to-peer network" :

A network of equals. A step forward from existing architectures where the client is nothing more than a dump terminal at the end of client-server connection to a new kind of network where all participants act both as clients and servers.

In fact, peer-to-peer networks are overlay networks in the sense that they are implemented on top of the underlying physical network. An illustration of this can be seen in Figure 2.1

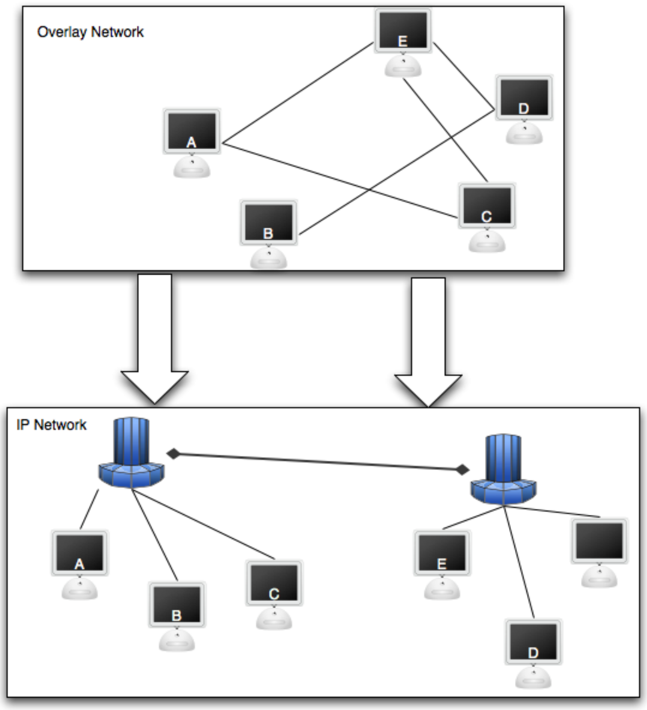

Figure 2.1: An overlay network illustration

2.1.1 A more formal definition

There has been a lot of interest in the peer-to-peer field by the research community resulting in a vast number of papers being published in the recent years. In this wide range of publications one can find somewhat different definitions for the term. Shirky (2000) defines peer-to-peer as "*a class of applications that takes advantage of resources –storage, cycles, content, human presence– available at the edges of the internet*". However, this is probably too broad. Androutsellis-Theotokis and Spinellis (2004) opine that this variation in definitions derives from the fact that people sometimes classify an application as peer-to-peer , not because of its internal operation but rather because it appears to facilitate direct communication between computers. They carry on to define peer-to-peer systems as "distributed systems consisting of interconnected nodes able to self-organise into network topologies with the purpose of sharing resources, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralised server or authority."

Nevertheless, for this report an even more strict definition is adopted. Roussopoulos et al. (2003) define a peer-to-peer system as one that satisfies three criteria:

1. Self organisation : There is no central directory. Nodes organise themselves.
2. Symmetric communication : Nodes act both as clients and servers.
3. Decentralised Control : No centralised mechanism exists that guides nodes. They decide for themselves how they participate in the network.

It is worth noting that under these terms, Napster does not qualify as a pure peer-to-peer application, as it does have a centralised service[1]. This however, reflects rather accurately the common conception of Napster by many researchers.

Schoder et al. (2005*a*) focus on decentralisation, autonomy and distribution of resources and services for their definition. They construct a three level model to resolve the "lack of clarity in respect to terminology which exists in theory and practice". The first level consists of the various peer-to-peer infrastructures built on top of the underlying physical network, which provide various services (e.g. resource locating). Level 2 represents the applications built on top of the infrastructures and finally level 3 describes the "social interaction phenomena" i.e. communities and their internal dynamics.

2.2 Why Peer-to-Peer?

Steinmetz and Wehrle (2005) explain how future internet applications will have three main requirements: *scalability*, therefore design flaws such as bottlenecks are not acceptable, *security and reliability* in the face of malicious attacks on centralised systems and *flexibility and Quality of Service*. The classic client-server architecture struggles to keep up with these increasing requirements as -among others- its centralised nature creates single points of failure.

[1]The operation of Napster is explained more analytically in the following chapters

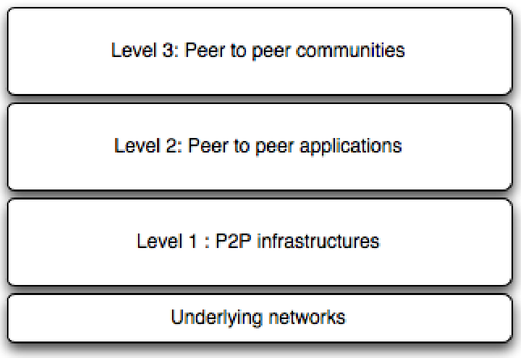
Level 3: Peer to peer communities
Level 2: Peer to peer applications
Level 1 : P2P infrastructures
Underlying networks

Figure 2.2: Levels of peer-to-peer networks according to Schoder et al. (2005*a*)

Peer-to-peer architectures on the other hand promise to provide a highly resilient network that will scale and function despite having to deal with transient nodes. They actually manage that very efficiently while they also have the ability to self-organise. Moreover, they do not need a central server (or the administration costs) and are (therefore) resistant to censorship or malicious attacks. In fact, maintenance costs and responsibility for the good operation of the system is divided among the participating nodes (Androutsellis-Theotokis and Spinellis, 2004). Thus, starting, maintaining and expanding such a system becomes a lot easier and cheaper. Schoder et al. (2005*a*) summarise the above benefits of using peer-to-peer in reduced cost of ownership, support for ad-hoc networks and good scalability. Hence, there are some very important motivations behind peer-to-peer .

Overall, by applying a peer-to-peer solution to a problem t

2.3 A bit of history

On a very interesting note, (Minar and Hedlund, 2001) suggest that the internet was essentially a peer-to-peer network. It was built with the purpose of allowing a diversity of networks to interconnect. The interconnections were not on the basis of a client/server architecture, but on a similar approach to what we see in peer-to-peer networks. In fact, the internet was built in a way that a single point of failure would not exist. These are some important similarities, but it would be rather risky to consider the Internet as the first peer-to-peer network. According to the definitions given above there are two very important differences : it is not an overlay or a self-organising network.

Usenet developed in 1979 fits the description much better and therefore could be considered as the first peer-to-peer application (Eberspacher and Schollmeier, 2005*b*; Risson and Moors, 2004). Peers in Usenet communicated with each other in order to propagate news articles over the whole Usenet network. However, there was some sort of hierarchy with some nodes being more powerful and providing content. As we will see later in the report however, the second generation of peer-to-peer systems introduces some kind of hierarchy as well. Nevertheless, the Internet was dominated by the client-server type of applications, until very recently.

In May of 1999 Napster (*Napster*, 2007) started its operation. Some aspects of the system were still centralised but it was the first network where the content was distributed among the peers. Second generation systems came along that introduced a completely decentralised topology. Third generation systems were born in the laboratory in the sense that they are mostly a product of research inside universities, while Napster and Gnutella, second generation systems, were introduced in the commercial world. Current research is mainly focused on improving and extending third generation systems.

2.3.1 Grid Computing

Another take on distributed computing is Grid computing. Along with peer-to-peer computing these are two approaches both concerned with resource sharing in computer networks. Foster and Iamnitchi (2003) define grids as sharing platforms built on a standardised infrastructure that allows for the creation of "distributed communities" in which resources can be shared. They describe the comparison between peer-to-peer and grid computing as "Grid computing addresses infrastructure but not yet failure, whereas P2P addresses failure but not yet infrastructure". In this respect, the two technologies have a lot to offer to one another. However, given the number of participating users and traffic peer-to-peer computing has a more evident presence. Androutsellis-Theotokis and Spinellis (2004) indicate that in the light of more complex and sophisticated applications to come, the two technologies are very likely to merge to a new category of approaches that will combine merits of both worlds.

2.4 Applications

A wide range of applications can be modified or build from scratch to run on a peer-to-peer overlay. Currently, the widest use of current peer-to-peer networks are for content distribution, more specifically file sharing applications. In fact, in this report such applications are the most used example in order to illustrate the different approaches undertaken in peer-to-peer research. There are numerous applications in this area many of which will be examined in later chapters. The core idea behind these is to provide an infrastructure that enables peers to share data (in the simplest form) or infrastructures for more sophisticated applications such as distributed storage systems.

Moreover, Schoder et al. (2005*b*) indicate that the unused computing power found in network entities gave the incentive for peer-to-peer applications that allow for the distribution of computing power. Essentially, a single task is broken down into smaller units distributed among peers in the network which will eventually return results. In such systems some central coordination is inevitably required in order to perform the breaking down of the work and to collect the results. An example of this category is SETI@HOME (*The SETI@HOME project*, 2007).

There have also been implementations of the peer-to-peer paradigm in the communications field. A prime example of this is the closed source[1] application Skype (*Skype*, 2007). Such applications facilitate real-time messaging in the form of text messaging and/or audio and video conferencing. Another example of this is Jabber(*Jabber*, 2007).

Androutsellis-Theotokis and Spinellis (2004) mention two more categories of applications: *Internet Service Support* and *Database Systems*. The former refers to applications built to support different aspects of internet services. An example is multicast systems built on top of peer-to-peer systems (Yeo et al., 2004). There have also been tries to port databases on peer-to-peer infrastructures resulting in a distributed database system. More formally such a system is characterised as "an integration system, composed by a set of (distributed) databases interconnected by means of some sort of logically interpreted mappings"(Franconi et al., 2003).

A comprehensive and more technical analysis of existing peer-to-peer applications and their potential can be found in Verma (2004).

2.5 Challenges

Essentially, the key to success for peer-to-peer systems is to deliver their promises along in an efficient way while avoiding other side-effects. For example, Schoder et al. (2005*a*) note how security mechanisms such as authentication and authorisation but also accountability are better dealt with in the client-server model. Moreover,

[1]Hence, what we know about it has derived from reverse engineering techniques

there are various approaches on how to deliver the desired network resilience in high "churn[2]" rates or how to deal with the challenge of scalability combined with efficient search mechanisms. Other issues include the availability of resources that are not highly replicated in the network in terms of search[3] or securing access to these resources in case of node failures. Such issues are further analysed and discussed in the following chapters.

[2]A state where large numbers of nodes in the network join and leave simultaneously

[3]In fact, this is directly related to search mechanisms in terms of discovering such resources

Chapter 3

Taxonomy

Proper classification of the different aspects of a research area is essential for the better understanding of the field and a very important prerequisite for further studying it.

As one would expect from a field that has got so much attention lately, there have been different approaches on how the various peer-to-peer systems can be classified. Most are not necessarily contradictory but emphasise on different aspects of the system and therefore adopt different criteria. Even approaches with similar criteria sometimes result in somewhat different results depending on the point of view of the author.

In the following sections we will examine the two basic characteristics of peer-to-peer systems that can be used for classification:

1. Centralisation : Related to the topology of the network. Depends on whether the network has some kind of centralised structure or not.
2. Structure: Relates to the routing and location infrastructure used in the system. Essentially this differentiation depends on whether the network is created deterministically or not. Of course the nature of this infrastructure also affects the topology of the network.

It has to be noted that these two characteristics are not independent of one another but merely two different characteristics of the same system. For example all structured networks are decentralised. This will be explained better in the following sections.

3.1 Centralisation

As it has been explained in previous chapters peer-to-peer networks are overlay networks. The most obvious and intuitive way to distinguish between the different "flavours" of such systems is the topology of this overlay network, the way the nodes are organised and whether there are central points such as servers or "supernodes".

3.1.1 Centralised Architectures

Some of the first generation peer-to-peer systems can be classified as such. Napster is probably the most prominent example which as Eberspacher and Schollmeier (2005*a*) indicate is considered to be the starting point of peer-to-peer networks.

Such systems facilitate searching by keeping a centralised directory of the available content in a server. Then, when a peer would like to find available content in the network it would communicate with the server to submit the search query and the server would reply with the available content -if any- and with the location that this content can be found (practically another peer in the network that has it). The requesting peer would then communicate directly with the peer returned by the server's reply and download the required content (Yang and Garcia-Molina, 2001). Such a network can be seen in figure 3.1.

The fact that the content is distributed among the nodes while the directory is centralised has lead many researchers to classify such networks as *Hybrid Decentralised* (Androutsellis-Theotokis and Spinellis, 2004; Eberspacher and Schollmeier, 2005*a*; Schmidt and Parashar, 2005; Yang and Garcia-Molina, 2001). In fact this is a more appropriate term as these networks can only be called "centralised" in the context of the peer-to-peer networks field. Outside this context they are in no way centralised in the classic sense (e.g. the client-server model). The terms "broker mediated" and "peer through peer" are also sometimes used to describe these networks (Kim, 2001; Taylor, 2005).

This "semi-distributed" nature of these systems allows them to avoid the problem of keyword search queries that other systems have problems with as it will be shown later. However, this centralised architecture also means a *central point of failure*. Androutsellis-Theotokis and Spinellis (2004) indicate that this typically means that they are very inefficient when it comes to scalability, and vulnerable to security related issues or technical failures, attacks and censorship.

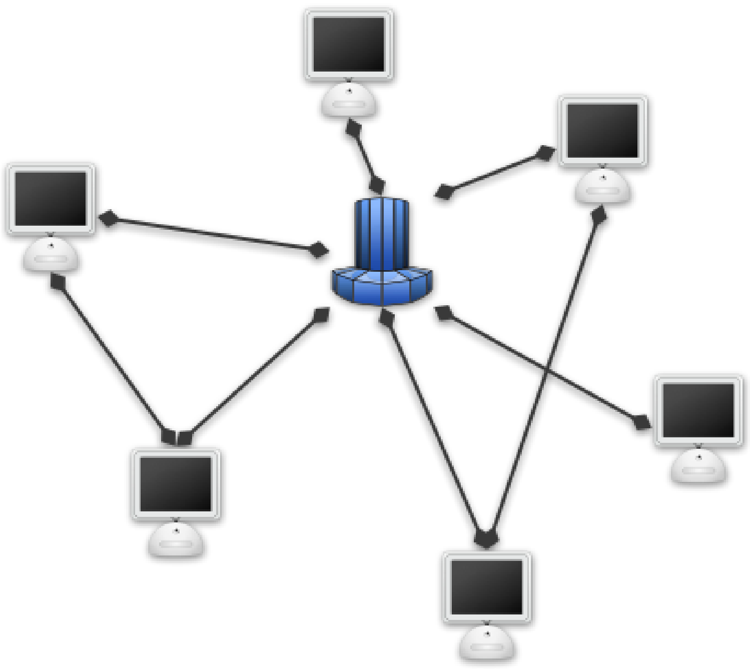

Figure 3.1: A simple example of a centralised system

3.1.2 Purely Decentralised Architectures

This kind of architecture requires all peers to act as both a client and a server. All nodes are therefore “assigned” the same tasks and there is no central point of failure. Sometimes these nodes are called ‘‘servents” a combination of the words “servers” and “clients” (Androutsellis-Theotokis and Spinellis, 2004). An illustration of how such a network might look like can be seen in figure 3.2.

The nature of these networks requires that the index cannot be stored on a central server. This means that it can be either distributed or local. Networks which adopt the local index logic require each node to hold an index for its own content. As Risson and Moors (2004) indicate an early P2P implementation for a distributed index network was Freenet (Clarke, 1999, 2003). The same paper goes on to indicate that early versions of Gnutella(v0.4) (*Gnutella*, 2007) used a locally stored index which is however a terribly inefficient technique: In order for a peer to find available content in the network the peer has to flood the whole network by broadcasting its request and wait for a response from the nodes that have the requires content (Castro et al., July 2004*b*).

This of course renders the network almost unable to scale to larger sizes. However, this also means that the network is very fault tolerant and if a node fails or just

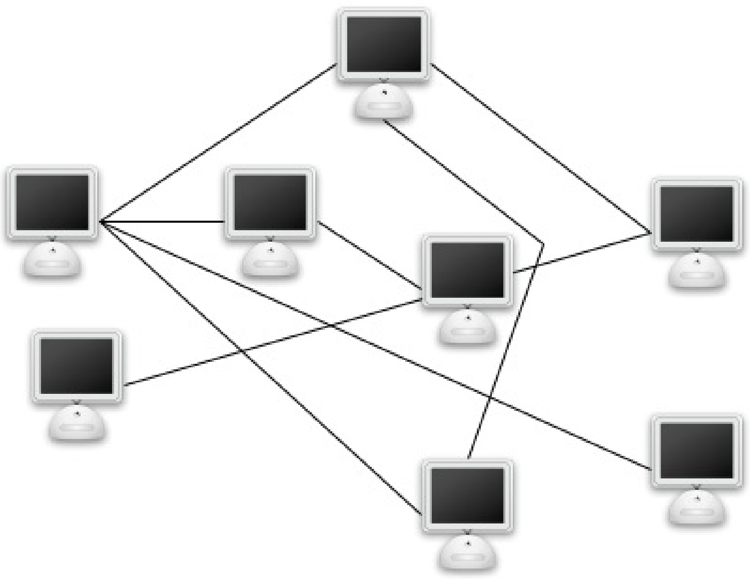

Figure 3.2: A purely decentralised architecture

disconnects this would not affect the network. When a node wants to join the network the only requirement is for it to connect to any existing and active peer (Karwaczynski and Kwiatkowski, 2005). Some techniques to improve the scalability issues will be discussed later in the report.

It is important to note that under this classification, besides the Gnutella-type networks, also fall networks that are formed deterministically (Karwaczynski and Kwiatkowski, 2005). Such networks form connections between their peers that are somehow organised, structured. However, "structured" in this case does not refer to the network topology but merely to the fact that peers do not join the network at random locations, as with Gnutella for example, but deterministically take a position in the decentralised network. More about these networks is discussed later on.

3.1.3 Partially decentralised architectures

The basic idea behind this architecture is similar to the one behind purely decentralised architectures. Chawathe et al. (2003) explains how Gnutella in its later versions evolved to incorporate this architecture. Some nodes of the network are now assigned a more important role: They act as central indexes for the surrounding nodes therefore forming a network topology like the one illustrated in figure 3.3. It is worth-noting that this does not constitute central points of failure though, as such nodes are dynamically assigned and in the case of failure the network takes action to replace them. The way such"supernodes" are selected differs from implementation to implementation.

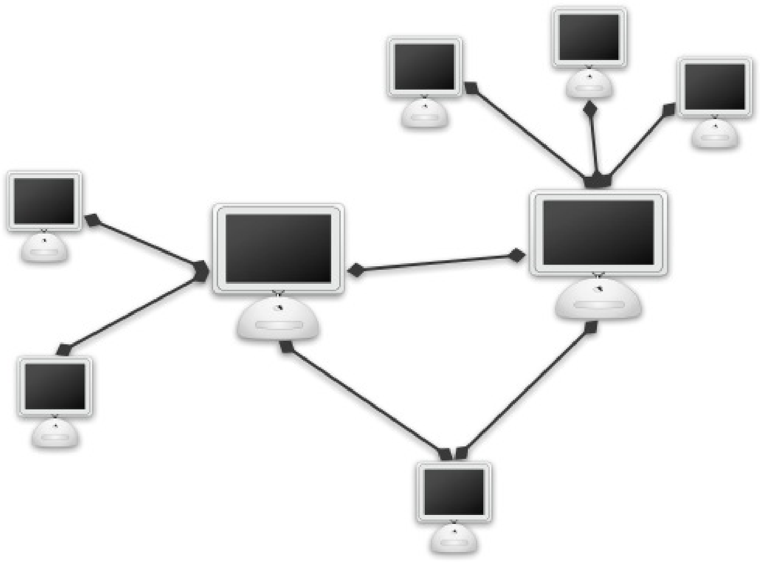

Figure 3.3: A partially decentralised architecture

3.2 Network Structure

This classification relates to the routing algorithm that is used in the network for resource discovering (Karwaczynski and Kwiatkowski, 2005). The existence -or not- of such an algorithm also affects the network topology but the main focus of this classification resides on whether the nodes join the network in a deterministic manner or not.

3.2.1 Unstructured Networks

In such networks the placement of the available content is not relevant to the network topology (Androutsellis-Theotokis and Spinellis, 2004). Then, a peer that requests a file for example needs to be provided with facilities that will provide it with the location of that file. In unstructured networks this has been implemented in various ways.

Examples include flooding the network as we described in the case of Gnutella 0.4. As it was indicated though this can have very important disadvantages regarding the scalability but also other aspects of the system. The main issues that arise in unstructured networks were discussed in the previous section, which proves what was suggested in the introduction of this chapter that the two different ways to categorise peer-to-peer networks, are not unrelated.

3.2.2 Structured Networks

It is the obvious downsides of unstructured networks that lead to the development structured architectures. Risson and Moors (2004) indicate that some have called unstructured approaches “first generation” and perceive them as inferior to the “second generation”structured ones. In a structured network a completely decentralised network topology is used which is tightly controlled by the algorithm. The nodes collaborate in order to maintain this structure while allowing for the system to be dynamic (in the sense that nodes can leave and join at any time) (Schmidt and Parashar, 2005).

As Wehrle et al. (2005) explain, the basic idea behind this approach is to combine the advantages of unstructured networks, but on the same time avoiding the drawbacks. Essentially, the ideal solution would combine the efficient searching of server based techniques with the storage efficiency that flood based techniques have (i.e. no central point needs to keeps data).

Scalability being the main drawback in these cases, Wehrle et al. (2005) indicate that for good scalability the requirement would be for the search and storage complexity of the system not to increase significantly (not more than $O(logN)$) even if the population grows by some orders of magnitude. They carry on to note that most often this is materialised in the form of Distributed Hash Tables (DHT).

Essentially, DHT-based approaches provide a way to map content to location through a universal algorithm. This same algorithm is used for inserting data and nodes deterministically in the network *and* for locating content. Queries for content then can be routed to the location of the appropriate node without searching since that location is already known through this algorithm.

Examples of DHT-based infrastructures include Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron and Druschel, 2001) and Tapestry (Zhao et al., 2001).

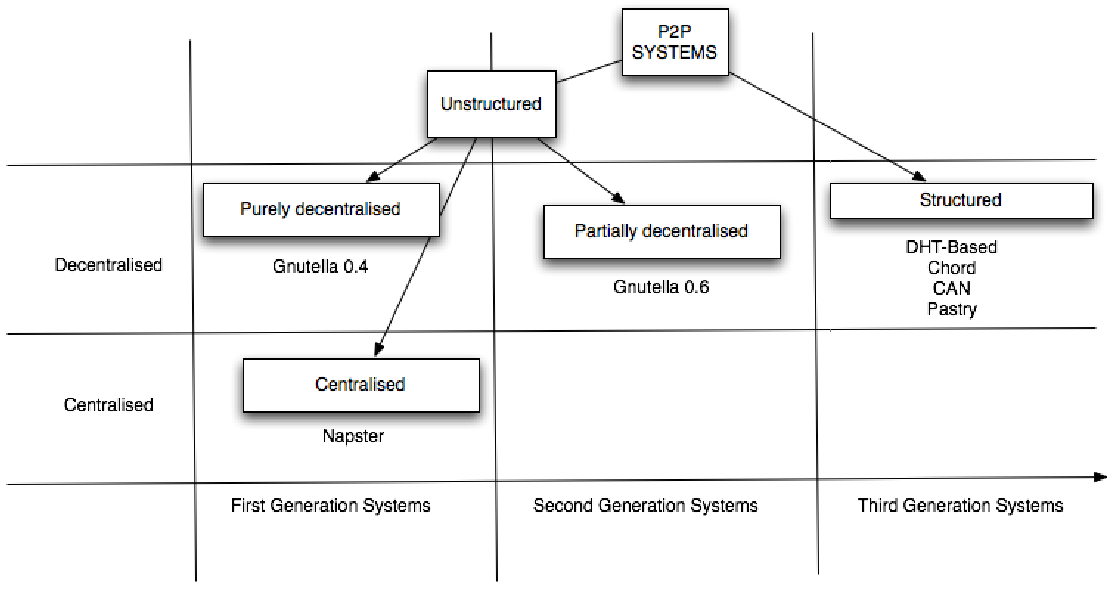

Figure 3.4: Taxonomy in peer-to-peer systems

3.3 Other approaches

Schmidt and Parashar (2005) introduce a somewhat different classification:

1. Data Lookup Systems: What has been described here as structured systems.
2. Hybrid: The equivalent to centralised systems
3. Unstructured : Gnutella like systems
4. Structured Keyword Search : Structured systems that support keyword searches

In another different approach, Mischke and Stiller (2004) have suggested a categorisation that depends on whether the system has structure in routing tables and network topology.

Moreover, Nejdl et al. (2003) have introduced a new approach inspired by the schema-based approach similar to what database systems have been utilising for storing and retrieving data: a schema-based network with supernodes and structure within documents. According to their paper with this new approach peer-to-peer systems are classified as follows:

- Key-based: The classic DHT-based systems.

- Fixed schema / keywords : Systems like Gnutella, Kazaa etc
- Schema-based : The Edutella infrastructure introduced by them which adopts a schema-based approach.

However, Risson and Moors (2004) do introduce the differences between structured and unstructured networks, but carry on to use index as the reference point for their classifications. In peer-to-peer networks the index is local, distributed or centralised. The classic example of local indexes as described in a previous section is Gnutella in its early versions. Napster and DHT-based approaches are examples for central index and distributed index systems respectively. Moreover, they mention that an index can be forwarding or non-forwarding depending on whether the content is reached within one hop (non-forwarding) or more (forwarding). Finally, peer-to-peer indexes can be semantic or semantic-free. Naturally, a semantic index is human readable while a semantic-free is not. A semantic-free index is designed to be "more data-centric", as they put it.

3.4 Chosen Approach

The most often used classification of peer-to-peer systems is the one that divides the field in structured and unstructured approaches. Risson and Moors (2004) discuss how this approach is becoming less relevant for two reasons. Firstly, they argue that most unstructured proposals are evolving to incorporate some kind of structure, therefore making the distinction between the two different kinds less clear. Secondly, the new schema based peer-to-peer designs -mentioned above- introduce a new category that cannot be classified under structured or unstructured designs.

However, the structured/unstructured approach is dominant overall in the literature and has been selected for the purposes of this report. The main criterion of this approach (i.e. determinism in the way the network formulates) constitutes arguably a very basic difference on the design of the network. Moreover, as it can be seen in the previous section, other approaches have mostly expanded this approach (and not introduce a completely new one) for the purposes of their research. For example Schmidt and Parashar (2005) introduce the new category "Structured Keyword Search" for DHT-based systems that facilitates keyword searches.

Chapter 4

Unstructured Peer-to-Peer Systems

In this chapter we will look further into what we have classified as unstructured peer-to-peer networks. As it was explained in the previous chapter while all structured architectures are decentralised, this is not the case with structured networks. It would be rather inelegant to try and analyse everything under the same umbrella.

Therefore, the classification introduced earlier in this report based on the network topology will be used. Namely, we have the following categories:

1. Centralised Networks
2. Purely Decentralised Networks
3. Partially Decentralised Networks

4.1 Different Generations

According to Eberspächer et al. (2004) there are two generations of unstructured peer-to-peer networks. The first generation comprises of the centralised architectures (Napster) and the purely decentralised systems (early Gnutella). This is not to be confused with decentralised systems in general. In this case we are referring to *unstructured* decentralised systems. The second generation consists of hybrid decentralised systems such as later versions of Gnutella (v0.6). Note how this is contradictory to Risson and Moors (2004)'s definition where first generation are unstructured systems in general and second generation are the structured ones.

To avoid confusion in this report we will consider structured systems *third* generation systems and will adopt Eberspacher and Schollmeier (2005*a*)'s view for the first two generations.

4.2 First Generation Systems

4.2.1 Centralised Systems Analysed

Figure 3.1, as explained in section 3.1.1, illustrates how a centralised peer-to-peer network looks like. In fact in the diagram one can notice that besides the connections of the peers to the central entity, there are also connections between them. These connections are established when the two peers are exchanging content.

More analytically, all clients are connected to the server as soon as they join the network. Each of the clients stores content locally that can be shared with the rest of the network. As soon as they connect to the server they need to provide information regarding this content, essentially a table of contents, but also information regarding the computer itself (e.g. connection speed, ip address). The overlay network constructed in this manner, is basically a star network.

It is after this initial formulation that the peer-to-peer element becomes relevant. In order for a client to find available content, it sends a query request to the server. The server having been informed about the tables of contents of all connected peers, is able to reply to this request by sending back to the client information about where this data is available in the network (IP's, ports). Then the requester communicates directly with the computer in this "co-ordinates" and they form a connection *outside* the overlay network. This connection over which the data is transmitted, is usually HTTP-based (Eberspacher and Schollmeier, 2005*a*).

As it has been mentioned before, the most well known example utilising this architecture is Napster (*Napster*, 2007). As Ding et al. (2005) points out the exact details of its protocol are not known, but through reverse engineering and the OpenNap Scholl (2001) protocol, it is considered that we have a very good description of its structure. Napster was a massive success that stimulated interest from the research and -more generally- the internet community but also from the music industry. It appeared at the same time that broadband connections were starting to become available. A good proportion of its users was using it to exchange music with other clients all over the world. It was eventually shut down by the music industry.

The fact that all content is known to the server meant that the system was aware that illegal music downloads were taking place and that is of course a legal issue. It is interesting to see how research papers, published before the legal issues were settled, suggest that such systems can be used through viable business models in a legal manner (e.g. Yang and Garcia-Molina (2001)). Nowadays, Napster has come back to operation as a way to legally download music. WinMX (*WinMX*, 2007) also used the same basic concepts behind Napster. More specifically it connected to OpenNap (a clone of Napster) servers (Scholl, 2001).

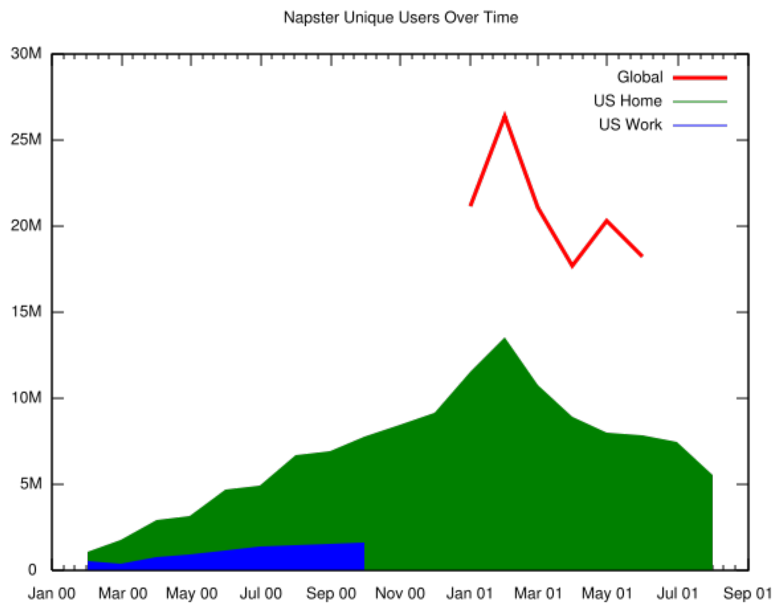

Figure 4.1: Napster Usage. Taken from wikipedia.org. Data based on Media Metrix press releases at [www.comscore.com]. This image belongs to the public domain.

In a very comprehensive study published in 2001 Yang and Garcia-Molina (2001) further divide hybrid decentralised architectures (another name for centralised as explained in the previous chapter) in four categories:

1. Chained architecture : Servers of the network are connected in a chain. This is used for replying to client queries. If a server does not have this content locally it forwards the query to another server and so on.

2. Unchained architecture : A set of independent servers that do not communicate with each other. This is the architecture that was used in Napster. One

can see that one disadvantage of this approach is that it does not allow the users to see the content available in the whole network.

3. Full replication architecture : Since forwarding client queries is not the most efficient technique, all servers of the network must keep a complete index of what is available in the whole network.

4. Hash architecture : metadata (keywords) are hashed to different servers in a way that a given server has a complete inverted list.

4.2.1.1 Discussion

There are certain advantages in this architecture. Firstly, its implementation is rather simple, or at least simpler than other, more advanced systems that appeared later on. The most important advantage however, is the ability of the network to locate available content efficiently since the server is aware of all available data. This also means that keyword queries are supported (this is not the case with DHT-based architectures as it will be shown).

However, as Risson and Moors (2004) point out, centralised systems are important because they demonstrated "the peer-to-peer scalability that comes from separating the data index from the data itself". A similar idea relies behind the operation of Google and Yahoo and as Chawathe et al. (2003) note, these services prove how a centralised system architecture can be a very successful model. They argue that in the case of Napster the millions of its users were eventually disconnected not because the system failed but because of the legal issues that were involved.

On the other hand, the central point of the system, the server, that all clients have to register with in order to join the network, constitutes a bottleneck. As the network grows the server infrastructure has to be enhanced (facilitate an increasing in size database and also bandwidth) to be able to handle the load, an obvious scalability issue. Furthermore, as Androutsellis-Theotokis and Spinellis (2004) note, since a single entity (company, administrator etc.) maintains the server and is aware of the whole content and able to manipulate access to it, the network is subject to:

- Censorship
- Technical failure - single point of failure
- Surveillance
- Malicious Attack

- Legal issues

Finally, it is worth noting that according to the definition of a peer-to-peer system by Roussopoulos et al. (2003), introduced in chapter 2, centralised architectures are not pure peer-to-peer systems as they do not have symmetric communication (peers do not act as both clients and servers) nor distributed control (there is a central control point). Nejdl et al. (2003) classify it under the category “Distributed” and not “P2P”.

4.2.2 Purely Decentralised Systems Analysed

Peer-to-peer networks that satisfy all three criteria for pure peer-to-peer systems while still remaining under the “unstructured” category, emerged right after Napster appeared. Figure 3.2 illustrates how such a purely decentralised network may look like. They are nowadays becoming rare -if not extinct- however, due to their evident disadvantages that we will discuss in this section.

Gnutella, and more specifically its early versions (v0.4), with its open architecture and the fact that it is a representative example of this category of peer-to-peer systems, is the most widely known example of decentralised unstructured networks(*Gnutella*, 2007; *The Gnutella Protocol Specification v0.4*, 2001). Indeed, in this section we will analyse it further.

When a node wants to join the Gnutella network a bootstrap server is used to allow it to find other peers already active in the network (Eberspacher and Schollmeier, 2005*a*). The existence of this server does not mean that the system has some kind of centrality. It is there merely to introduce new peers to the network. Users communicate with each other through an application that acts as both a server and a client (servent). According to the *The Gnutella Protocol Specification v0.4* (2001) there are 5 kinds of “descriptors” used for communicating data over a Gnutella network:

1. Ping : Used to discover other servents on the network. A servent that receives a ping is expected to reply with a pong.
2. Pong : As described above, this is the expected response to a ping. Contains information about the amount of data the servent is sharing with the network and its IP address and port.
3. Query : Used for searching the network. Contains a description of the query of the requesting servent and the minimum speed requirements.

4. QueryHit : A servent that receives a query and has the requested data, is expected to reply with a queryHit with information for the retrieval of the data (IP, port, number of hits, result set).

5. Push : A mechanish to allow the operation of the application behind firewalls.

Only the first four descriptors are of relevance to this section. Eberspacher and Schollmeier (2005*a*) describe how n order for a peer to become part of the network an average of 3 TCP connections are required to other active Gnutella nodes. New connections are then explored by sending a Ping message to all the nodes the peer is currently connected to. The nodes that receive the ping reply with a Pong in order to "identify" themselves according to the description given above, and also propagate this ping to any other nodes they are connected to. These ping messages are broadcasted in regular time intervals as they are also used as keep-alive pattern.

The messages exchanged in the network are in plain text which results in large message sizes if we consider Query and especially QueryHit messages (which contain descriptions of the data). Under this architecture there is not really any other way to search for content other than non-deterministically. When a peer requests data from other peers it simply floods the network with its request.

More analytically, when a peer sends either a Ping or a Query request it sends it to all the peers that it "knows". A node then receiving such a message replies by backward rooting sending a message on the same path but opposite direction. It also broadcasts the message to other nodes that this second node is aware of. The next node repeats the same procedure and so on. However, to limit flooding to some extend, there are some requirements for a message to be broadcasted. Message spread is limited by TTL (Time-To-Live) values that each message "carries". The default value is TTL=7. For every hop this value becomes $TTL = TTL - 1$. When the value becomes 0 the message is dropped. Furthermore, a message is only broadcasted if it reached that particular node (that is about to broadcast it) for the first time. An illustration of how a Query with TTL=2 travels through a Gnutella network can be seen in figure 4.2. Note how when a node requires content, the the search technique used is breadth first, or more accurately breadth first flooding.

The way messages are backwards routed to its original sender is implemented as follows: Each node has a routing table that is dynamically updated each time a message is received. Every message has a unique identifier. When a message is received, the routing table is updated with information about the sender of the message along with the message's identifier. The response messages also contain the same identifier and therefore each node through the routing table knows where the response should be send to. Moreover, using this identifier messages that are received for the second time, are dropped (Jovanovic, 2000).

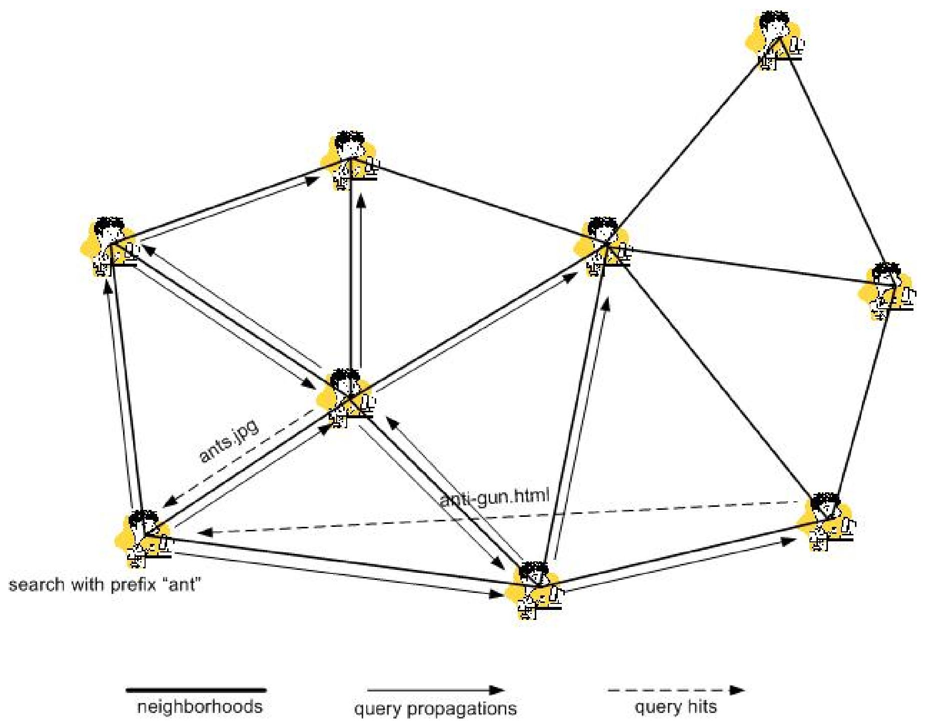

Figure 4.2: Query Flooding with initial TTL=2. Taken from www.cs.rochester.edu

Finally, when a peer discovers the requested content, a connection is established directly between the two nodes, outside the Gnutella overlay network. Note, that this is similar to the operation of Napster.

4.2.2.1 Discussion

The obvious advantages of the Gnutella (v0.4) network are that the network is inherently very adaptable to changes. Regardless of whether one or more nodes fail or just disconnect, the network will continue its operation undisturbed. In fact the network will continue to exist until the last of its nodes fails. Moreover, in contrast with Napster, the system is immune to censorship, technical failure, surveillance and malicious attacks as there is no central point of failure. Furthermore, there are no bottlenecks in the structure of the system.

However, there are a few important drawbacks in these early versions of Gnutella-like systems. The amount of information that has to be exchanged grows to be significantly more than what a Napster peer would cause (Eberspacher and Schollmeier, 2005*a*). It is worth noting that these calculations have been made considering only traffic in the application layer.

Moreover, Schollmeier and Kunzmann (2003) prove that the overlay network of Gnutella is not properly mapped over the physical network. This practically results to “zigzag routes over the Atlantic” and therefore very significant delays are created between messages and their responses. In fact these delays are noticeable from any user connected to the network. Of course, these zigzag routes also have an important impact on bandwidth consumption.

Finally, Jovanovic (2000) describes how the TTL limit imposed on messages, effectively creates a “horizon” for each peer that limits its view of the rest of the network. In other words, it practically divides the network into sub-networks. The bigger the TTL value the further a node can “see” but also the more traffic generated in order for the node to “see” that far. Of course, that would also mean that the network cannot guarantee that all available content will be found. Data outside the horizon of a peer will never be discovered.

4.3 Second Generation Systems

4.3.1 Partially Decentralised Systems Analysis

What has been described so far constitutes the first generation of peer-to-peer networks. Due to the very important drawbacks of these approaches a second generation of systems emerged merely as an evolution to deal with these problems.

As it became apparent from the use of the early unstructured systems, there was a need to limit the inefficiencies that came from network flooding. In order to deal with the problem a network hierarchy was introduced in later versions of Gnutella, namely v0.6 (Klingberg and Manfredi, 2002). This protocol is going to be analysed further in this section.

As it was shown earlier the Napster protocol was very efficient with regard to resource locating. The reply of the system was quick and binary in logic. The content either existed or not. The second generation of peer-to-peer systems tried to incorporate this efficiency without loosing the merits of the decentralised approach. Instead of having one central entity the network would dynamically create many. These nodes are called usually “supernodes”, “superpeers” or “ultrapeers” (Chawathe et al., 2003; Karwaczynski and Kwiatkowski, 2005; Risson and Moors, 2004). Each supernode keeps the indexes of all surrounding “leaf” nodes. An illustration of a real Gnutella v0.6 network can be seen in figure 4.3

The reason this assignment of supernodes is dynamic is in order to avoid central

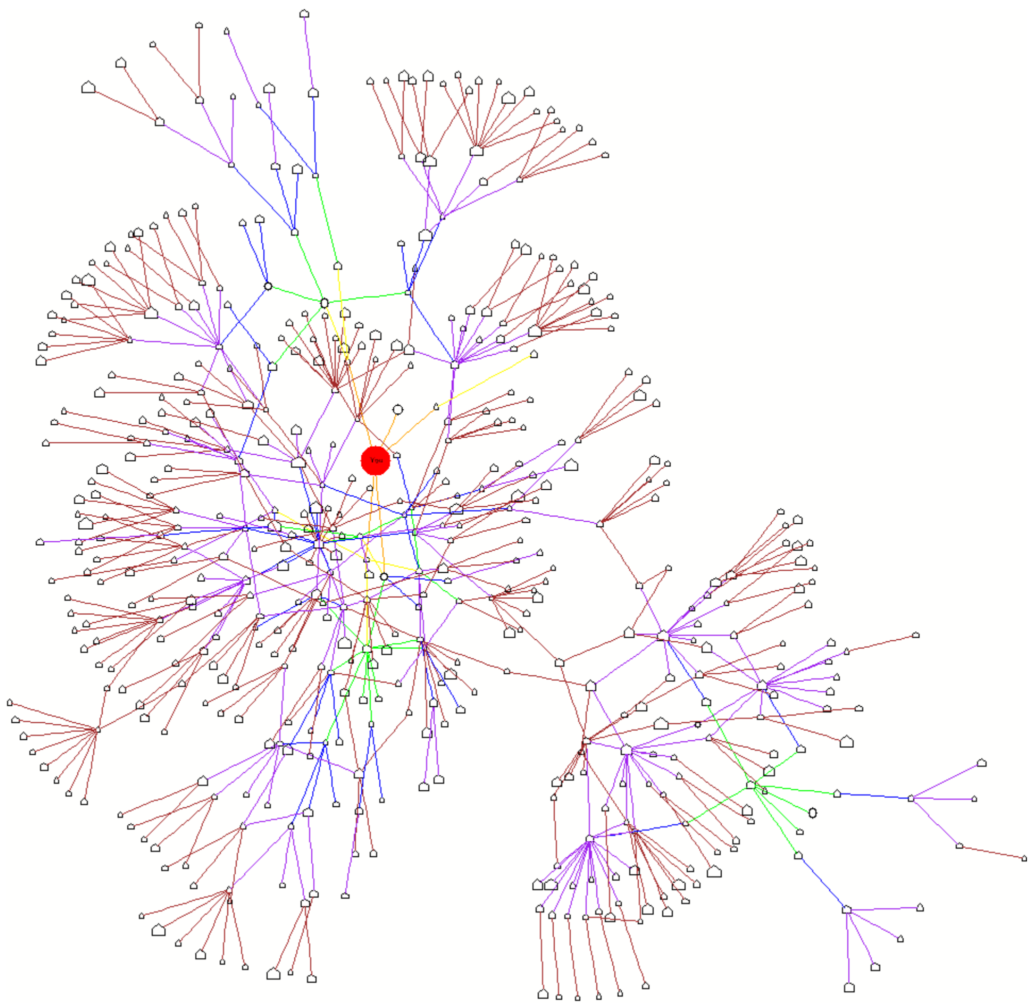

Figure 4.3: A Gnutella v0.6 network. Taken from www.cybergeography.org. This image is licensed under the Creative Commons Licence

points of failure like with the Napster protocol. If a supernode fails the leaf nodes associated with it will be reassigned to another supernode. Despite the introduction of supernodes in Gnutella 0.6 the messaging protocol remained the same for backwards compatibility purposes.

A node that wants to join the network now only has to establish a connection with a supernode and send to it the content that it is sharing with the network. The node then Pings the supernode and receive a response Pong messages from the whole neighbourhood (i.e. all nodes under the supernode). This is to make sure that if the supernode fails the other nodes of the network will still be able to communicate. The original Ping, in accordance with the Gnutella protocol v0.4, is flooded, but this time only to the supernode level.

Similarly, when a node sends a query, the Query message is sent only to the supernode which propagates it only to other nodes in its neighbourhood that actually have the requested content. The supernode also floods the Query to the supernode level and each supernode propagates it to the respective leaf nodes that actually have the content. Therefore, the message is transmitted to the network more efficiently and the requesting node receives QueryHit replies from the nodes that can provide the desired content. Eberspacher and Schollmeier (2005*a*) show how even in small experimental networks this results in less traffic than the 0.4 protocol, while in real more broad networks the difference should be more obvious.

Supernodes are usually selected on the basis of CPU power and available bandwidth even though the exact details may vary in different applications. In particular in Gnutella 0.6 the way this assignment happens is as follows: when a peer joins the network assuming it has enough CPU power it immediately becomes a supernode and forms connections on the supernode level. If after a specific period of time enough leaf nodes have formed a neighbourhood under it then it remains a supernode. Otherwise it becomes a leaf node itself.

Nevertheless, in this way the network now takes advantage of the heterogeneity of peer-to-peer systems (Androutsellis-Theotokis and Spinellis, 2004). Different nodes with different capabilities with assume different roles in the network.

Many more applications nowadays have adopted this approach. To name a few, Kazaa, Skype, Emule and Edutella even though the first two are closed source applications and the exact details of their implementations are not known (*Emule*, 2007; *Kazaa*, 2007; Nejdl et al., 2002; *Skype*, 2007).

4.4 Discussion on Unstructured Peer-to-Peer Networks

Throughout their evolution unstructured peer-to-peer networks have been quite popular. Saroiu, Gummadi, Dunn, Gribble and Levy (2002) showed that in the network of the University of Washington 6.04% and 36.9% of TCP traffic was caused by Gnutella and Kazaa respectively. This popularity of such networks can be explained given a number of factors: Firstly they very successfully handle keyword based queries. Most file sharing applications have a sole purpose of allowing the user to find data -usually based on filenames or descriptions given by the user- using keyword based searches. We believe that the effective way in which unstructured deal with this, could be the most important reason for their popularity.

Secondly, most of these applications are used to share popular content in principle. Popular content means high degree of replication, which as we saw allows such networks to operate more efficiently. Therefore they are successful at providing what is needed.

Furthermore, another important reason for their success is that they are very resilient, one of the reasons being that there need be no proactive efforts for a node to join the network. They are able to handle highly transient peers very well, something not very well supported by structured approaches. In a real network of peers this is very essential, and it is also very important that any failures will not affect the network. Therefore the end user is provided with a stable network infrastructure that s/he can join or leave at any time s/he wishes without any implications.

Of course, the simplicity of such systems has played its role. There were quite a few "flavours" of unstructured systems implemented as it has been shown in this chapter, some more popular than others. Their simplicity allowed developers to produce and adapt solutions more easily to what was requested by the public.

However, there are still many unresolved issues in this category of peer-to-peer systems. One issue that arose even from the first generation systems, as we described in previous sections, is the TTL dilemma (i.e. big TTL=big amount of traffic whereas small TTL=unreachable network). Even in second generation networks where the situation has improved the problem still remains.

Risson and Moors (2004) explain how a different approach could have been used to give a solution to the dilemma: a random walk. Essentially an unduplicated query that wonders around the network until it finds the required content. Of course that would have an enormous impact on retrieval time. A more viable solution that has been suggested is to replicate this query at each peer essentially having a number of walkers. This technique is called random k-walkers. This solution combined with TTL limits can prove more effective than simply flooding the network (Lv et al., 2002).

Another suggestion by Yang and Garcia-Molina (2002) was to use an iterative deepening[1] when searching. Flooding is still used with an increasing TTL each time, until a result is found. This allows the system to not have a default TTL but adapt to the circumstances. If something is found in one hop there is no need to search further. If not, the TTL is increased and search goes further.

However, this still does not solve entirely the problem of content discovery that is not highly replicated in the network. Essentially, this is a very significant problem that has to be addressed: how to provide better and more reliable search facilities that would make the system more scalable. There has been a lot of research on this issue. Chawathe et al. (2003) propose several improvements on Gnutella and introduce Gia a peer-to-peer file-sharing system that exploits the heterogeneity of the network and performs 3 to 5 orders of magnitude better than other unstructured systems of the time.

A very comprehensive analysis of the different search methods that have been proposed can be found at Tsoumakos and Roussopoulos (2006).

[1]Most well known as "expanding ring"

Chapter 5

Structured Systems

The evident success of the first implementations of peer-to-peer systems, induced interest to the field by the research community worldwide. The first "wave" of publications were mostly identificatory, analysing existing peer-to-peer networks. As one would expect, the next step was suggesting ways to improve the existing systems, and even completely new approaches that would perform better than the existing solutions.

As it was explained in the previous chapter, there were significant problems with the existing systems. Many have shown (and continue to do so) ways that these systems could be improved. In fact Risson and Moors (2004) argue that the scientific community may have turned its back a bit to soon to unstructured architectures (more specifically referring to centralised systems) most probably because of the early failure of Napster. However, other research more recently gave birth to a different approach: algorithms based on Distributed Hash Table (DHT), a general and surprisingly simple interface. Such systems guarantee that information that exists in the network *will* be found[1] while keeping the number of hops under a specified limit.

The look-up algorithm is right in the centre of these approaches. In fact, the look-up problem is a centric issue in every peer-to-peer system and provides a good demonstration of the challenges in the design of such networks (Balakrishnan et al., 2003).

In this chapter, we will describe the basic principles of this approach in general and will carry on to discuss some of the most prominent algorithms in this area.

[1]Note the contradiction with unstructured systems that do not generally guarantee data retrieval

5.1 The logic behind DHT's

5.1.1 Different terms

DHT-based networks are most often classified under the the term structured that is also used in this report. However, depending on the perspective of the researcher they have also been described as "Data lookup" systems or systems with semantic-free or distributed indexes (Risson and Moors, 2004; Schmidt and Parashar, 2005). Data lookup because this is basically their core operation (looking up data). The other two classifications have been used with regard to the indexes in such systems. They are both semantic free, in the sense that they are not human readable and distributed in the sense that they are not stored locally (like Gnutella 0.4) or centrally (like Napster). Nevertheless, these classifications just describe different aspects of the same system. The academic community has given much attention to this approach to peer-to-peer systems lately and one could argue that they appear to be one of the most promising and exciting technologies in the field of network computing.

5.1.2 Merits of both worlds

In the previous chapters we discussed the merits and drawbacks of unstructured approaches. Wehrle et al. (2005) describe how DHT's try to keep merits of both centralised and decentralised unstructured systems while avoiding their drawbacks. A centralised system has a search complexity in the order of $O(1)$, since there is a central server replying to search requests, and storage complexity in the order of $O(N)$ where N is the number of items to be stored, as the server needs to be aware of that. A Gnutella 0.4 like system on the other hand, has a storage complexity of $O(1)$ since data and indexes are stored locally, and a searching complexity of $O(N^2)$ as they show. The DHT-based algorithms give us both a storage and searching complexity in the order of $O(logN)$, no matter how big the network grows.

5.1.3 Basic mechanism

Like in unstructured decentralised systems, there are no bottlenecks or central points of failure, as the nodes in the network are all equal, hence peers. However, in DHT's there is the concept of address space onto which data are mapped. Each node is then "responsible" for a portion of this address space. Typically the address space is a very large integer number.

Each data item has an identifier which is unique. This identifier could be assigned by the system or it could be derived from the data itself (Wehrle et al., 2005). Most often the latter is the case: some property of the file is chosen (e.g. its name or the whole binary file) and is hashed by some algorithm which is consistent in the sense that it is collision free like SHA-1 (NIST (2002). See also Karger et al. (1997)). According to this identifier the data item is then mapped to the address space. The exact specifics on how the address space is managed differs from algorithm to algorithm. For example modulo operations could be applied resulting in a ring like in Chord (Stoica et al., 2001).

In any case however, this mechanism maps data items to the predefined address space which is divided in areas of "jurisdiction" for the different nodes. Therefore each node knows what data it is "responsible" for. Based on this infrastructure the very simple interface mentioned in the introduction is provided in the form of two operations: *put(key, data)* and *lookup(key)* where key could be the hashed value of a property of the data. It is worth noting that Balakrishnan et al. (2003) describe that there is only one operation in a DHT, the *lookup(key)*. *put(key,value)* is described as a double operation instead: the node has to perform a lookup to find the relevant node and then send the data. However, most DHT's implement a storage interface as well.

5.1.3.1 Routing and storage

Each node in a DHT keeps information about some other nodes in the system in order to perform routing functions. Which exact nodes it will keep information about depends on the implementation but commonly it has a limited view of the system. Hence, when it receives a message with an identifier which is not mapped into the space it is responsible for, it sends it to one of the nodes it knows about. This is not a random operation, there is a deterministic way that the routing decisions are made.

Moreover, as Wehrle et al. (2005) indicate, there are two possible solutions for storing the data. After it is determined to which part of the address space the data item belongs, either it is literally stored in the network (i.e. at the node responsible for that portion of the space), or a reference to the data is stored. The bandwidth consumption that is avoided in the second approach comes with the cost that once a node leaves the network the data items it offered are lost (at least until it reconnects).

5.1.3.2 Joining, leaving and failing nodes

In order for a new peer to join the network a bootstrap technique is needed to introduce it to the existing nodes of the system. Then the node will have to be assigned a portion of the address space. Depending on the implementation which portion that will be, is decided either arbitrarily by the node, or depending on the system's state. Finally, some other nodes will need to update their routing tables and the node will need to receive the data it is responsible for.

A reverse procedure happens when a node leaves the network. Some approaches demand from the node to send a warning before it leaves in order for the network to readjust sooner than later (i.e. routing updates and replicate the content on the leaving node in some cases). Unfortunately, when a node fails these procedures will have to happen in a reactive rather than proactive basis (which means that data cannot be replicated before the node leaves for example). The operations of the network that would normally be routed through the failed node, will search alternative routes. A way to avoid such "surprises" in some implementations is to check periodically for "dead" nodes and appropriately update the network.

5.2 Chord

The most well known implementation of DHT-based algorithms is Chord (Stoica et al., 2001). In fact according to Google Scholar[1] the original paper has been cited almost 4000 times[2]! Much of this success is due to the simplicity of the algorithm. It is the approach that will be described more analytically in this chapter.

In Chord what we have described as address space is called identifier space as everything -both data items and nodes- in a Chord network is assigned an identifier. More specifically consistent hashing (Karger et al., 1997) is used to assign to each node and data item an identifier. The base hash function used is SHA-1(NIST, 2002). Androutsellis-Theotokis and Spinellis (2004) indicate that consistent hashing tends to balance load in the system as each node receives a -roughly- balanced load of keys. Another advantage is that adding or removing a node results in a fraction of the keys "moving" to new locations in the order or $O(1/N)$.

Typically the identifier of a data item is called a key. Keys are essentially m-bit identifiers which are mapped to an identifier circle modulo 2^m. At the core of Chord

[1]http://scholar.google.com
[2]Last checked January 2007

lies the idea that each key k is assigned to the first node whose identifier is equal or greater than k in the identifier space. That node is called the *successor* of k. The classic representation of a Chord network is a circle (the identifier circle) on which keys and nodes can take values ranging from 0 to $2^m - 1$. In that circle the successor of a key would be the first node clockwise from the key. In figure 5.1 an illustration of an example Chord network can be seen with 6-bit identifiers (i.e. 64 identifiers in total). The network consists of 5 keys and 10 nodes. Keys and their successor nodes are indicated by arrows which map the key to the node.

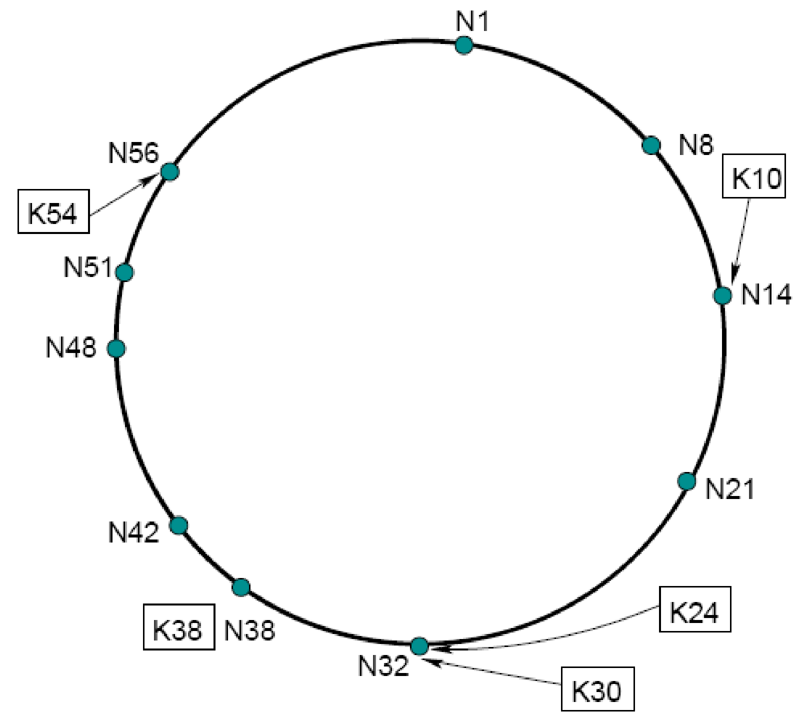

Figure 5.1: A Chord Network illustration. Taken from http://www.inf.u-szeged.hu

5.2.1 Routing in Chord

Given the infrastructure described above a very primitive routing algorithm can be implemented as follows: When a node needs to find a key it essentially wants to find the successor of that key. If the key lies between itself and the next node in the circle (in a clockwise manner) then the problem is solved. Otherwise it forwards the query to the next node. The successor node similarly then determines whether the required key lies between itself and its successor. This recursive algorithm continues until the successor node of the key is found and returned to the original requester. This obviously inefficient algorithm would result in a number of messages linearly analogous to the number of nodes in the network. Its advantage however would be that each node would only need to keep information for the next (successor) node.

In reality the Chord algorithm utilises a much more intelligent routing technique using what is called “finger tables”. Since it is a completely distributed system its node needs to keep its own finger table which constitutes a *partial* view of the whole network. Given an identifier space of m-bit identifiers each node keeps a finger table of m entries at the maximum.

Each row is numbered from 0 to m-1 (0 to 5 in our example). The entry for each row points to the node that has an identifier equal to $n + 2^i$ where n is the identifier of the current node and i is the index number of the row. Table 5.1 shows what the finger table of node N14 would be in our example. Note that according to this mechanism the first pointer always shows the successor node of the current node.

Finger table for node N14		
Index	Target identifier	Successor
0	N14 + 1	N21
1	N14 + 2	N21
2	N14 + 4	N21
3	N14 + 8	N32
4	N14 + 16	N32
5	N14 + 32	N48

Table 5.1: Finger table for node N14 correspoding to figure 5.1

An actual finger table consists of entries with an IP address and port number, the identifier of the target node and possibly some additional information used for bookkeeping.

The hops required for routing when utilising a finger table, is in the order of $O(logN)$ where N is the number of nodes forming the network. In fact, Stoica et al. (2001) show that in average the lookup time is $\frac{1}{2}logN$. This distribution of the view of the network results in rather small amounts of information that each node needs to keep. The size of the finger table is independent of the number of nodes or data items participating in the network and only relates to m (from the $m - bit$ identifier space). As a result, even in a network with an identifier space of 2^{512} locations the size of the table would be 512 entries.

Using finger tables the routing takes a more efficient form. The requester forwards the query to the closest *predecessor* of the key (i.e. a predecessor node with the identifier value closest to the key value) that can be found in its finger table. The request travels from node to node according to this logic until it reaches a node such that the key value lies between that node and its successor. That node then reports its successor node as an answer. Note how this algorithm is not recursive unlike the primitive one described earlier.

5.2.2 Node Joins

In a Chord network each node besides the finger table keeps information about its successor and predecessor nodes. When a new node joins a network it would normally need to update its information regarding the above, but also to inform other nodes which involve the joining node as successor, predecessor or in their finger tables.

More specifically, when a node joins it needs to "choose" some identifier a for itself. In fact, the original Chord protocol allows the node to choose that identifier in any way it chooses, but as Gotz et al. (2005) indicate there have been several proposals in order to restrict this choice according to certain criteria in order for example to exploit network locality. The joining node then queries some already connected to the system node about its own identifier and retrieves its successor node. It then informs its successor about its presence resulting to the successor node updating its predecessor link.

In order to build its finger table it simply by querying for $a + 2^i$ where a is its own identifier and i is the index number of the relevant row of the finger table (range $[0, m-1]$[1]).

Chord has introduced a stabilisation protocol in order to update the successor pointers in node joins. In fact, originally, without this protocol there is no need to have predecessor pointers in the network at all. Predecessor pointers were introduced for the stabilisation protocol to function properly. The way the `stabilise()` function works is as follows: The node n asks its successor s to return its predecessor node p. If p is equal to n then there is nothing to be done. If however, p lies between n and s, that means that p has recently joined the network and n has to update its successor link to point to p and to notify p of being its predecessor.

One would notice that `stabilise()` does not inform a node of its predecessor. Instead a newly joined node b would have to wait for its predecessor a to use `stabilise()`, detect the inconsistencies and inform b of being its predecessor. After that b would copy all keys between itself and its predecessor a, while a would drop these entries. That of course, means that `stabilise()` runs periodically on every node.

[1]m from the m-bit identifier that we have described earlier

At this stage other nodes in the network are not aware of the new node and forward queries and therefore send queries that should be send to this new node to its predecessor instead. The predecessor of course would then forward the message to the new node. This however slows down the procedure linearly with new node joins. The nodes that need to update their finger tables can be determined:

"Node n will become the i^{th} finger of node p *iff*:

1. p precedes n by at least 2^{i-1}
2. the i^{th} finger of node p succeds n"

(Stoica et al., 2001)

Nevertheless, this can be an expensive procedure, especially if many nodes join the network at the same time. Hence, since outdated finger tables have a relatively small impact on routing, instead of performing this for every node join, Chord choses instead the more "relaxed" approach of running `fix_fingers()` periodically.

5.2.3 Node failures/departures

Saroiu, Gummadi and Gribble (2002) note that since the Chord overlay network is not based on the underlying physical network one failure in the underlying network can result in multiple node failures in the overlay topology. In fact, multiple node failures could lead to incorrect replies regarding queries for a key. Gotz et al. (2005) show that it is very crucial to maintain accurate successor information.

Chord deals with node failures by keeping information for r successors instead of just one. The network will only be affected if all r successors of a node fail simultaneously. Moreover, node connections are checked for timeouts and if a finger does not respond, the previous finger is contacted. Using `fix_fingers()` (which can be used specifically for a failed node) the health of the finger tables is maintained.

When a node fails, data that it was responsible for are lost too. Replication can be utilised to avoid data loss. The successor of a node becomes responsible for the data the failed node had under its "jurisdiction". Ideally, an application based on Chord would replicate data on each node to its successor nodes.

On the other hand, a departing node could be treated as a failed one. This obviously would not be the most effective approach as a failure needs to be detected and repaired. A leaving node normally should notify its successor and predecessor and “pass” the data that it is responsible for to its successor.

5.2.4 Chord Implementations

An implementation of the Chord algorithm in C can be found in the project’s webpage[1], where the source code is available.

The Cooperative File System (CFS) is a read-only storage solution built on top of Chord with the design goals of being a robust, efficient and load balanced implementation (Dabek et al., 2001). CFS servers use block storage by providing a distributed hash table called DHash. Clients of the system perceive DHash blocks as the file system. According to the authors’ tests the system proves to be almost perfectly robust and maintain its performance even as many as half of the network’s nodes fail.

5.3 Content Addressable Network

A different approach can be seen in Ratnasamy et al. (2001)’s work for their Content Addressable Network (CAN). The innovational characteristic of CAN is the form of its identifier space. Instead of adopting a single dimensional space like Chord and Pastry, data in a CAN network are mapped in multiple dimensions. This allows for a great improvement in routing compared to other approaches.

Keys and values are mapped to the identifier space using a uniform hash function, like in other approaches. This space is then divided into regions which are assigned to the different participating nodes. However, unlike other networks, nodes in the network are not assigned a specific identifier but are located by the extent of the zone they are responsible for. Data items are normally assigned an identifier which has different numbers of arguments depending on the dimensionality of the address space. For a two dimensional space for example, it would be (x, y).

Representation of the identifier space depends of course, in the number of dimensions. Operations on identifiers are -similarly to chord- performed modulo the largest coordinate for each dimension. Therefore a single dimensional space would be represented as a circle while a two dimensional space’s geometrical representation is a

[1]http://pdos.csail.mit.edu/chord/

torus. Three and more dimensions are represented as an n-torus, which however is not very easy to visualise.

All nodes that participate in the network are assigned a partition of the identifier space and the identifier space is always covered completely by these partitions. In other words, at any given point *the whole* of the address space is assigned to the participating nodes.

An issue that arises is how to create multidimensional identifiers for data items from a single hash value. The simplest solution to this is to divide the hashed value and each portion would represent a coordinate. For example the first 20 bits of a 40 bit value would be x and the last 20 y for a two-dimensional address space. Since these coordinates (x, y) represent a point in that space, the data item is assigned to the node responsible for that partition of the space. Figure 5.2 shows an illustration of how a two dimensional space could be represented in a CAN network. For simplicity it has been represented as a plane instead of a torus.

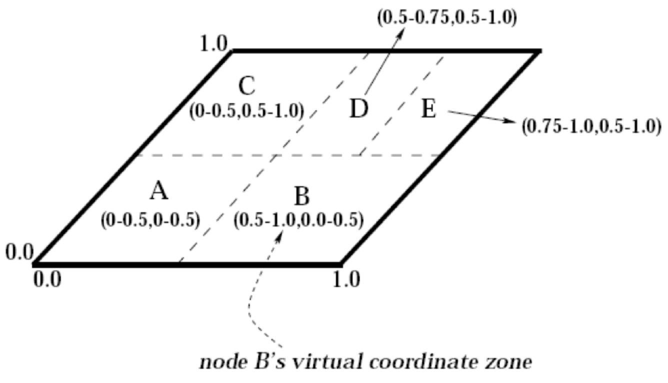

Figure 5.2: Two dimensional identifier space in a CAN

5.3.1 Routing

Routing in CAN networks is based on a simple idea : Each node keeps routing information about its immediate neighbours. This information consists of the IP address and port but also the zone of the neighbour. For two nodes to be considered neighbours their coordinates must overlap in one of the dimensions. For example in figure 5.2 A and B are neighbours while A and D are not.

Each message contains the coordinates of the destination and travels through intermediate nodes. Greedy routing is used and each node forwards the message to the its neighbour with the closest coordinates to the destination. Ratnasamy et al.

(2001) indicate that in d dimensions equally partitioned in n zones the average travel steps required are $(d/4)(n^{1/d})$. Thus, a larger number of dimensions increases the efficiency of the routing algorithm.

5.3.2 Node Arrivals

In order for a node to join the network it needs to find another node already participating in the network, to allocate some zone in the address space and finally to inform the neighbours of its presence. The original proposal for CAN, similarly to Chord and Pastry, does not oblige nodes to allocate zones through a particular mechanism. A bootstrap technique needs to be used in order for a node to find existing nodes in the system.

The joining node then needs to choose randomly a point in the address space and then issue a `Join` request to the system. This request is routed though the network following the standard routing scheme to the node currently covering that random point. The zone is then split in half with one half being assigned to the new node. Finally, the node that previously covered the whole area "passes" the $(key, value)$ pairs located in the new half to the new node.

The new node then needs to build its routing table. That is achieved by exchanging neighbourhood information with the node that was already in the system. The new node gets this information and informs its neighbours for its presence and the existing node adds the new node to its routing table.

An obvious advantage of this technique is that during node arrivals only a small area of the system is affected and needs to take action.

5.3.3 Node failures and departures

Periodically nodes exchange `update` messages to inform their neighbours about their neighbourhood and zone, but most importantly to state that the node is still alive. If a node has not send `update` messages for a long period of time that is interpreted as a failure of the node. The node that detects the failure sets a timer mechanism with the timeout depending on its zone size. Nodes with large zones have bigger timeouts. When the timer fires the node sends `takeover` messages to the neighbours of the failed node. When a node receives a takeover message, if the zone of the node that send the message is smaller than its own, it cancels its timer. If the node sending the message has a larger zone, the receiving node sends its own takeover message.

Effectively through this procedure the node with the smaller area overtakes the area of the failed node.

The node that overtook the area tries, if possible, to merge the two zones otherwise it manages both simultaneously. If a message is routed through a failed node before an overtake takes place the next closest neighbour to the destination address is chosen. If all neighbours have failed an expanding ring flooding mechanism takes place until a live node has been reached.

A node that gracefully leaves the network finds a neighbour to merge its zone and copies its hash table to it so that data is not lost. If such a node cannot be found it assigns its zone to the node with the smallest zone. In order to avoid permanent fragmentation of the system CAN utilises a background process which reassigns zones.

5.3.4 Improvements

In the original paper (Ratnasamy et al., 2001) there is a number of improvements that can be implemented in CAN. Firstly, as we explained earlier, increasing the number of dimensions decreases the number of routing steps required to reach a destination, but also increases the information a node has to keep in its routing tables. It also increases the networks fault tolerance by providing more neighbours in case of a a failure.

Another suggestion is to use multiple realities i.e. copies of the network where a node manages different zones and has different neighbours. Given multiple realities a node can choose the most suitable in terms of the shortest path for a route or in case of failure more paths around failed nodes. Data replication is also achieved.

Another technique includes taking network locality into account by measuring latencies and choosing the neighbour with the smallest latency. Network overloading is another technique: zones are not split unless there are enough nodes. The zone is then managed by multiple nodes. Fewer zones mean shorter paths. Also since data are kept by more than one nodes there is improved fault tolerance. Of course, a neighbour node can then choose to contact the node in the zone with the smallest latency.

Other improvements include using multiple hash functions to assign the same data into multiple nodes, taking the IP network's locality into consideration when forming the network and also uniform partitioning.

5.4 Pastry

Pastry (Rowstron and Druschel, 2001), like Kademlia (Maymounkov and Mazieres, 2002) and Tapestry (Zhao et al., 2001) belongs to a family of peer-to-peer systems based on what is called a "Plaxton tree"(Plaxton et al., 1997) a system that can locate objects in an overlay network and route messages to them using fixed length routing maps.

The address space in Pastry is similar to Chord's in that it is single dimensional and circular and therefore can be represented as a ring. In fact Pastry has quite a few similarities with Chord that will be shown in this section. Data items and nodes are assigned an l-bit identifier. Values are in the range of $[0, 2^l - 1]$[1]. Data item identifiers are called keys and node identifiers are nodeIDs. Pastry identifiers are sequences of digits with base 2^b, where b is typically 4. A node is responsible for keys that are numerically close to them[2].

5.4.1 Routing

Each node in Pastry keeps two kinds of routing information. The routing table, similar to Chord's finger table and the leaf set which contains information about other nodes related to the current node in terms of proximity in the identifier space (i.e. $L/2$ predecessors and $L/2$ successors). There is in fact a third piece of information needed to be kept, the neighbourhood set which contains information about nodes which are close to the current node in terms of *network topology*. This set is not actually involved in the routing process though, but is used to maintain network locality in routing.

The routing table consists of l/b rows and $2^b - 1$ columns. For each row i the entries are the identities of other nodes in the network which share an i-digit prefix with the current node[3]. If there is no node with the appropriate prefix the table entry is left blank. In fact, as a node has limited knowledge for nodes in the distant identifier space the last rows of the routing table are almost always not complete. Actually, $log_{2b}N$ rows are completed where N is the number of nodes in the network.

To route a key, the procedure goes as follows: If the key is within the range of the nodes leaf set, then it is sent to the node in the leaf set numerically closer to the key. If not, the key is sent to the node in the routing table whose identifier has at least

[1] l is normally 128

[2] Notice the difference with Chord where keys were assigned to the *successor* node

[3] The node who's routing table we are talking about

one extra digit[4] in common with the key than the current node (i.e. if the current node shares a 3 digit prefix in common with the key then a node that shares a 4 digit prefix with the key needs to be found). If such a node cannot be found the key is sent to another node that shares the same number of digits with the key as the current node but is numerically closer to it, in the hope that the new node will have a more updated routing table. Using this procedure the network ensures that there will be no loops.

5.4.2 Node Arrivals

A node has to choose a nodeID in order to join the Pastry Network. This normally is a hash value of its IP or public key. The node then with some bootstraping technique gets to know a node A in the existing network. That node is assumed to be close to the new node in terms of network proximity. The new node asks node A to route a join message with value its identifier value. Routing the message eventually reaches node B which is closest numerically to the new node.

All nodes that routed the join message to its final destination sent their routing tables to the new node. This way the new node can build its own routing table. For row 0 (which is independent of the new nodes identifier) entries will typically be filled with entries from node A's row 0. As the join message was routed the nodes it encountered had identifiers increasingly common prefixes with the new nodes identifier. Hence, each row can be filled with information from these nodes respectively. Row i will be filled with information from row i of the i^{th} node the join message encountered. The leaf set of the new node is a copy of B's leaf set as it is the closest node to the new node numerically. Similarly, the neighbourhood set is copied from A which is close to the new node physically. Finally, the new node sends a copy of its routing table to all nodes found in its rows. These nodes then update their tables accordingly.

Notice how when a node joins the network all updates are actively take place in contrast with Chords lazy algorithm. That results in a more consistent network but it is of course more costly. The arrival of a new node involves updates in a rather small number of nodes which means that in the case multiple nodes join simultaneously there will be little chance of overlapping changes. In fact, when the new node sends its newly build routing table to the involved nodes it contains the original time stamps that it received from the nodes when it joined. If there is a mismatch in the time stamps, the join procedure would start over.

[4]from more significant to less significant

5.4.3 Node departures/failures

In Pastry departures and failures of nodes are treated the same way since the protocol can maintain uncorrupted routing information when a node fails. However, there are indications that there could be benefits by utilising departure procedures like the ones suggested in Chord (Gotz et al., 2005).

In order to repair a routing table entry j in row i the node needs to contact another node referenced in that row and "ask" for an entry in row i of the referenced node. If one cannot be found it asks another node and so on. If no appropriate entry can be found like this, then nodes in row $i - 1$ are contacted. It can be shown that there is a high probability that appropriate entries will be found using this procedure.

In the case of a failure in the leaf set other nodes listed in the set are contacted and appropriate entries from their leaf sets are replacing the failed node. In order for the procedure to be unsuccessful L/2 nodes need to fail simultaneously where L is the size of the leaf set. This can be minimised with appropriate values of L.

A similar procedure is used for the replacement of nodes in the neighbour set. However, in this case there needs to be a periodic check of the nodes in that set as they are not used for routing purposes regularly. If a node fails the local node replaces it with entries from neighbour sets from other nodes in its neighbourhood.

5.4.4 Routing efficiency

Pastry improves its routing algorithm on two levels. Firstly, by minimising the number of hops for routing and secondly by exploiting network locality to minimise latencies. In fact, it can be shown that the complexity of routing in Pastry is in the order of $O(log_{2b}(N))$. Increasing b would result in faster routing but again in the expense of having to store more routing information in each node.

Furthermore, network locality is exploited for reduced latency. During routing there is normally a choice among a number of nodes that satisfy the criteria described so far. Choosing the node which is in terms of network locality closer reduces routing lengths.

5.5 Discussion

In this chapter we have described three of the most prominent implementations of the DHT abstraction, all of which appeared in 2001. Each is representative for a different category of approaches. Risson and Moors (2004) sketche Pastry, Chord and CAN as a characteristic example of Plaxton trees, rings and tori respectively.

5.5.1 Comparison

Androutsellis-Theotokis and Spinellis (2004) notes that these systems can be compared in terms of size of routing information kept by network nodes, how well they perform searching and retrieving information from the network and how flexible they are in "churn"[1]. However, the last two are correlated and one could argue that they are effectively a very similar comparison in the sense that the number of changes that need to be done in the network, when a node joins or leaves the network, is directly related to the amount of information other nodes keep. Indeed, routing latency versus maintenance costs is a classic trade-off in these systems. Parameter b in Pastry for example is used for adjusting this balance.

As it was described in the first section of this chapter, DHT-based approaches try to combine "merits of both worlds" in terms of maintenance and searching, referring to centralised and decentralised unstructured architectures. In fact they do manage to keep their promise by staying somewhere in the middle: table 5.2 shows how the algorithms perform in comparison:

DHT system	Routing hops	Size of the routing table (node state)
Chord	$O(logN)$	$O(logN)$
CAN	$dN^{\frac{1}{d}}$	$O(dr)$
Pastry	$O(logN)$	$O(logN)$

Table 5.2: DHT-based systems comparison

N is the number of nodes participating in the system and d is the number of dimensions in the case od CAN. CAN distinguishes itself in both criteria but mostly because the routing information need to be kept are independent of the number of nodes in the system. In fact, Ratnasamy et al. (2001) note that the algorithm can match Plaxton's properties if d is set to $log_2(N)/2$. Androutsellis-Theotokis and Spinellis (2004) argue that churn probably proves more costly for Chord.

[1]A term used to describe continually arriving and departing nodes

5.5.2 The "Keyword Queries" Challenge

Moreover, it is arguable that the biggest drawback of DHTs is the fact that by design, they do not support keyword queries. Unstructured systems have been able to do so since the very beginning. In fact, they are build around this procedure. Obviously, that is a major reason for their success in file sharing applications where keyword queries, since file names are used, are not only essential, but also efficient. In contrast, DHTs are designed to work with exact identifier searches in the address space of the network. In order to find a value (data item) the user needs to, somehow, know its identifier. This is an open question in DHT research and there have been a number of papers trying to deal with the issue.

The classic approach to this problem is using *inverted indexes* an idea that derives from search engines. A list is maintained with all the words that characterise items in the network/database. A *posting list* is kept binding these words with identifiers of the documents relevant to them. In a query then, intersections or unions of sets with the relevant documents can be created based on the terms given in the query. The point is to implement this on top of a DHT. Overnet[1] implements a similar mechanism on top of the DHT Kademlia (Maymounkov and Mazieres, 2002). However, this proves not so straight-forward.

Harren et al. (2002) propose a system where the underlying network is responsible for indexing and routing while an extra layer for processing is build on top. Garces-Erice et al. (2004) created a hierarchically indexed system which is capable of answering user keyword searches, even with incomplete information -at a higher cost-. Data is stored on one or a few nodes. The hierarchical indexes that maintain information about the date are distributed in the network.

5.5.3 Other Challenges

However, Chawathe et al. (2003) built Gia, an unstructured system built on top of Gnutella in order to introduce techniques to make Gnutella-like systems more scalable and report many improvements. They argue that for a number of reasons they did not choose a DHT approach. Notably, they agree with other research on the disturbing effect of high churn rates in DHT's and mention that while in a DHT churn involves $O(log(N))$ repair operations for a failure. Correspondingly, in an unstructured network a node failure involves just the bootstraping procedure in order for the node to re-join the system.

[1]http://www.overnet.com

They carry on to indicate the keyword search problem, arguing that even though some of the problems involved may be addressable, still in DHTs there is a lot of effort involved with arguable results while unstructured systems perform this procedure effortlessly. Moreover, the fact that as we described earlier in this report, in unstructured networks content that is not higlhly replicated is likely not to be found, is compared to DHT's ability to find with certainty whatever is available in the system. "Most queries are for hay, not needles" as they put it, and according to their experiments 80% of all queries was about content that was replicated at least by 80 nodes (Chawathe et al., 2003). In any case, this comparison has no point if the keyword problem in DHTs is resolved.

Nevertheless, research papers sometimes reach contradictory results. An example of this in the case of Pastry : Rhea et al. (2004) use Bamboo, a DHT implementation, which uses Pastry as its routing algorithm to reach to the conclusion that DHTs do not perform well under high churn rates, after arguing that they should -at least- be able to handle churn rates equal to those observed in networks such as Kazaa. DHT performance, according to the paper, is affected by three main drawbacks: the way neighbours are chosen, timeouts in the lookup procedure and finally the fact that reactive rather than proactive recovery of nodes is utilised.

Nevertheless, Castro et al. (2004*a*) in their study of performance and dependability of structured systems conclude that "previous concerns are unfounded". They present a new implementation of Pastry, called MSPastry, to achieve a smaller number of routing hops while keeping maintenance costs in an acceptable level, therefore increasing the dependability of the system in high churn rates.

The above two, keyword searches and network resilience in high churn rates are the most "famous" issues. Nevertheless, not the only open questions. Wehrle et al. (2005) show how in a DHT system, despite normally using uniform hash functions to balance load equally among nodes, there are some problems. A node may manage data items that are particularly popular or it may happen that the address space that it has been assigned to manage is very large (as we saw it can happen in CAN for example). Furthermore, it may also be the case that a node manages an address space with a larger number of data items mapped to it. Hence, load balancing mechanisms are needed despite the precaution of the uniform hash function. Things can get even more complicated when keyword searching techniques are implemented on top of the DHT.

Moreover, Keleher et al. (2002) discuss DHT-based peer-to-peer networks which they characterise as "virtualised" in the sense that an overlay network based on a hash function is build which is normally not related to the underlying's network locality. In fact, they actually indicate how this exact property of DHTs -destroying network locality- results in latencies as two nodes close to each other physically are normally separated in a DHT. In effect such systems do not use *proximity routing*. Of course, as it was shown earlier, there have been tries to correct this and Pastry even has mechanisms to exploit network locality as described in the original paper. However, this virtualisation also discards application level information normally described in hierarchies. Data close in the hierarchy are normally related in some way and representing data solely as keys has the effect of discarding this information.

Finally, as Saroiu, Gummadi and Gribble (2002) indicate, the fact that structured systems are build in a deterministic manner, allows for malicious nodes to exploit this by placing node identifiers in the network in such a way that they could “spy” and possibly intercept on network traffic. Balakrishnan et al. (2003) describe how this is a challenge that needs to be faced. There are techniques that can be employed to detect malicious acts (for example Pastry employes certificate to authenticate nodes). However, the balance between allowing the system to be open to new nodes and the need to keep nodes responsible for their actions has to be reached.

There are numerous approaches to DHT-based networks and it is not very likely that there will be a settlement to a small number of designs in the near future. In fact, comparisons of existing DHT approaches have been discouraged from the overwhelming diversity in the field (Risson and Moors, 2004).

Overall, DHTs are considered to be a very important development in peer-to-peer research. Their elegance and the simplicity of the DHT abstraction which allows numerous applications to built on top of a DHT are key reasons for this. They are much more efficient in many ways than earlier peer-to-peer approaches. Therefore, there is much potential in this approach if the issues analysed here find an acceptable solution.

Chapter 6

Simulation

As it was shown in previous chapters peer-to-peer networks can grow to very large scales and in the light of third generation DHT-based systems can become quite complex. Mathematical analysis of such networks would therefore become hard. In order for the analysis to remain practical and produce results simplifications could be assumed that would “harm” the overall view of the experiment. Another approach usually undertaken in similar situations is that of simulation. Models of the system are produced and studied further by collecting statistical data while the simulation is running. While this by no means can replace mathematical analysis it is a valuable tool. Ideally the two methods should be combined and the validity of the results proved with experiments on the actual system.

In this chapter we explore what a general purpose peer-to-peer simulator is and provide an overview of the available tools currently.

6.1 A generic approach

There have been several implementations of peer-to-peer simulators with varying characteristics. Ting (2003) indicates that sometimes simulators are built with a special purpose in mind while obviously a general purpose simulator would be more useful. He then carries on to sketch the architecture such an approach should have.

Naiken et al. (2006) describe the requirements a general purpose peer-to-peer simulator should meet. The main points include:

- Simulation Architecture: There are two different architectures for simulation, discrete event and query-cycle. In query cycle Schlosser et al. (2003) describe how in each cycle the program iterates through each nodel. The cycle is over when all queries have been given a satisfactory response. The discrete event architecture utilises a scheduler to synchronise the messages exchanged among the nodes.
- Behaviour simulation : Ability to simulate behaviours such as node failures and churn
- Usability, in terms of a good interface for the user
- Good documentation is essential in order for other researchers to be able to expand on the given work
- Ability to produce good statistics for various aspects of the simulated system
- Ability to avoid system limitations by facilitating distribution of the simulation on more than on one computers
- Ability to scale in terms of the number of nodes that can be simulated in a network.
- Ability to simulate the underlying network for more realistic results. This ideally should be done in a adjustable way depending on the focus of the experiment.

6.2 Overview of existing tools

There is a variety of tools that can be used for simulating peer-to-peer systems. General network simulators like NS-2[1] can also be used for this purpose but we will focus on peer-to-peer specific tools in this report. In fact, network simulators have the advantage that they simulate the underlying network in more detail but as one can imagine this results in limited scalability in terms of the number of nodes the simulation can involve.

[1]http://www.isi.edu/nsnam/ns/

6.2.1 P2PSim

P2PSim[2] uses the discrete event architecture. Developed by MIT's Parallel and Distributed Operating Systems Group, the same group behind Chord, it focuses on simulation of DHT-based systems. It actually comes with three different implementations of Chord which differ to one another in terms of size (or existence) of their finger tables. Its developers claim that they have simulated networks with 3000 nodes. Distributed simulation is not supported on the current version. Node failures and churn can be included in the simulation by using appropriate scripts. Unlike other simulators it does also include the ability to model the underlying network in sufficient detail providing more accurate results. The language used is C++.

6.2.2 PeerSim

PeerSim[1] developed in Java is composed of two engines: a cycle based one and a discrete event driven. The event based one is more realistic but less efficient. It can be used to simulate both structured and unstructured systems while according to its developers it has been used to simulate up to 1,000,000 nodes. It does support transport layer simulation even though not in much detail as its focus is performance on the overlay network. Its implementation allows for most of its components to be replaced easily but with a focus on graph-like protocols this replacement is not always easy Naiken et al. (2006). Moreover,there are mechanisms that can be used to model churn adn node failures. Finally, it does not support distributed simulation.

6.2.3 GPS

GPS[2] is also written in Java. Like the above simulators it does not support distributed simulation. It comes with built-in support for BitTorrent but can be used to model other structured and unstructured approaches. It tries to combine the efficiency of message level simulators with the accuracy of more complete network simulators by partially modelling the underlying network. Moreover, there are mechanisms to facilitate simulation of churn. Finally the simulation is discrete event driven.

[2]http://pdos.csail.mit.edu/p2psim/

[1]http://peersim.sourceforge.net/

[2]http://www.cs.binghamton.edu/ wyang/gps/

6.2.4 Query-Cycle Simulator

The Query-Cycle Simulator as its name suggests utilises the query-cycle architecture for simulation (Schlosser et al., 2003). The constructed model is based on real world observations in order to model peer content (type, and quantity), peer behaviour and network parameters. However, this project is inactive.

6.2.5 Neurogrid

Neurogrid[1] is a discrete event driven simulator written in Java. Originally written to simulate the Freenet, Gnutella and Neurogrid protocols it can now be used to simulate both structured and unstructured algorithms. Its authors claim that it has been used to simulate up to 300,000 nodes in a network. Node behaviours such as failures are not supported in the current version of the program. Finally, the underlying protocol cannot be modelled.

6.2.6 PlanetSim

PlanetSim[2], written in Java, is a simulation framework for overlay networks in general as stated in the application's website. A very distinctive feature of PlanetSim is the fact that it has been built to implement the common API for peer-to-peer overlays described in Dabek et al. (2003). This common API introduced to allow "independent innovation in overlay protocols" unifies the key abstractions that can be built on top of structured overlays. The simulator supports both structured and unstructured algorithms. Furthermore, in the extensive documentation available for this platform it is described how the simulator's architecture is comprised of three layers : The application layer which communicates with the common API with the Overlay network layer and finally the network layer. Hence, an application layer service can be used on different overlay algorithms and the overlay can run on different underlying networks. It should be noted however that the available underlying network simulation is rather limited but due to the openness of the platform this can be amended. Moreover, there are mechanisms to simulate churn and node failures. Nevertheless, distributed simulation is not supported even though it is listed as an item in the TODO list published on the program's website. Finally, networks with up to 100,000 peers are supported.

[1]http://neurogrid.net/php/index.php
[2]http://planet.urv.es/planetsim/

6.2.7 Overlay Weaver

Overlay weaver[1], written in Java is meant to be an “overlay construction toolkit” i.e. it is mainly meant to allow for development of peer-to-peer protocols. It can also be used as a simulator and comes with implementations of Chord, Kademlia, Pastry and Tapestry among others. However, it does not provide any simulation of the underlying network. Unlike most of the simulators described above it allows for distributed simulation. According to its authors it has been used to simulate networks consisting of up to 4000 nodes. Churn and node behaviours can be simulated using a script mechanism. Finally, it can only be used to simulate structured algorithms.

For a more comprehensive comparison of the above tools also see Naicken et al. (2006).

6.2.8 Conclusion

The most prominent tools for simulating peer-to-peer systems were introduced and according to the approach explained in section 6.1 their characteristics were overviewed. All systems are available to download along with their source code and therefore one can make modifications to suit the needs of his/her research. Nevertheless, the most promising application appears to be PlanetSim due to its plethora of features and the very clean and generic architecture that it uses.

[1] http://overlayweaver.sourceforge.net/

Chapter 7

Conclusion

Peer-to-peer systems introduce a new approach to network computing. It is a move from the classic client-server model to a network of peers where all participants act as both clients and servers. It is essentially a move to cooperation instead of coordination, to incentives instead of control (Steinmetz and Wehrle, 2005).

The promise of peer-to-peer comes as a solution to the ever-growing demands of future network applications. These self-organising networks provide us with the ability to share resources such as storage, content and CPU cycles. Whether the challenges presented in this report will be successfully faced, it remains to be seen. Nevertheless, by studying the various aspects of such systems and looking at their evolution, one can only have an optimistic view of what is yet to come.

7.1 Further extending this project

It was the purpose of this project to study the different aspects of the field and produce a critical analysis. The biggest challenge was the fact that there has been a overwhelming amount of research produced over the last 6 years on peer-to-peer computing. Due to the wide diversity of the various approaches in the different aspects of such systems, taxonomising this knowledge is not a trivial task. For example, as explained in chapter 3, even for the classification of peer-to-peer implementations there have been various approaches, depending on the point of view of the researcher. Moreover, since this is a "young" child of distributed computing, there have not yet been many textbook publications which would help establish a more structured learning approach to peer-to-peer systems.

However, an analysis of background information was presented in chapter 2, which allowed for the following chapters to classify and further analyse the different generations of peer-to-peer systems and the challenges they face. Finally, an overview of the available simulation tools has been provided.

Nevertheless, the list of "TO DO" items for further extending this project is endless. In fact, there needs to be a decision on which direction should be followed. Some suggestions follow :

- As it can be understood, the incentive behind the analysis of available simulators in chapter 6 is to prepare some more practical work to be done. This can range from experimenting with different implementations of DHT-based protocols by setting up simulations to adding features to the existing tools. An example would be to try to implement application layer multicasting on a peer-to-peer simulation. These experimentations would have the purpose of familiarising with the more practical aspects in order to eventually produce some more original work.
- Further studying of more peer-to-peer implementations such as BitTorrent and a critical analysis of why it has been so successful.
- Further studying of current challenges in DHT-based systems.
    - The keyword search challenge appears as a very interesting candidate.
- Focus on different aspects of the system such as security (authentication, authorisation, malicious attacks etc)
- Further studying of storage systems built on top of peer-to-peer infrastructures.
- Study the relationship with Grid computing more analytically and future possibilities on how the two technologies could combine their forces.

Bibliography

Androutsellis-Theotokis, S. and Spinellis, D. (2004), 'A survey of peer-to-peer content distribution technologies', *ACM Computing Surveys* **36**(4), 335–371.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R. and Stoica, I. (2003), 'Looking up data in p2p systems', *Commun. ACM* **46**(2), 43–48.

Barkai, D. (2001), *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*, Intel Press.

Castro, M., Costa, M. and Rowstron, A. (2004*a*), Performance and dependability of structured peer-to-peer overlays, *in* 'DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)', IEEE Computer Society, Washington, DC, USA, p. 9.

Castro, M., Costa, M. and Rowstron, A. I. T. (July 2004*b*), Peer-to-peer overlays: structured, unstructured or both?, Msr-tr-2004-73, Microsoft Research.

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

Christensen, C. M. (1997), *The Innovator's Dilemma*, Collins.

Clarke, I. (1999), A distributed decentralised information storage and retrieval system, Master's thesis, University of Edinburgh.

Clarke, I. (2003), 'Freenet's next generation routing protocol'.
URL: *http://freenetproject.org/index.php?page=ngrouting*

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R. and Stoica, I. (2001), Wide-area cooperative storage with CFS, *in* 'Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)', Chateau Lake Louise, Banff, Canada.

Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. and Stoica, I. (2003), Towards a common api for structured peer-to-peer overlays, *in* 'Proceedings of the 2nd

International Workshop on Peer-to-Peer Systems (IPTPS03)', Berkeley, CA.
URL: *http://pdos.csail.mit.edu/papers/iptps:apis/*

Ding, C. H., Nutanong, S. and Buyya, R. (2005), Peer-to-peer networks for content sharing, *in* R. Subramanian and B. D. Goodman, eds, 'Peer-to-Peer Computing: The evolution of a disruptive technology', Idea Group Publishing, chapter 2, pp. 28–65.

Eberspacher, J. and Schollmeier, R. (2005*a*), First and second generation of peer-to-peer systems, *in* R. Steinmetz and K. Wehrle, eds, 'Peer to peer systems and applications', Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 5, pp. 35–56.

Eberspacher, J. and Schollmeier, R. (2005*b*), Past and future, *in* R. Steinmetz and K. Wehrle, eds, 'Peer to peer systems and applications', Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 3, pp. 17–23.

Eberspächer, J., Schollmeier, R., Zöls, S. and Kunzmann, G. (2004), Structured p2p networks in mobile and fixed environments, *in* 'Proceedings of HET-NETs '04 International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks'.

Emule (2007).
URL: *http://www.emule-project.net/home/perl/general.cgi?l=1*

Foster, I. and Iamnitchi, A. (2003), On death, taxes, and the convergence of peer-to-peer and grid computing, *in* '2nd International Workshop on Peer-to-Peer Systems'.

Franconi, E., Kuper, G., Lopatenko, A. and Serafini, L. (2003), A robust logical and computational characterisation of peer-to-peer database systems, *in* 'Proceedings of the VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)', Berlin, Germany.

Garces-Erice, L., Felber, P. A., Biersack, E. W., Urvoy-Keller, G. and Ross, K. W. (2004), 'Data indexing in peer-to-peer dht networks', *icdcs* **00**, 200–208.

Gnutella (2007).
URL: *http://www.gnutella.com/*

Gotz, S., Rieche, S. and Wehrle, K. (2005), Selected dht algorithms, *in* R. Steinmetz and K. Wehrle, eds, 'Peer to peer systems and applications', Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 8, pp. 95–117.

Harren, M., Hellerstein, J. M., Huebsch, R., Loo, B. T., Shenker, S. and Stoica, I. (2002), Complex queries in dht-based peer-to-peer networks, *in* 'IPTPS '01:

Revised Papers from the First International Workshop on Peer-to-Peer Systems', Springer-Verlag, London, UK, pp. 242–259.

Jabber (2007).
URL: *http://www.jabber.org*

Jovanovic, M. A. (2000), Modeling large-scale peer-to-peer networks and a case study of gnutella modeling large-scale peer-to-peer networks and a case study of gnutella modeling large-scale peer-to-peer networks and a case study of gnutella, Master's thesis, Division of Graduate Studies and Research of the University of Cincinnati Division of Graduate Studies and Research of the University of Cincinnati.

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D. (1997), Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web, *in* 'STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing', ACM Press, New York, NY, USA, pp. 654–663.

Karwaczynski, P. and Kwiatkowski, J. (2005), Analysis of overlay network impact on dependability, *in* 'HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 9', IEEE Computer Society, Washington, DC, USA, p. 290.3.

Kazaa (2007).
URL: *http://www.kazaa.com/us/index.htm*

Keleher, P. J., Bhattacharjee, B. and Silaghi, B. D. (2002), Are virtualized overlay networks too much of a good thing?, *in* 'IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems', Springer-Verlag, London, UK, pp. 225–231.

Kim, H. (2001), P2p overview, Technical report, Korea Advanced Institute of Science and Technology.

Klingberg, T. and Manfredi, R. (2002), The gnutella protocol specification v0.6, Technical report.
URL: *http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html*

Li, J., Loo, B. T., Hellerstein, J. M., Kaashoek, F. M., Karger, D. R. and Morris, R. (2003), *On the Feasibility of Peer-to-Peer Web Indexing and Search*, Springer-Verlag.
URL: *http://dx.doi.org/http://www.springerlink.com/content/drdkdgvlgnw0ljkh*

Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S. (2002), Search and replication in unstructured peer-to-peer networks, *in* 'ICS '02: Proceedings of the 16th international conference on Supercomputing', ACM Press, New York, NY, USA, pp. 84–95.

Maymounkov, P. and Mazieres, D. (2002), Kademlia: A peer-to-peer information system based on the xor metric, *in* 'IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems', Springer-Verlag, London, UK, pp. 53–65.

Minar, N. and Hedlund, M. (2001), A network of peers : Peer-to-peer models through the history of the internet, *in* A. Oram, ed., 'Peer-to-Peer : Harnessing the power of disruptive technologies', O'Reilly & Associates, Inc., chapter 1, pp. 3–20.

Mischke, J. and Stiller, B. (2004), 'A methodology for the design of distributed search in p2p middleware', *Network, IEEE* **18**(1), 30–37.

Naicken, S., Basu, A., Livingston, B. and Rodhetbhai, S. (2006), A survey of peer-to-peer network simulators, *in* 'Proceedings of the 7th Annual Postgraduate Symposium (PGNET 2006)'.

Naiken, S., Basu, A., Livington, B. and Rodhetbhai, S. (2006), Towards yet another peer-to-peer simulator. Proceedings: Fourth International Working Conference of Performance Modelling and Evaluation of Heterogeneous Networks.
URL: *http://www.comp.brad.ac.uk/het-net/tutorials/P37.pdf*

Napster (2007).
URL: *http://www.napster.com/*

Nejdl, W., Siberski, W. and Sintek, M. (2003), 'Design issues and challenges for rdf- and schema-based peer-to-peer systems', *SIGMOD Rec.* **32**(3), 41–46.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M. and Risch, T. (2002), Edutella: A p2p networking infrastructure based on rdf, *in* 'Proceedings International WWW Conference(11)', Honolulu, Hawaii, USA.

NIST (2002), Secure hash standard, Technical report, National Institute of Standards and Technology.

Plaxton, C. G., Rajaraman, R. and Richa, A. W. (1997), Accessing nearby copies of replicated objects in a distributed environment, *in* 'SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures', ACM Press, New York, NY, USA, pp. 311–320.

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

Rhea, S., Geels, D., Roscoe, T. and Kubiatowicz, J. (2004), Handling churn in a DHT, *in* 'In Proceedings of USENIX Annual Technical Conference'.

Risson, J. and Moors, T. (2004), Survey of research towards robust peer-to-peer networks: Search methods, Technical Report UNSW-EE-P2P-1-1, University of New South Wales.

Roussopoulos, M., Baker, M., Rosenthal, D. S. H., Giuli, T., Maniatis, P. and Mogul, J. (2003), '2 p2p or not 2 p2p?'.
URL: *http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0311017*

Rowstron, A. I. T. and Druschel, P. (2001), Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *in* 'Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg', Springer-Verlag, London, UK, pp. 329–350.

Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D. and Levy, H. M. (2002), An analysis of internet content delivery systems, *in* 'OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation', ACM Press, New York, NY, USA, pp. 315–327.

Saroiu, S., Gummadi, P. K. and Gribble, S. D. (2002), Exploring the design space of distributed and peer-to-peer systems: Comparing the web, triad, and chord/cfs, *in* 'IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems', Springer-Verlag, London, UK, pp. 214–224.

Schlosser, M. T., Condie, T. E. and Kamvar, S. D. (2003), Simulating a file-sharing p2p network, *in* 'Proceedings of Workshop on Semantics in Peer-to-Peer and Grid Computing'.

Schmidt, C. and Parashar, M. (2005), Peer-to-peer information storage and discovery systems, *in* R. Subramanian and B. D. Goodman, eds, 'Peer-to-Peer Computing: The evolution of a disruptive technology', Idea Group Publishing, chapter 4, pp. 79–112.

Schoder, D., Fischback, K. and Schmitt, C. (2005*a*), Core concepts in peer-to-peer networking, *in* R. Subramanian and B. D. Goodman, eds, 'Peer-to-Peer Computing: The evolution of a disruptive technology', Idea Group Publishing, chapter 1, pp. 1–27/.

Schoder, D., Fischback, K. and Schmitt, C. (2005*b*), Distributed hash tables, *in* R. Steinmetz and K. Wehrle, eds, 'Application Areas', Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 4, pp. 25–32.

Scholl (2001), 'Opennap'.
URL: *http://opennap.sourceforge.net/*

Schollmeier, R. and Kunzmann, G. (2003), 'Gnuviz - mapping the gnutella networks to its geographical locations', *Praxis der Informationsverarbeitung und Kommunikation (PIK)* **26(2)**, 74–79.

Shirky, C. (2000), ‘What is p2p ... and what isn’t’.
URL: *http://tim.oreilly.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html*

Skype (2007).
URL: *http://www.skype.com*

Steinmetz, R. and Wehrle, K. (2005), What is ”peer-to-peer” about?, *in* R. Steinmetz and K. Wehrle, eds, ‘Peer to peer systems and applications’, Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 2, pp. 9–16.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H. (2001), Chord: A scalable peer-to-peer lookup service for internet applications, *in* ‘SIGCOMM ’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications’, ACM Press, New York, NY, USA, pp. 149–160.

Taylor, I. J. (2005), *From P2P to Web Services and Grids*, Springer-Verlag London Limited.

The Gnutella Protocol Specification v0.4 (2001).
URL: *http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf*

The SETI@HOME project (2007).
URL: *http://setiathome.ssl.berkeley.edu/*

Ting, N. S. (2003), A generic peer-to-peer network simulator, *in* ‘Proceedings of the 2002-2003 Grad Symposium, CS Dept, University of Saskatchewan’.

Tsoumakos, D. and Roussopoulos, N. (2006), Analysis and comparison of p2p search methods, *in* ‘InfoScale ’06: Proceedings of the 1st international conference on Scalable information systems’, ACM Press, New York, NY, USA, p. 25.

Verma, D. C. (2004), *Legitimate Applications of Peer-to-Peer Networks*, John Wiley & Sons, Inc.

Wehrle, K., Gotz, S. and Rieche, S. (2005), Distributed hash tables, *in* R. Steinmetz and K. Wehrle, eds, ‘Peer to peer systems and applications’, Vol. 3485 of *Lecture Notes in Computer Science*, Springer-Verlag, chapter 7, pp. 79–93.

WinMX (2007).
URL: *http://www.winmx.co.uk/*

Yang, B. and Garcia-Molina, H. (2001), Comparing hybrid peer-to-peer systems, *in* ‘VLDB ’01: Proceedings of the 27th International Conference on Very Large Data Bases’, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 561–570.

Yang, B. and Garcia-Molina, H. (2002), Efficient search in peer-to-peer networks, *in* 'Proceedings of the 22nd International Conference on Distributed Computing Systems'.

Yeo, C. K., Lee, B. S. and Er, M. H. (2004), 'A survey of application level multicast techniques', *Computer Communications* **27**(15), 1547–1568.

Zhao, B. Y., Kubiatowicz, J. D. and Joseph, A. D. (2001), Tapestry: An infrastructure for fault-tolerant wide-area location and, Technical report, University of California at Berkeley, Berkeley, CA, USA.

Bibliography style used : "agsm" of the Harvard family

Appendix A

JXTA report

Appendix B

Mini-project declaration

Appendix C

Statement of Information Search Strategy

C.1 Literature search

C.1.1 Forms of literature

The important categories are (in order):

- Conference Papers
- Journal Articles
- Books
- Technical Reports

C.1.2 Geographical/Language coverage

Most work is expected to originate from Europe and North America but is not limited to these two areas. The preferred language is English but papers in Greek can also be read. However, it is unlikely that Greek language papers will be found as publications from Greek Universities are also expected to be in English.

C.1.3 Date restrictions

Peer-to-peer is a very recent advancement in distributed computing and therefore papers dating at the earliest to 1999 should be searched. Most useful papers will be from 2001 and on. Earlier publications can are acceptable when searching for basic ideas exploited by peer-to-peer systems.

C.1.4 Search Technique

Due to the vast amount of publications on the field recently, arbitrary searches will not probably return satisfactory results. Even searches for specific items are likely to return large numbers of publications of varying quality. The best technique in order to overcome this difficulty is to choose key papers and systematically search for papers that are linked to them (either papers referenced in the key papers or papers that cite the key ones). The papers selected for this should be original proposals for the three most prominent DHT-based systems (CAN, Chord, Pastry). Also survey papers are very useful.

Nevertheless, appropriate search tools include:

- IEEE Explore
- The ACM Digital Library
- Google Scholar
- Index to theses

C.1.5 Search Terms

For the various aspects of this report the only constant keyword is *p2p*. Other than that depending on the issue examined other keywords include: *structured, unstructured, gnutella, napster, chord, simulators, load balancing*

An example search statement would be : `p2p AND structure*`. This has to be refined according to the returned results and the combination of issues involved.

C.2 Evaluation

Given the nature of the subject of this project all generic (and not so generic) queries returned large amounts of results. Refined searches for more specific issues returned somewhat more satisfactory confined results. Google scholar's feature that displays which papers cite the current paper proved very useful. ACM and IEEE indeed were very valuable resources. IEEE was more efficient for the more technical articles. Index to theses did not prove to be very helpful, but this is probably because of the generic nature of this project.