Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking

Semester Thesis
presented by

Robin Jan Maly
ETH Zurich, Switzerland

Supervisor:
Jan Mischke,
Pascal Kurtansky,
Prof. Dr. Burkhard Stiller
of the
Computer Engineering and Networks Laboratory

March 2003

Abstract

Der Trend der Anwendung von Peer-to-Peer Applikationen in vielen Bereichen der Informations- und Kommunikationstechnologie ist in den letzten Jahren verstärkt worden. Speziell im Bereich des “File sharing” haben Anbieter wie Napster, Gnutella oder Kazaa grosse Bekanntheit erlangt. Allgemein wird behauptet, dass Peer-to-Peer Systeme Client-Server Systemen speziell in Bezug auf ihre Kosten überlegen sind. Daher hat die folgende Arbeit mehrere Ziele.

Zum einen soll sie Unterschiede zwischen Peer-to-Peer und traditionellen Client-Server Systemen aufzeigen und anhand eines generischen Ansatzes feststellen, ob es möglich ist, alle Client-Server Systeme komplett durch Peer-to-Peer Systeme zu ersetzen.

In einem zweiten Schritt soll ein Modell entwickelt werden, welches eine Aussage zulässt, ob Client-Server Systeme sich besser zur Anwendung eignen oder Peer-to-Peer Systeme. Das Modell soll dabei speziell auf ökonomische Gesichtspunkte, welche verschiedene Perspektiven umfassen, Rücksicht nehmen.

Peer-to-Peer Systeme haben in vielen Bereichen Verwendung gefunden. Der Schwerpunkt für die Erstellung eines ökonomischen Modells wird in dieser Arbeit jedoch aufgrund ihrer weiten Verbreitung und Nutzung auf “File sharing” Systeme gelegt.

Dabei werden typische Vertreter von “File sharing” Systemen untersucht. Die untersuchten Systeme sind Gnutella, als Vertreter eines puren Peer-to-Peer Ansatzes, Napster als Vertreter eines hybriden Ansatzes und FTP als Vertreter des Client-Server Ansatzes.

Die Schlussfolgerungen, welche aus den Untersuchungen gezogen werden konnten, sagen aus, dass Peer-to-Peer aus einer Gesamtkostensicht in den meisten Fällen traditionellen Client-Server Systemen tatsächlich überlegen ist. Es konnte festgestellt werden, dass besonders hybride Peer-to-Peer Systeme aus Kostensicht sehr effektiv sind. Allerdings muss diese Tatsache mit Nachteilen in Bezug auf Datenverfügbarkeit, Konsistenz der Daten und Übertragungsleistung (Bandbreite) wieder relativiert werden. Es konnte desweiteren ein “Spannungsfeld” zwischen den verschiedenen Interessensgruppen identifiziert werden. Das “Spannungsfeld” sagt aus, dass quasi jede Interessensgruppe, seien es Clients, ISPs oder Server Betreiber aus Kostengründen einer anderen Architektur den Vorzug geben würden.

Daher wurde der Schluss gezogen, dass wenn man darüber nachdenkt Client-Server Systeme im Gebiet des “File Sharing” mit Peer-to-Peer Systemen zu ersetzen oder auch umgekehrt, im Vordergrund ein ökonomischer Vergleich der Systeme aus verschiedenen Perspektiven stehen muss, anhand von dem dann eventuelle Kostenvorteile eines Systems mit eventuellen Nachteilen z.B. in Bezug auf dessen Qualität abgewogen werden. Deswegen muss es im Moment den Benutzern und Anbietern solcher Systeme überlassen werden, ob sie eventuelle Qualitätsnachteile aufgrund von billigeren Kosten in Kauf nehmen wollen.

Abstract

The trend to a higher usage of Peer-to-Peer applications within several areas of the information and communication industry has increased. Especially in the area of file sharing several providers of such solutions have gained publicity, e.g., Napster, Gnutella or Kazaa. Generally it is stated that Peer-to-Peer systems have costs advantages compared to traditional Client-Server systems. Therefore, this work has the following goals.

In a first step the key differences of Client-Server systems and Peer-to-Peer systems are pointed out. On the basis of a generic approach it is figured out, if it is possible to replace all Client-Server systems with Peer-to-Peer systems.

In a second step a model is developed which allows a statement whether Client-Server systems are more suited for an application or Peer-to-Peer systems. This model especially focuses on economical aspects while considering different perspectives.

Peer-to-Peer systems are several in number and are used in many different areas today, therefore, the development of the economic model will only consider file sharing applications.

Typical representatives of such file sharing systems are analyzed, e.g., Gnutella as a pure Peer-to-Peer system, Napster as a hybrid Peer-to-Peer system and FTP as a Client-Server system.

The conclusions which this work found out, state that Peer-to-Peer is superior from an overall costs perspective compared to Client-Server in most of the cases. It could be assessed that especially hybrid Peer-to-Peer systems like Napster are very cost effective. But this fact has to be put into perspective regarding disadvantages in data availability, data consistency and bandwidth performance. In addition, an area of conflict between the participating stakeholders in such systems could be identified. The area of conflict includes the fact that basically each stakeholder group, e.g., clients/peers, ISPs or server operators, prefer another architecture regarding costs issues.

Therefore, the conclusion has been drawn, that when considering to replace Client-Server with Peer-to-Peer in the area of file sharing or vice versa, an economic evaluation of the different systems must come first including the different stakeholder perspectives. Based on such an evaluation cost advantages of a system can be compared with possible disadvantages regarding quality, e.g., data consistency. That’s why today users and providers of such systems have to decide whether they can accept shortcomings in quality in order to benefit from lower costs or not.

Contents

1 Introduction

Peer-to-Peer is a topic which is often discussed in the public. Peer-to-Peer applications especially in the area of file sharing, like Gnutella or Kazaa have a significant users base [17]. Also in areas like distributed computing or communication and collaboration Peer-to-Peer has contributed applications, e.g., SETI@home in the area of distributed computing.

Key characteristics of Peer-to-Peer are the usage of resources (storage, cpu cycles or bandwidth) at the edges of the Internet instead using a central server [5]. The edge means desktop computers which are normally located at home or at offices and could be connected to the Internet with dialup-, cable modem-, ADSL connections etc. Generally by using such untapped resources at the edge of the Internet, without using an expensive central server, costs can be saved at first sight. Client-Server systems have a well known history and users and developers have gained a lot of experiences with these systems. Peer-to-Peer systems instead are confronted with several deficiencies, e.g., that peers which participate in file sharing applications do not have sufficient available bandwidth to serve other peers as well as a Client-Server system would or that developed Peer-to-Peer applications creates high network traffic due to their implementation [1].

Therefore this work intends to analyze the Peer-to-Peer and Client-Server paradigm and will focus especially on economical aspects in order to compare both architectures. The following section will describe the procedure within this work.

1.1 Outline of work

In this work the Peer-to-Peer paradigm is described and compared to the Client-Server paradigm. Thereby key differences as well as advantages and disadvantages are pointed out in section 2.2 and 2.3. In section 2.4 a qualitative assessment is made whether it is possible to replace all centralized (Client-Server) systems through decentralized (Peer-to-Peer) systems. Limits and problems of such an approach are figured out. Subsequent to this assessment in chapter 3 a general classification of current Peer-to-Peer systems is presented whereas the main focus is put on the class of file sharing applications. In section 3.1 a general approach of how the costs of file sharing applications could be calculated is presented. Subsequently a simulation model is developed, which is able to compare typical Peer-to-Peer and Client-Server systems in the class of file sharing. Therefore in section 3.2 three typical representatives of applications that are used for file sharing are analyzed. The analyzed applications are Gnutella as pure Peer-to-Peer representative, Napster as hybrid Peer-to-Peer representative and FTP as Client-Server representative. In chapter 4 the economic model is presented, including a system definition (section 4.1), the simulation models for the three compared applications (from section 4.2 until 4.4), followed by a cost distribution analysis (section 4.5) and the economic models (section 4.6). In section 4.7 limitations of the cost models are discussed. In section 4.8 three different scenarios are defined in order to draw conclusions. In chapter 5 the results of the three scenarios are presented and commented as well. Based on the economic model and the results, in section 5.5, 5.6 and chapter 6 conclusions and recommendations are presented, where to apply a (pure) Peer-to-Peer solution or a Client-Server solution. This analysis considers different perspectives like ISPs, server operators and clients or peers. In section 6.2 an outlook is given, which provides proposals for further research topics and further improvements regarding the presented cost model. Finally it is discussed whether Peer-to-Peer is likely to evolve further in the class of file sharing or not.

2 Peer-to-Peer vs. Client-Server Paradigm

Today we encounter two main types of network paradigms based on the:

- Client-Server Architecture and the
- Peer-to-Peer Architecture

Definitions of both architectures are given in section 2.2 and 2.3. In section 2.1.1 a classification of Peer-to-Peer and Client-Server-Systems is done. Both Client-Server and Peer-to-Peer architectures are widely used and each has unique advantages and disadvantages which will be discussed in section 2.2.4 and 2.3.3. In section 2.4 a generic attempt is made to replace all Client-Server systems with Peer-to-Peer systems.

2.1 Internet Architectures

2.1.1 Classification of Computer Systems

All computer systems can be classified into centralized and distributed, see Figure 1. Distributed systems can be further classified into the Client-Server model and the Peer-to-Peer model. The Client-Server model can be flat where all clients only communicate with a single server (possibly replicated for improved reliability), or it can be hierarchical for improved scalability. In a hierarchal model, the servers of one level are acting as clients to higher level servers.

The Peer-to-Peer architecture is split into pure and hybrid architectures. The pure architecture works without a central server, whereas the hybrid architecture first contacts a server to obtain meta-information, such as the identity of the peer, on which some information is stored, or to verify security credentials. From then on, the Peer-to-Peer communication is performed. Examples of a hybrid model include Napster[1] and iMesh[2]. There are also intermediate solutions with SuperPeers, such as Kazaa[3]. SuperPeers contain some of the information that others may not have. Other peers typically lookup information at SuperPeers, if they cannot find it otherwise [7].

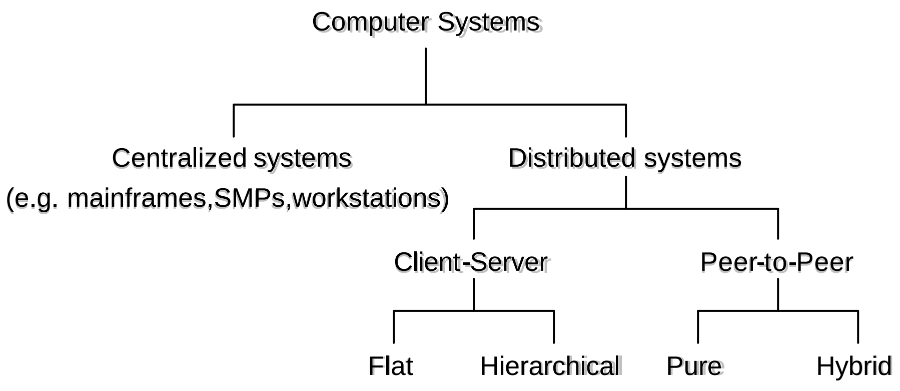


Figure 1: Classification of Computer Systems

1.http://opennap.sourceforge.net/
2.http://www.imesh.com/
3.http://www.kazaa.com/

2.2 Peer-to-Peer Architecture

Purpose of the following section is to get a first feel, what Peer-to-Peer in particular means. The following extracts shall give an overview about Peer-to-Peer definitions given in the literature and the basic Peer-to-Peer architectures used today.

2.2.1 Definitions

Peer-to-Peer architecture:

1 “P2P is a class of applications that takes advantage of resources -- storage, cycles, content, human presence -- available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.” [5]

2 “A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P,…) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers,…). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept).” [6]

3 “Peer-to-Peer computing is the sharing of computer resources and services by direct exchange between systems. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files. Peer-to-Peer computing takes advantage of existing desktop computing power and networking connectivity, allowing economical clients to leverage their collective power to benefit the entire enterprise. In a Peer-to-Peer architecture, computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network. This reduces the load on servers and allows them to perform specialized services (such as mail-list generation, billing, etc.) more effectively. At the same time, Peer-to-Peer computing can reduce the need for IT organizations to grow parts of its infrastructure in order to support certain services, such as backup storage.” [9]

Peer:

4 “A peer is a network node that can act as a client or a server, with or without centralized control, and with or without continuous connectivity. The term “peer” can apply to a wide range of device types, including small handheld and powerful server-class machines that are closely managed.” [11]

Node:

5 A node is a computing device residing on a network. Nodes may be general-purpose computers, or they may be specialized to provide particular services or capabilities (e.g. a storage node or control node). Note the term computing device is used in the most generic sense in that a node can range from a multi processor server to embedded systems.” [11]

As it is stated above, there are two different approaches for Peer-to-Peer architecture: pure and hybrid. Both architectures will be described in more detail below.

2.2.2 Pure Peer-to-Peer Architecture

“A distributed network architecture has to be classified as a “Pure” Peer-to-Peer network, if it is firstly a Peer-to-Peer network according to definition 1,see section 2.2.1, and secondly, if any single, arbitrary chosen terminal entity can be removed from the network without having the network suffering any loss of network service.” [5]

In this architecture each peer is an equal participant. There is no peer with special or administrative roles. According to the definition, no central server is needed to control and coordinate the connections between the peers, see Figure2. Therefore nodes have to self-organize themselves, based on whatever local information is available and interacting with locally reachable nodes (neighbors). Normally the data within such systems is distributed across multiple peers. Today pure Peer-to-Peer architecture does mostly exist in the area of file sharing (e.g. Gnutella).

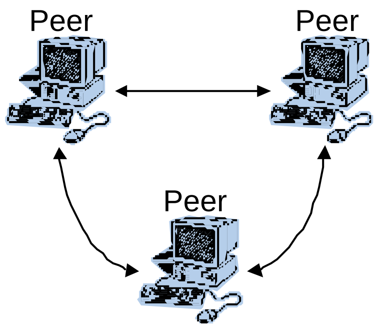


Figure 2: Pure Peer-to-Peer Architecture

2.2.3 Hybrid Peer-to-Peer Architecture

Unlike the pure Peer-to-Peer model, hybrid Peer-to-Peer models, such as Napster, incorporate some traces of the Client-Server relationship. Hybrid in the case of Peer-to-Peer means, that there is a central server in the system, but it takes only an intermediary role in the system. Central servers within the network fulfill two primary functions. First, they act as central directories where either connected users or indexed content can be mapped to the current IP address. Second, the servers direct traffic among the peers. Normally the initial communication of a peer is done with a server (1), e.g., to obtain the location/identity of a peer, followed by (2) direct communication with that peer, see Figure3.

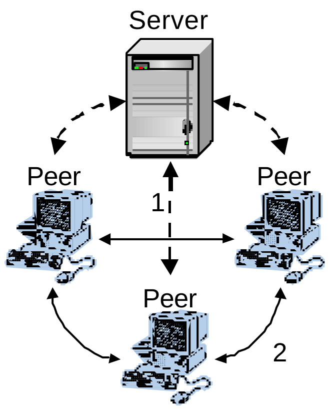


Figure 3: Hybrid Peer-to-Peer Architecture

2.2.4 Advantages and Disadvantages

Below advantages and disadvantages are presented, which are often stated in technical reviews and books concerning Peer-to-Peer architecture.

Advantages

- In a pure Peer-to-Peer architecture there is no single point of failure, that means, if one peer breaks down, the rest of the peers are still able to communicate.
- Peer-to-Peer provides the opportunity to take advantage of unused resources such as processing power for computations and storage capacity. In Client-Server architectures, the centralized system bears the majority of the cost of the system. In Peer-to-Peer, all peers help spread the cost, e.g. Napster used the file storage space of participating peers to store all the files.
- Peer-to-Peer allows to prevent bottleneck such as traffic overload using a central server architecture, because Peer-to-Peer can distribute data and balance request across the net without using a central server.
- There is better scalability due to a lack of centralized control and because most peers interact with each other.

Disadvantages

- Today many applications need a high security standard, which is not satisfied by current Peer-to-Peer solutions.
- The connections between the peers are normally not designed for high throughput rates, even if the coverage of ADSL and Cable modem connections is increasing.
- A centralized system or a Client-Server system will work as long as the service provider keeps it up and running. If peers start to abandon a Peer-to-Peer system, services will not be available to anyone.
- Most search engines work best when they can search a central database rather than launch a meta search of peers [2]. This problem is circumvented by the hybrid Peer-to-Peer architecture.

2.3 Client-Server Architecture

Just as in section 2.2 the purpose of the following section is to get a first feel, what Client-Server means and which definitions are mentioned in the literature and how the basic Client-Server architecture looks like.

2.3.1 Definitions

1 “A Client-Server network is a distributed network which consists of one higher performance system, the server, and several mostly lower performance systems, the clients. The server is the central registering unit as well as the only provider of content and service. A client only requests content or the execution of services, without sharing any of its own resources.” [6]

2 “A Client-Server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power.” [8]

2.3.2 Architecture

The most commonly used paradigm in constructing distributed systems is the Client-Server model. In this scheme clients request services or content from a server. The client and server require a known set of conventions before the can communicate. This set of conventions contains a protocol, which must be implemented at both ends of a connection. Examples of protocols are the TELNET protocol used in the Internet for remote terminal emulation, the Internet file transfer protocol, FTP and the most widely used hypertext transfer protocol, http.

Server

As a provider of services the server must compute requests and has to return the results with an appropriate protocol. A server as a provider of services can be running on the same device as the client is running on, or on a different device, which is reachable over the network. The decision to outsource a service from an application in form of a server can have different reasons.

- **Performance:** In certain circumstances the clients are inefficient devices, which have interfaces to high performance demanding applications. In this case the computation is done on a high-performance server. Today this approach is less used, but has still its area of application, e.g., virtual reality computations for film scenes.
- **Central data management:** This aspect of the Client-Server model does have the most impact today. Data is stored on a server, which can be used or manipulated from different clients. Typical examples of services provided by a server are:
  - ♦ File server: One server provides multiple clients with a file system. Tasks of this server include access control and transaction control (only one client may access a file with write permissions at a time).
  - ♦ Web server: The Web server provides multiple clients (Web browser on different devices) with information. The information can be static on a Web server or dynamic, generated by different service applications.

Client

A client is typically a device or a process which uses the service of one or more servers. Since clients are often the interface between server-information and people, clients are designed for information input and visualization of information. Although clients had only few resources and functionality in the past, today most clients are PCs with more performance regarding resources and functionality. Early clients had only the task to display the application, that was running on the server and to forward inputs of the user to the server. All computations are done on the server. In this case one speaks of a thin client. A thin client has limited local resources in terms of hardware and software. It functionally requires processing time, applications and services to be provided from a centralized server. Network computers are examples of the development of thin clients.

A thick client is functionally rich in terms of hardware and software. Thick clients are capable of storing and executing their own applications as well as network centric ones. Thick client typically refers to a personal computer [10].

As it is stated in section 2.1.1, Client-Server architectures can be classified into flat and hierarchical. If the Client-Server model is flat, all clients communicate only with a single server, see Figure4. If the Client-Server model is hierarchical the servers of one level are acting as clients to higher level servers. A pretty good example is a request of a certain web page. The user enters a URL into the web browser (client). The client establishes a connection to his nearest name server to ask for the address. If that server does not know the name, it delegates the query to the authority for that namespace. That query, in turn, may be delegated to a higher authority, all the way up to the root name servers for the Internet as a whole. Name servers operate both as clients and as servers, see Figure 4.

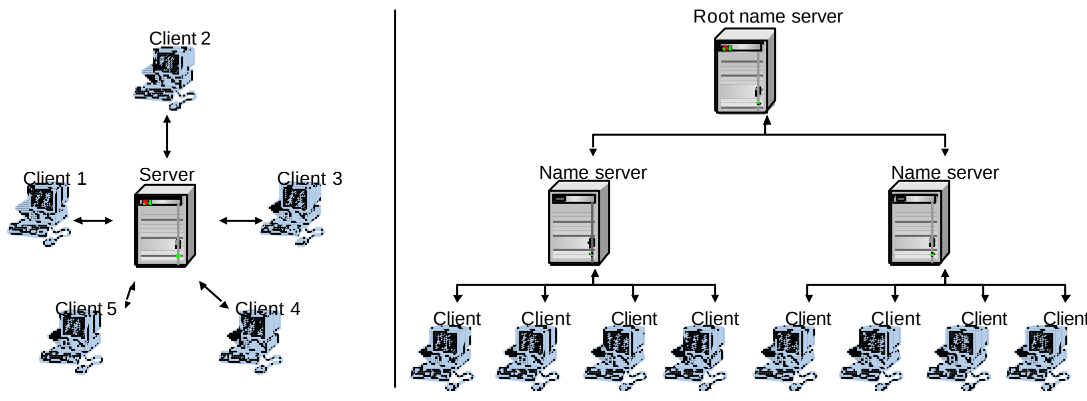


Figure 4: Flat architecture (left); Hierarchical architecture (right)

2.3.3 Advantages and Disadvantages

The following advantages and disadvantages of the Client-Server architecture are extracted from several technical reports and books concerning Client-Server architecture. The following points should not be regarded as a complete listing, but rather as key advantages and disadvantages which are comparable to list presented in section 2.2.4.

Advantages

- Data management is much easier because the files are in one location. This allows fast backups and efficient error management. There are multiple levels of permissions, which can prevent users from doing damage to files.

- The server hardware is designed to serve requests from clients quickly. All the data are processed on the server, and only the results are returned to the client. This reduces the amount of network traffic between the server and the client machine, improving network performance.
- Thin client architectures allow a quick replacement of defect clients, because all data and applications are on the server.

Disadvantages

- Client-Server-Systems are very expensive and need a lot of maintenance.
- The server constitutes a single point of failure. If failures on the server occur, it is possible that the system suffers heavy delay or completely breaks down, which can potentially block hundreds of clients from working with their data or their applications. Within companies high costs could accumulate due to server downtime.

2.4 A Generic Approach - Replace all C/S Systems with pure P2P Systems

In this chapter a generic approach should be made, if it is possible to replace all Client-Server systems with pure Peer-to-Peer systems. Therefore it is attempted to point out, which changes regarding resources have to be done and which advantages and disadvantages this would implicate. The proceeding is as follows: First it is considered how to map the hardware, which is used for Client-Server architecture, onto an environment based on Peer-to-Peer architecture. The focus of this “mapping” is set on processing power, storage and bandwidth. Secondly the impact of such a “mapping” on the different classes of application is discussed. Especially the problems and limits which could occur by such a “mapping” are analyzed. Out of this analysis, conclusions for the different classes of applications are drawn.

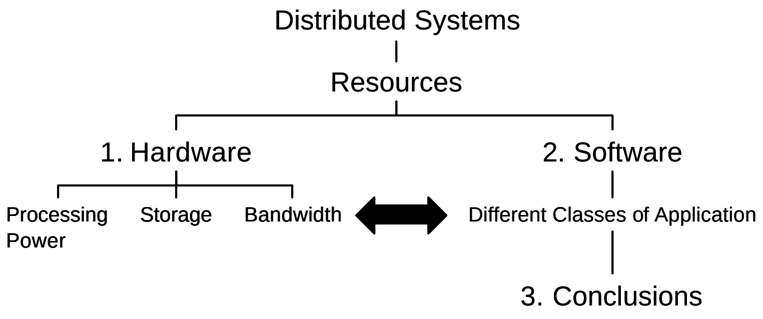


Figure 5: Generic Approach: Procedure

2.4.1 Hardware changes

Peer-to-Peer computing is about sharing of computing resources and using these resources more efficiently, like the peers’ processing power and storage (cf. section 2.2.1). In order for the peers to be able to take advantage of these resources a network connection must be available to connect the peers. Since the sharing of these resources depends strongly on the class of application the hardware changes are discussed very pragmatically.

Processing Power

In Client-Server systems the servers provide processing power and services to the clients. If it is a thin-client and after the calculations are done, the result is sent back to the client. In pure Peer-to-Peer systems there is no central server. The sharing of processing power must be done on the peers. Since the processing power of peers is normally lower than on strong servers the goal is to parallelize large tasks into smaller pieces that can execute in parallel over a number of independent peer nodes. There are problems, which are not parallizable by nature. This leads to Amdahl's Law [3], which is a law governing the speedup[4] of using parallel processors on a problem, versus using only one processor. With this law it is possible to calculate applicability of a centralized computing approach vs. a decentralized approach provided that peer nodes exist, which want to share their processing power.

If a Client-Server system requires more processing power for its calculation the server resources can be extended, so that the desired performance is available. In a pure Peer-to-Peer system, where peers are more or less autonomous and distributed over the net, the possibility for administration is low because responsibility lies with the owner of the peer. Therefore if a Peer-to-Peer system needs more processing power there are two ways how this extension can be done. First, the amount of peers, which share their processing power, must increase or, secondly, the peers themselves have to upgrade their processing power.

Storage (memory)

In a Client-Server system most of the data is normally stored on a central server. The data is accessible from there by the clients if they have the appropriate access permissions. The server provides a specific storage capacity, which can be used by the clients. In a Peer-to-Peer system the peers have to provide the storage function. Therefore, if it is intended to store the same amount of information, the aggregated storage capacity of the peers, that means only the storage, which is dedicated for Peer-to-Peer storage, should be at least equal to that on a central server.

Administration is as well an important issue for storage. Client-Server systems have to be maintained and repaired or resources - in this case storage - have to be extended. If a Peer-to-Peer systems is implemented, it is difficult to maintain due to autonomy of the peers and their distribution across the net. Analogously to an extension of processing power, there are two approaches to increase the amount of storage capacity, either by adding more peers or letting the peers upgrade their storage by themselves.

Bandwidth

By decentralizing data and therefore redirecting users so that they can download data directly from other users’ computers instead of a central server the load on the central server can be taken away. If the Client-Server system is replaced by Peer-to-Peer it has to be taken into account, that the connections between the peers should have an equivalent provided bandwidth as they had in the Client-Server system to satisfy the users expectations. In case of a file server: If ten clients want to download the same file from the server, they have to share the bandwidth, with which the server is connected to the net. In a Peer-to-Peer system the peers are at the edge of the Internet and often only have dialup or asymmetrical connections like cable modems. So if a file is stored on a peer and 10 other

4.The speedup of a parallel program is defined to be the ratio of the rate at which work is done when a job is run on N processors to the rate at which it is done by just one.

peers want to download this file at the same time, they have to share the upload-bandwidth from this peer to its ISP. For this reason, when implementing a Peer-to-Peer system, an upgrade of the peers' bandwidth capacity has to be done in order to prevent an overload of some peers.

Another issue concerning bandwidth is to keep distributed systems together. In a Client-Server system, a client normally knows how to contact the server (e.g., via its IP-address) and therefore the client does not have to stay connected with the server all the time. For this reason the consumption of bandwidth is minimized. In a Peer-to-Peer system where peers may come and go and IP-addresses are changing constantly, it is important that the peers know where to find the other peers. So they have to send information to the other peers in order to let them know how they can be found on the net. Therefore, it is important that the algorithms, which are required to maintain such a system and which often carry a lot of overhead, e.g., ping pong messages in the Gnutella network, are optimized. The overhead should not overload the network, if the network grows. Otherwise the system will not scale well.

2.4.2 Software changes

The goal of computer systems is to support applications that satisfy the needs of users. Therefore approaches of computer systems are driven by several goals like cost reduction, improved performance and reliability, etc. Each application has different needs regarding hardware, e.g., for a compute intensive application it could be more important to have enough processing power than a large hard disk. For an online flight reservation system it is important that enough storage capacity is available and all files are accessible, so that the application can search the database for available seats. To evaluate the impact of the hardware changes in case of a pure Peer-to-Peer system a set of three Client-Server application classes is defined and assessed. The three classes of applications are compute intensive applications, storage demanding applications and bandwidth demanding applications. It is not the intention that applications will fit exactly in one of these categories, but rather that each application is a mix of these three classes. It will be analyzed what characteristics each class of application in a Client-Server systems has and how an implementation in a Peer-to-Peer system could look like and which limits could occur.

Compute Intensive Applications

Client-Server computing gives clients the opportunity to work on desktop workstations, while compute intensive applications remain on high performance servers. If the Client-Server system is replaced by Peer-to-Peer the compute intensive applications have to rely on desktop PCs to provide the processing power. As it is stated above, within such an architecture it is intended to split up large tasks in smaller pieces (parallelization), which then are sent to the peer. But if a problem contains many non parallizable parts in a pure Peer-to-Peer system, there still have to be high performance desktop computers, which can solve these parts in an appropriate amount of time.

Since the peers are distributed over the net, a mechanism has to be implemented which manages the task distribution between the different peers and looks for unused processing power. Due to the fact that Peer-to-Peer computing contains unpredictable connectivity regarding its peers, the performance of compute intensive calculations implemented by Peer-to-Peer is difficult to ensure. If a peer disconnects from the network without having sent the results to the requesting peer time losses can occur, if the calculations are not re-sent to another peer for calculation.

Another point which has to be solved within Peer-to-Peer systems is the insurance of the result quality. In a Client-Server architecture the client normally trusts the server, where the computations are done. In a Peer-to-Peer architecture, where peers come together, which want to share their processing power, the danger of data manipulation of “bad” peers have to be solved. This can be done by sending the same calculations to different peers in order to compare their results afterwards. This procedure implies a higher resource consumption and therefore a lower performance.

In a Peer-to-Peer system there are peers, which are unknown and which should not take advantage of the calculation’s content or the result. Therefore security options have to be implemented.

Storage Demanding Applications

Within a network a server may provide storage capacity to its clients and their applications. Traditional LAN computing allows users to share resources, such as data files, by moving them from desktop PCs onto a network file server. Client-Server systems normally have constantly-updated directories. So if a client requests a file the server knows where to look for it and can return the result to the client. If the Client-Server system is replaced by a Peer-to-Peer system there is neither a centralized directory nor any precise control over the network topology or file placement. Gnutella is an example of such a design. The network is formed by nodes joining the network following some loose rules (for example, those described in [1]). To find a file, a node queries its neighbors. The most typical query method is flooding, where the query is propagated to all neighbors within a certain radius. These designs are extremely resilient to nodes entering and leaving the system. However, the current search mechanisms are not at all scalable, generating large loads (bandwidth) on the network participants [14]. Therefore, it is important to analyze if the network can cope with the additional load a Peer-to-Peer system would imply and if latency of replies or unsuccessful queries due to missing data availability or ineffective search algorithms are justifiable for the application. Another approach to deal with search efficiency is to bring in more structure as it is described in [14]: “Structure” means that the Peer-to-Peer network topology (that is, the set of connections between Peer-to-Peer members) is tightly controlled and that files are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy. In highly structured systems both the Peer-to-Peer network topology and the placement of files are precisely determined. This tightly controlled structure enables the system to satisfy queries very efficiently.

The consistency of data is an important issue. In a Client-Server system this is addressed with a central data management, where a file is accessible (read/write) for only one client at the same time. In Peer-to-Peer systems copies of the same data are available on different peers. This leads to a higher availability of the data on the one hand, but on the other hand problems with the consistency of the data come up. If a file on one peer is changed all the other copies of this file in the system have to be synchronized as well to ensure consistency.

In a Client-Server system administrators are responsible for the data stored on their servers. They search for harmful files, e.g., viruses on their servers to protect the clients. So if a client requests a file from a specific server and if the client trusts this server, the client can rely on the file quality. In pure Peer-to-Peer networks the authenticity of search results is a security problem. If a peer enters a search term this request is forwarded to multiple other peers, not to a central server, which would search its index. Each peer searches in its public files for the corresponding term and replies the results to the searching peer. A “bad” peer could, e.g., rename viruses to common search terms so that they are downloaded by other peers and maybe executed by unexperienced users. To solve these issues in Peer-to-Peer networks trust, scoring, and reputation models have to be developed.

If the server is available in a Client-Server-System and the clients are connected, data is accessible for the clients. “Peer-to-Peer systems on the other hand seem to contain an inherent fuzziness. Gnutella, for instance, doesn't promise you'll find a file that's hosted on its system; it has a horizon beyond which you can't see what's there. [...] Most computers come and go on the Internet, so that the host for a particular resource may be there one minute and gone the next.” [12] In theory, when increasing the number of nodes in a pure Peer-to-Peer system, aggregated storage space and file availability should grow linearly [13]. If data availability has to be guaranteed in a pure Peer-to-Peer system the peers have to be either forced to stay connected or the data has to be replicated on other nodes, which would also lead to a need for more storage capacity. The replication implies that consistency problems have to be addressed again.

Administration is also an important issue for storage demanding applications. In Client-Server systems applications have to be installed or updated and access permissions for the clients have to be defined. If a Peer-to-Peer systems is implemented it is difficult to administrate issues like access permissions due to autonomy of the peers. That means each peer has normally the same rights as all the others. Therefore it is important to develop at least a function for file access control, so that the peers can define access permission for their files.

Bandwidth Demanding Applications

In a Client-Server system the clients are connected directly with the server over a network link, which provides a certain bandwidth. Depending on the implementation of the Client-Server system (cp. thin-or thick-clients in section 2.3.2) applications can cause more or less network traffic and therewith a higher bandwidth consumption, e.g., if an application runs on a client but needs large files, which are stored on the server, the bandwidth consumption will be higher than if the application ran on the server as well and only the visualization of the results were done on the client. So if a Peer-to-Peer system is implemented, it depends on the application how much bandwidth between the peers is consumed. If one considers a possible video conference application for two peers the bandwidth demand for the network link between those two peers is higher than it would be for a text chat. So if the Client-Server system is replaced by Peer-to-Peer, it has to be taken into account, that the connections between the peers should have an equivalent guaranteed bandwidth as they had in the Client-Server system to satisfy user expectations.

2.4.3 Conclusions

Bearing in mind the considerations above and the advantages and disadvantages mentioned in section 2.2.4 and 2.3.3, it could be possible to replace Client-Server systems through Peer-to-Peer versions, but associated with severe disadvantages and concessions in some classes of application.

Both systems offer a number of advantages. Peer-to-Peer provides the opportunity to make use of untapped resources. These resources include processing power for computations and storage potential. Peer-to-Peer allows the elimination of bottlenecks at the server and can be used to distribute data and control load across the Internet. The Peer-to-Peer mechanism also may be used to eliminate the risk of a single point of failure. Nevertheless as it was stated at the beginning of section 2.4.2 the applicability of the pure Peer-to-Peer paradigm is strongly dependent on the area of application.

A main limitation seems to be the availability and consistency of data in the class of storage demanding applications. In the class of compute intensive applications not parallizable problems, the availability of peers and the result quality seem to be the main limitations in the class. Whereas in the class of bandwidth demanding applications the main burden seems to be the bandwidth between the peers causing scalability problems. Another problem is the extensibility of hardware resources in all three classes of application.

Therefore applications, which have to provide high data availability, consistency etc., should better be built on the Client-Server paradigm. As Andy Oram says analogously in [2], Peer-to-Peer will not replace the Client-Server model entirely. Client-Server remains extremely useful for many purposes, particularly where one site is recognized as the authoritative source of information and wants to maintain some control over that information.

Above severe disadvantages and concessions in some classes of application were mentioned regarding a possible replacement of Client-Server with Peer-to-Peer. This could be, e.g., time loss in the area of compute intensive applications, if a task is not parallizable and no high performance computer is available within the network. Another example could be a flight reservation system, which has mainly storage demanding characteristics and where availability and consistency of the data is very important unless a passenger wants to share his seat with other passengers due to data which is not topical and not consistent while a reservation was made.

These may be some reasons why today there are only few pure Peer-to-Peer applications like Gnutella or Freenet. Hence users and developers of applications have to find an optimal trade-off for each class of application between Client-Server architecture and pure Peer-to-Peer architecture. The trade-offs must consider issues like the ones pointed out above and economic considerations.

3 System Classification and Focus

In [7] a taxonomy of Peer-to-Peer system is provided. It classifies Peer-to-Peer systems into four classes, namely into distributed computing (e.g., SETI@home), file sharing (e.g., Gnutella, Napster, Freenet), communication and collaboration (e.g., Jabber, MSN Messenger) and platforms (e.g., JXTA, .NET), see also Figure 6.

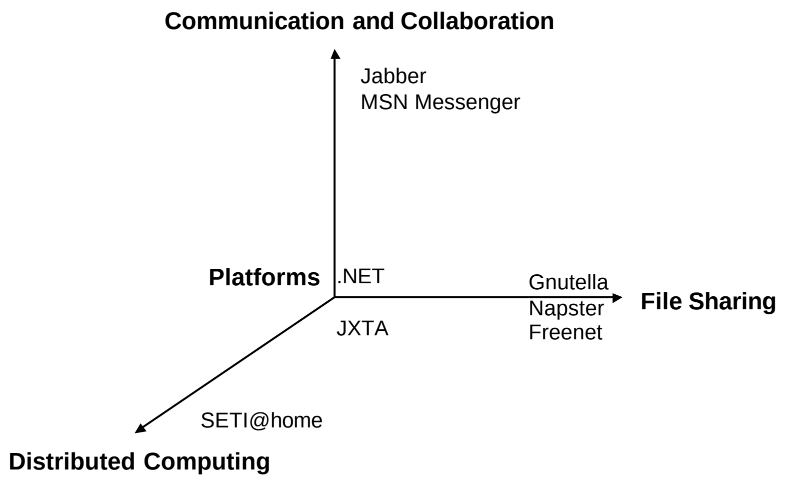


Figure 6: Peer-to-Peer taxonomy: Sample systems/applications

Since file sharing is the most known and may be the most widely used Peer-to-Peer class and also the most criticized class due to, e.g., copyright infringements, the focus of the following work is led on this class in order to compare, if it is beneficial to replace a file sharing system that is based on a Client-Server architecture with a Peer-to-Peer architecture. The further goal of this work will be to build a model which allows to compare file sharing applications on the basis of economical and performance related aspects.

Therefore in section 3.2, three file sharing applications will be analyzed regarding their history and functionality. Each application is a typical representative of the Client-Server, the hybrid and pure Peer-to-Peer architecture. In section 3.3 some qualitative criteria are established to benchmark file sharing applications in order to find out if it is worth to replace a Client-Server system with Peer-to-Peer.

3.1 General Approach to evaluate Costs

Within this section a general approach should be described of how one could evaluate the costs of file sharing applications like Gnutella, Napster or FTP.

First step

In section 2.4 it is stated that applications could be classified roughly into storage demanding, bandwidth demanding and compute intensive applications, whereby an application will typically be a mix of these classes. Therefore the first step must comprise an analysis of the application (see section 3.2), which infrastructure it needs and which communication protocols the application is using and is built on.

Second step

Before the model is built the system's scope (see section 4.1) should be defined regarding system constants, variables and assumptions. Within this system definition the values for constants (e.g. harddisk price), assumptions and variables have to be specified.

Third step

When applications' protocols are understood, models (e.g., see section 4.2) must be built, which can simulate the behavior of the application under different values. The simulation model should include "checkpoints", which allow to prove the plausibility of the model and may show limits to the systems' components (e.g. the application traffic exceeds the client's available bandwidth). After the application model has been built, a cost distribution analysis (e.g., see section 4.5) should be done, which clarifies who pays for what when using a certain file sharing application.

Fourth step

Since in the second step the costs for "system equipment" has been defined, these costs have to be combined with the simulation model and the cost distribution analysis (third step). See, e.g., section 4.6.1.

Fifth step

This step includes the setup for the simulation, that means to define which variable values (e.g. number of clients) will be used as input for the simulation in order to draw conclusions regarding the costs. See section 4.8.

Sixth step

After the simulations, a stakeholder analysis should be done, in order to evaluate the application regarding their cost impact to the different stakeholders. See chapter 5.

3.2 Compared Applications in the Area of File Sharing

Within this subsection an overview of the compared file sharing applications, namely Gnutella, FTP and Napster, will be provided. The overview contains a brief history about the application, and its design including protocol specifications. The protocol specification are discussed quite deeply so as to allow a detailed cost analysis later on.

3.2.1 Gnutella

History

Gnutella is a file sharing protocol, which allows applications that use this protocol as a client software to search and download files of other users using Gnutella. The protocol was developed by Justin Frankel and Tom Pepper employees of AOL as open-source in 2000. They published the Gnutella client software on a web site, describing it as “a software tool to file-sharing, that can be more powerful than Napster”. This software was immediately taken offline after one day, due to merger talks between AOL and Warner Music and EMI. But many copies of this software have been downloaded, so that Gnutella clients were communicating soon after, building a network for file searching and file sharing. Today many companies implemented clone software and try to overcome the shortcomings of the original protocol. Such clone software is e.g. “Limewire”, “BearShare”, or “Nucleus”. The actual number of Gnutella hosts accepting incoming connections on the 20th of March 2003 is 14000 hosts and the number of unique hosts is ~80000 as stated in [17].

General Design

The goal of Gnutella is to provide a file sharing solution, which follows the pure Peer-to-Peer architecture as described in section 2.2.2, whereas Gnutella allows all types of files to be shared. Users that run applications, which have implemented the Gnutella protocol can share specified files, can accept queries of other clients and match them with their local files in order to deliver results. At the same time, the client can search for new files. Therefore every client is a server and also a client, see “servent-concept” [6]. Below the word servent will be used to describe a Gnutella client. Since the peers do not have to provide any personal information, Gnutella provides anonymity for it’s users.

The Gnutella protocol specifies how the servents communicate over the network. In [1], a set of descriptors is defined, which are used to communicate and to transmit data between servents. In Figure7 these descriptors are presented.

Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.
QueryHit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.

Figure 7: Gnutella descriptors

A servent can connect to the network by establishing a connection with another servent currently on the network. Therefore, the servent must know the IP address of another servent in the Gnutella network to log in. Most of the applications today have implemented a set of well known IP addresses of other servents, which can be contacted first. When the servent has a valid IP address a TCP/IP connection to the servent is tried to create. After a successful “handshake” between the servents, the communication between the servents runs by sending and receiving Gnutella protocol descriptors.[5] Each descriptor is preceded by a byte structure shown in Figure 8 and is transported on TCP. All fields in the following descriptors are in little-endian byte[6] order unless otherwise stated.

Gnode ID	Payload Descriptor	TTL	Hops	Payload length
16 Bytes	1 Byte	1 Byte	1 Byte	4 Bytes

Figure 8: Gnutella descriptor header

- The Gnode ID is a 16-byte string, which is uniquely generated for each new descriptor.
- The Payload Descriptor defines which descriptor, see Figure7 is sent.
- Time to live (TTL) defines the times the descriptor will be forwarded by Gnutella servents before they delete this descriptor. Each servent receiving a descriptor decrements its TTL by 1. If the value of TTL reaches 0 the descriptor will not be forwarded.
- Hops define the number of times a descriptor has been forwarded. As described in [1], the TTL and Hops fields of the header have to satisfy the following equation, where TTL(i) and Hops(i) are the actual values of the descriptor at hop i:

$$\mathrm{TTL}(0) = \mathrm{TTL}(i) + \mathrm{Hops}(i)$$

Equation 1

- Payload length defines the byte length of the following descriptor.

The TTL is the only mechanism for expiring descriptors on the network. TTL and Hops is the way Gnutella defines the scope of servents’ queries and network knowledge; the higher the value of TTL, the bigger the scope.

The value of the Payload length is the only way to identify the next descriptor in the data input stream. Therefore, if a servent gets out of synchronization with its input stream the servent should drop the connection. After the descriptor header follows the payload, consisting of one of the descriptors mentioned in Figure 7. Below these descriptors are described more detailed, except the Push descriptor, since it is only used to overcome problems with servents behind firewalls, which is not relevant for this work.

5.By obtaining more IP addresses of other servents, a servent can try to open up more than one TCP/IP connection. Depending on the application the servent can define the minimum and maximum number of direct connections to other servents.

6.Endianness is the attribute of a system that indicates whether integers are represented from left to right or right to left. Little endian means that the least significant byte of any multibyte data field is stored at the lowest memory address, which is also the address of the larger field. Big endian means that the most significant byte of any multibyte data field is stored at the lowest memory address, which is also the address of the larger field.

Ping descriptors have no payload. They only consist of a descriptor header with payload zero. In the Gnutella protocol specifications [1] no recommendations are given about the frequency at which a servent should send a ping descriptor, although it is stated that implementors should make any effort to minimize the amount of ping traffic on the network.

Pong descriptors are sent in response to a ping descriptor. It is valid to send more than one pong descriptor as reply for a single ping descriptor. This enables a servent to submit his cached addresses of other servents in response to a ping. The pong descriptor payload consists of the host IP address (in big-endian format) and port number, the number of files and the number of kilobytes shared. A pong descriptor has a 14 byte length.

Port	**IP Adress**	**Number of Files shared**	**Number of Kbytes shared**
2 Bytes	4 Bytes	4 Bytes	4 Bytes

Figure 9: Pong descriptor

Query descriptors are search messages containing a query string and the minimum speed. The minimum speed field forces servents to only reply, if they can communicate at the specified minimum speed. A query descriptor has a 2+n byte length depending on the search criteria.

Minimum Speed	**Search criteria**
2 Bytes	n Bytes

Figure 10: Query descriptor

Queryhit descriptors are only sent in response to an incoming query descriptor. The servent should only reply with a queryhit descriptor, if it contains data that matches the search criteria. A queryhit descriptor contains the IP address, the port number and speed (Kb/second) of the responding host, followed by a result set and a servent identifier.The result set contains names and sizes of files matching to the search criteria. The size of the result set is limited by the size of the payload length in the descriptor header. A queryhit descriptor has a 27+n byte length, whereby the result set has a 4+n byte length:

Number of Hits	**Port**	**IP Address**	**Speed**	**Result Set**	**Servent Identifier**
1 Byte	2 Bytes	4 Bytes	4 Bytes	n Bytes	16 Bytes

File Index	**File Size**	**File Name**
4 Bytes	4 Bytes	n Bytes

Figure 11: Queryhit descriptor

Descriptor Routing

Due to its distributed nature, a network servent that implements the Gnutella protocol must have the ability to route incoming descriptors (ping, pong, query, queryhit) through its open connections. Therefore a Gnutella servent should follow the rules explained below:

- Servents will forward incoming pings and queries to all its directly connected servents except the one which delivered the ping or query descriptor.

- Pong and queryhit descriptors may only be routed back on the same path by which the corresponding ping or query descriptor was routed. This is possible since servents run tables in which they can review the way a certain ping or a query came from.
- A servent will decrement the descriptor header’s value of TTL and increment the value of hops by one, before a descriptor is forwarded to directly connected servents. If the value of TTL is zero the descriptor may not be forwarded to any of the directly connected servents.
- If a servent receives a descriptor with the same payload descriptor and descriptor ID as it has received anytime before, it should not forward this descriptor to any of it’s directly connected servents, since they have probably already received the same descriptor.

File download

If a servent receives a queryhit descriptor, it can initiate a direct download of one of the files listed up in the descriptor’s result set. Files are not downloaded over the Gnutella network. Instead, the download runs over HTTP protocol. The servent sends a download request that contains <File Index> and <File Name>, which correspond to one of the files in the result set field of the queryhit descriptor. When the other servent receives this request, it will respond and the data transfer follows.

3.2.2 Napster

History

The idea of Napster came from its creator Shawn Fanning in 1998. Shawn created programs where one could locate and transfer files in real time. This particular notion was unlike other computer or traditional search engines. A system was created where users could log on and update their current files. By May 1999 Shawn's idea developed into full operation, which later became known as Napster Incorporated. Napsters popularity continued to grow and its size exceeded the amount of 60 million users. The demise fell earlier than anticipated by Napster fans, when the music industry took the company to court in 2000 and 2001. The Court appealed rules that Napster must prevent its subscribers from gaining access to content on its search index, that could potentially infringe copyrights. Today Napster itself is not longer present as an “open” file sharing solution, however there are many “open” clones available which can connect to OpenNap servers.

General Design

The goal of Napster was to provide a file sharing solution which follows the hybrid Peer-to-Peer architecture as described in section 2.2.3, whereas, compared to Gnutella, Napster restricts the file types to be shared to mp3 files. Napster works with a central server, which contains an index of all mp3 files shared by clients[7]. The clients run the Napster software on their computers. By launching Napster, the index of the central server will be updated with the client’s shared file names. If a peer wants to download a specific file, it sends a query to the central server, which replies with port number and IP address of a client, which is sharing this file. Since Napster is not open source the following protocol analysis relies on the napster protocol specification [18], which was done by reverse engineering. Compared to Gnutella, the Napster protocol defines a large number of message types. Basically every activity of a client concerning Napster is related to the central Napster server. This makes

7.Even Napster is widely regarded as an hybrid Peer-to-Peer system the peers are called clients in most of the essays concerning Napster.

anonymity of the clients impossible but provides additional services like instant messaging or creating hotlists. Since Gnutella, FTP and Napster are analyzed regarding their function of "file sharing", the following protocol analysis of Napster will limit the scope on the essential messages to provide a file sharing solution. These messages are login, client notification of shared files, client search request, search response, download request, download ack, download complete and upload complete, see Figure 12:

Message	Message receiver	Description
Login	Server	A registered Napster client sends a login message to the server in order to share and download files.
Client notification of shared files	Server	This message is sent by a Napster client at the beginning of a session to update the index on the central server with all the files it want to share.
Client search request	Server	This message is sent by a napster client searching for files.
Search response	Client	Response to a client search request message.
Download request	Server	Client request to download a file from another Napster client.
Download ack	Client	Server sends this message in response to a download request message.
Downloading file	Server	Client sends this message to the server to indicate that it is in the process of downloading a file.
Uploading file	Server	Client sends this message to the server to indicate that it is in the process of uploading a file.
Download complete	Server	Client sends this message to the server to indicate that the process of downloading a file is completed.
Upload complete	Server	Client sends this message to the server to indicate that the process of uploading a file is completed.

Figure 12: Napster messages

Napster uses TCP for client to server communication and typically runs on port 8888 or 7777. Each Napster message is preceded by a byte structure shown in Figure13, where <length> and <byte> are in little-endian format, whereas the <data> portion of the message is a plain ASCII string.

Length	**Type**	**Data**
2 Bytes	2 Byte	n Byte

Figure 13: Napster message header

- Length specifies the length in bytes of the <data> portion of the message.
- Type defines the message type, which follows in <data>.
- Data contains a plain ASCII string.

Within the Napster protocol each block of header and <data> is separated by a single space character (1 byte), which allows to identify single blocks of an incoming bit stream.

Login messages are sent by the client to the server in order to get connected to the Napster network, so that a client can share and download files. A login message contains fields for the client's nickname, its password, its port number, information about the client's software version and the link type, which specifies the client's bandwidth. The client's IP address has not to be added, since the server can extract it out of the TCP packet.

Nick	Password	Port	"Client-Info"	Link-Type
1 Byte/Char	1 Byte/Char	1 Byte/Char	2 Bytes+1 Byte/Char	1 Byte/Char

Figure 14: Login message

Client notification of shared files messages are sent by the client to the server in order to submit the client's actual shared files. For each file a client wants to share such a message is generated. The message contains field for the shared filename, a hash value of the shared file[8], the size in [bytes], the bit rate in [kbps], frequency in [Hz] and the length in [seconds] of the shared music file.

Filename	MD5	Size	Bitrate	Frequency	Time
1 Byte/Char	32 Byte	1 Byte/Char	1 Byte/Char	1 Byte/Char	1 Byte/Char

Figure 15: Client notification of shared files message

Client search request messages are sent by clients searching for files. A search message contains fields for the artist, for the maximum number of search results, the song name, the demanded linespeed for possible download candidates, the desired bit rate in [kbps] and the frequency in [Hz] of the music file the client is searching for. The <compare> condition can be "AT LEAST", "AT BEST" and "EQUAL TO".

[FILENAME CONTAINS "artist name"] 20 Bytes+1 Byte/Char	**[MAX-RESULT <max>]** 11 Bytes+1 Byte/Char
[FILENAME CONTAINS "song"] 20 Bytes+1 Byte/Char	**[LINESPEED CONTAINS <compare> <link-type>]** 20 Bytes+7 to 8 Bytes+1 Byte/Char
[BITRATE <compare> " "] 11 Bytes+7 to 8 Bytes+1 Byte/Char	**[FREQ <compare> "<freq>"]** 8 Bytes+7 to 8 Bytes+1 Byte/Char

Figure 16: Client search request message

Search response messages are sent by the server upon client search request messages. A search message contains fields for the filename matching to a search request, the hash value, the size, the bit rate, the frequency and the length of the returned file. Additionally the nickname, the IP address and the link-type of the client, which is sharing this file is submitted. For each match on the server's index list such a message is sent to the client.[9]

"Filename" 2 Bytes+1 Byte/Char	**MD5** 32 Bytes	**Size** 1 Byte/Char	**Bitrate** 1 Byte/Char
Frequency 1 Byte/Char	**Length** 1 Byte/Char	**Nick** 1 Byte/Char	**IP** 10 Bytes
Link-type 1 Byte/Char			

Figure 17: Search response message

8.For more detailed information about the hash value contact: http://www.faqs.org/rfcs/rfc1321.html
9.If the client has not limited the number of results being returned within the client search message, the server will return up to 100 matching filenames.

Download request messages are sent by the client to the server. The client requests to download a certain file. Thereby the client submits the filename and nickname of the file's owner.

Nick	**"Filename"**
1 Byte/Char	2 Bytes+1 Byte/Char

Figure 18: Download request message

Download ack messages are acknowledge messages from the server, which contain specific information about the desired file. This information includes nickname, IP address, port number, filename, the hash value of the file and the link-type of the client sharing this file.

Nick 1 Byte/Char	**IP** 10 Bytes
Port 1 Byte/Char	**"Filename"** 2 Bytes+1 Byte/Char
MD5 32 Bytes	**Link-Type** 1 Byte/Char

Figure 19: Download ack message

Normal downloading - after the client has received a download ack message the client, which is requesting a file, makes a TCP connection to the data port, which is specified in the download ack knowledge message received from the server. To request the file the client sends a HTTP GET message to the client containing nickname, requested filename and offset. Offset is the byte offset in the file to start the download at. It is useful to resume earlier downloads.

Nick	**"Filename"**	**Offset**
1 Byte/Char	2 Bytes+ 1 Byte/Char	1 Byte/Char

Figure 20: Normal downloading message

When the downloading process is initiated the downloading client sends a **downloading file message** and the uploading client sends a **uploading file message** to the server. When the download is completed, the downloading peer sends a **download complete message** and the uploading peer sends a **upload complete message** to the server. All four messages only consist of the header with 0 bytes data load.

3.2.3 File Transfer Protocol (FTP)

History

FTP is one of the oldest protocols on the Internet. The first proposals for a first file transfer mechanism came from M.I.T in 1971.[10] There have been several changes and updates to the protocol during the years, but in 1980 motivated by the transition from the NCP to the

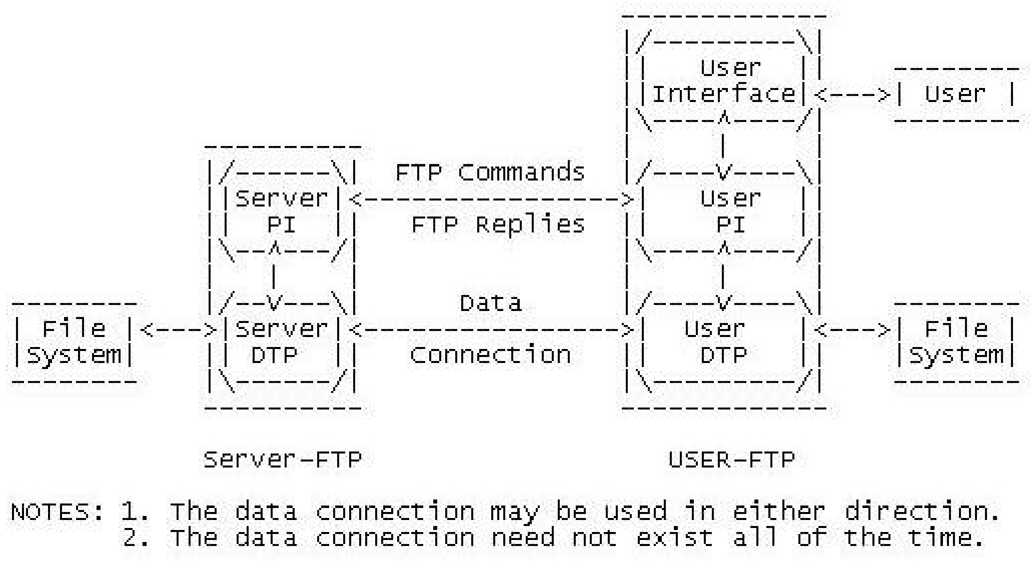
10.See RFC 114 at http://rfc.sunsite.dk

TCP as the underlying protocol, new FTP specifications were proposed in RFC 765 for use on TCP. The latest FTP specifications are described in RFC 959 in 1985.

Today FTP is commonly used to transfer web page files from their creator to the computer that acts as their server for everyone on the Internet. It's also commonly used to download programs and other files to the computer from other servers.

General Design

As described in [19] the "objectives of FTP are to promote sharing of files (computer programs and/or data), to encourage indirect or implicit (via programs) use of remote computers, to shield a user from variations in file storage systems among hosts, and to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs." Today FTP can be seen as a typical Client-Server solution as described in section 2.3.2. The server's service is mainly to provide the file system to the clients.



NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

Figure 21: FTP model[11]

In Figure21 the FTP model is shown as it is provided in [19]. If a user wants to download a file from a server, a control connection must be initiated by the user-protocol interpreter (User PI). The control connection follows the Telnet protocol specifications. At the initiation, standard commands are sent to the server by the user protocol interpreter over the control connection. Standard replies are sent back from the server protocol interpreter to the user protocol interpreter over the control connection. The communication channel from the user protocol interpreter to the server protocol interpreter is established as a TCP connection from the client to the server. FTP commands specify the parameter for the data connection, that means (transfer mode, data port etc.) and also which operations on the file system should be done (store, delete etc.). The user data transfer process (User DTP) should listen on the specified data port, while the server initiates the data connection and data transfer according to the parameters.[12] Files and information of folders and files are transferred only via the data connection. The control connection is used for the transfer of commands,

11.Source: J. Postel, J. Reynolds: File Transfer Protocol (FTP), RFC 959, October 1985.

which describe the functions to be performed, and the replies to these commands. ASCII is the default data representation type.

In FTP there are three different modes to transfer data. These are the stream mode, the block mode and compressed mode. For this analysis only the stream mode is considered. Over this mode data is transmitted as a stream of bytes.

Since it is important for this work to figure out, how FTP can provide a certain service to the client, the analysis will focus on the client's side about the effort to get connected, to search for files, to share files and to download files. Normally after a client has logged into a server, it will see the standard directory with some folders and eventually some files. If the standard directory only consists of folders the client has to send a command (e.g. CWD <new folder>) to change the directory. The server will then submit the content of the new directory over the data connection.

Below typical log files of the control connection of a FTP server is shown. This is important to evaluate how many bytes are being transferred over the control connection for a typical command. The log files show a login of a client into an FTP server, see Figure22. For log files of a directory change, a download of a file and an upload of a file see section 7.1.1 on page60.

```
(not logged in) > 220 Robin's FTP Server
(not logged in) > USER rmaly_up
(not logged in) > 331 Password required for rmaly_up.
(not logged in) > PASS ********
rmaly_up > 230 User rmaly_up logged in.
rmaly_up > SYST
rmaly_up > 215 UNIX Type: L8
rmaly_up > FEAT
rmaly_up > 500 Unknown command.
rmaly_up > PWD
rmaly_up > 257 "/" is current directory.
rmaly_up > TYPE A
rmaly_up > 200 Type set to A.
rmaly_up > PASV
rmaly_up > 227 Entering Passive Mode (192,168,0,144,41,152).
rmaly_up > LIST
rmaly_up > 150 Data connection accepted from 192.168.0.144:2040; transfer starting.
rmaly_up > 226 Transfer ok
```

Figure 22: FTP login log file

Figure22 shows the control connection communication of a client trying to login on a FTP server. Text preceded by a numbers like 220 are server replies to the client, whereas commands in capital letters are client commands submitted to the server. For a login 18 commands and replies are sent over the control connection in total, whereas the home directory information, see Figure23, is transferred over the data connection.

```
drwxr-xr-x   1 ftp      ftp             0 Mar 24 07:54 new folder
-rw-r--r--   1 ftp      ftp        121385 Mar 23 12:25 rfc959 -ftp.pdf
```

Figure 23: FTP directory information

12.The server will only initiate the data connection in active mode. If the User PI wants to establish a connection in passive mode, the server will reply which port is listening for a data connection. Then the client can initiate the data connection from its data port to the specified server data port.

Since the directory information is submitted in ASCII the representation for directory information demand 57 bytes as overhead (date, file size etc.) per folder or file and 1 byte per char for the folder or file name.

In case of a change folder command of the client, 9 messages are sent from or to the server. To initiate a file download over the data connections it needs 9 commands and replies on the control connection. For the file upload it takes 14 messages (cf. section 7.1.1 on page 60).

In Figure24 the communication overhead in bytes for login, change folder, file down and upload commands is shown. The amount of bytes being transmitted between client and server, e.g. within the login process, includes all 18 commands and replies which are sent within this process. The used byte size can vary, since usernames, passwords, FTP server names, folder names, file names and file sizes can be different.

Login Overhead
311 Bytes+3*<username>*1Byte/char+<password>*1Byte/char+<FTP server name>*1Byte/char
Change Folder Overhead
236 Bytes+3*<foldername>*1Byte/char
File Download Overhead
284 Bytes+2*<username>*1Byte/char+<password>*1Byte/char+<filesize in bytes>*1Byte/char
File Upload Overhead
352 Bytes+2*<filename>*1Byte/char

Figure 24: FTP communication overhead

3.2.4 A Qualitative Comparison

In Figure25 a qualitative comparison of Gnutella, Napster and FTP is provided, which covers characteristics, that can be applied on all of the three applications. These characteristics are data availability, data consistency, extensibility of hardware, robustness of the system and an estimation of the download performance. These characteristics can vary depending on the user and its requirements for a file sharing application. Therefore these characteristics are not meant to be cut in stone. They more resemble important characteristics in the eyes of the author of this work.

As it is stated in section 2.4.2 data availability means, that if a peer or a client needs a file for usage, the file must be available. In Gnutella, queries are propagated to only a certain number of peers, therefore not all available peers can be included into a query. So if a file is stored on one of the peers, that has not received the query, a peer will not get the file. Therefore the data availability is low. Napster maintains an index list to which all queries from clients are sent, so if a client needs a specific file, assuming that the file is stored anywhere on an active client, the client will receive a message with the location of the file. Therefore the data availability is high. In the case of FTP assuming that the server is online the data availability is high, since all files are stored there and can be accessed by the clients.

Data consistency is a characteristic, which covers the topicality of files. That means, if copies of the same file are available and in case that one of them is changed, that all copies are changed as well. Neither Napster nor Gnutella has implemented such functions. In FTP this problem is addressed with central data management. A client could upload a file, which is already stored on the server, but FTP server operator can filter out such duplicates.

The characteristic extensibility of hardware, means the ability to extend hardware resources namely storage or extra bandwidth to enhance the application deliberately. The extensibility of hardware resources, which concern clients or peers, is low. This means, when e.g., an application developer wants to extend resources of the system, the developer will not be able to extend the storage or the bandwidth of peers located anywhere on the world, since peers are autonomous and can do what they want. In case of Napster and FTP, which are using a server, the extensibility of server hardware is high, since it is centrally located and administrated by an authority. But since file sharing applications mainly have to rely on storage capacity and the storage capacity at Napster and Gnutella is provided by the peers their extensibility of hardware is low, whereas the extensibility of FTP is high because of the ability to extend the FTP server with additional hardware and bandwidth and therewith to extend the system.

Characteristics	**Gnutella**	**Napster**	**FTP**
Data availability	low	high	high
Data consistency	low	low	high
Extensibility of hardware	low	low	high
Robustness	high	medium	medium
Download performance	low	low	high

Figure 25: Qualitative comparison[13]

Robustness is a characteristic, that expresses the ability of a system to deliver the service of file sharing, while parts of the system are not working properly. Gnutella is pure a Peer-to-Peer application, therefore if, e.g. one peer does not work properly the whole system is only affected in that files on this peer are not accessible. The rest of the system remains working, so the robustness of Gnutella is high. Both Napster and FTP have a single point of failure, since both applications rely on a server. But since it can be assumed that servers today run very stable the robustness for both applications is assumed to be medium.

The download performance resembles the performance, if a client or a peer wants to download a file. In case of FTP the server will normally be connected to the Internet over a company, which is directly connected to the Internet backbone and therewith the server will have a higher bandwidth than a normal peer has. In case of Napster and Gnutella files are downloaded from other peers. Since today most peers are connected over asymmetric connections (see [4]) and therefore often have higher download bandwidth capacities [Kbps] than upload, the download performance for Napster and Gnutella is therefore lower than when using FTP.

3.3 Criteria to evaluate File Sharing Applications

If it is considered to replace a Client-Server system with Peer-to-Peer in section 2.4.3, it is stated, that a corresponding Peer-to-Peer application can have severe disadvantages and concessions. Therefore users of such applications have to find an optimal trade-off. The goal would be a quantifiable model, which would allow to evaluate each application. In the following, it is assumed that an economic evaluation of the compared applications must be the core to find a trade-off. It is intended that after applications have been evaluated economically, they will then be compared by qualitative characteristics, like the ones

13.For net sizes of ~50000 users

mentioned in section 3.2.4. That means, each user has to identify characteristics which are important to him. If these characteristics are not included in the economic evaluation, the user has to decide - e.g. in case that application A has a big disadvantage regarding its data availability, which application B does not have, but application B is 20 times more expensive than application A - if he wants to accept the shortcomings of application A in order to save a lot of money. Therefore the further work will concentrate on economic measurable characteristics. In section 2.4.2 applications have been classified into bandwidth demanding, storage demanding and compute intensive applications. Bandwidth, storage and processing cycles are characteristics, which can be economically evaluated. Storage capacity for example can be measured by evaluating the price of a harddisk for a certain storage capacity. According to section 2.4.2 and the protocol specifications from above, file sharing applications seem to fit more in the categories of storage demanding and bandwidth demanding applications, since no large computations are done in order to share files. Due to this fact the economic evaluation of an application will focus mainly on bandwidth and storage costs. Additionally, administration costs like maintenance or installation costs will also be taken into account where possible.

4 File Sharing - Economic Modeling

In this chapter economic models will be developed, which allow to compare the three in section 3.2 discussed file sharing applications. In section 4.1 a system definition will be done, in order to clearly specify the scope of the models. In section 4.2, 4.3 and 4.4 simulation models for each application will be built upon the system definition. In section 4.6 the cost models which should calculate the costs of the applications are developed. In section 4.7 limitations of the economic models will be discussed. In section 4.8 different scenarios will be defined, in order to compare the three file sharing applications economically for different settings.

4.1 System Definition

The system definition should provide a basis on which an economic model can be built and the three different applications of file sharing can be compared. Since the internet is a very heterogeneous system, assumptions are made in order to limit the complexity to a certain manageable degree. In Figure26 the system is illustrated as it is used for the economic model. In the following sections (from 4.1.1 until 4.1.4) the system and its boarders will be explained in more detail. Especially the system variables and constants with their corresponding values are described which are also listed in Table 1 on page29.

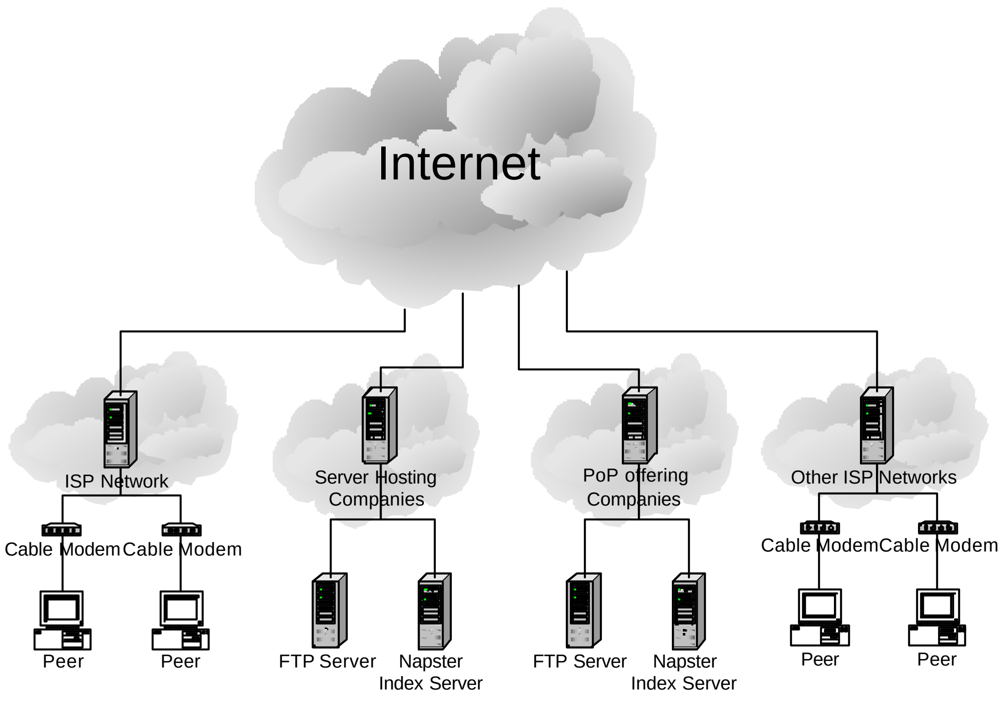


Figure 26: System

Client/Peer					
No.	**Variable name**	**Value**	**Unit**	**Explanation**	**Source**
1	hardisk_size	80	[GB]	Size of client's or peer's harddisk	Assumption
2	harddisk_price	179	[CHF]	Price of a client's or peer's harddisk	www.arp-datacon.ch
3	avg_files_size	3.5	[MB]	Average size of files, which are shared	www.centerspan.com/technology/cscc_p2pwhitepaper.pdf
4	file_length_in_bytes	7	[chars]	Number of chars needed for file size information	see avg_files_size
5	avg_files_per_peer	340	[1/peer]	Average number of files shared per client or peer	www-db.stanford.edu/~byang/pubs/p2psearch.pdf
6	replicates	4	[files]	Average number of replicas a shared file has	Assumption
7	avg_unique_files_in_system	derived	[files]	Number of files stored on the server	see avg_files_per_peer, replicates
8	dialup_speed	512	[Kbit/s]	Downstream connection from the peer or the client to the ISP	Assumption
9	dialup_price	80	[CHF/month]	Price for the connection to the the ISP per month	www.cablecom.ch
10	no	10-1000000	[peers]	Number of active peers or clients	Variable
11	queries_per_day	412	[queries]	Number of queries of a client or a peer per day	www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html
12	queryrate	derived	[1/minute]	Average number of queries per minute and peer	see queries_per_day
13	peer_downupload_rate	100-10000	[MB/month]	Effective file transfer of a client or a peer	Variable
14	download_request_rate	derived	[]	Average number of downloads per month and peer	see peer_downupload_rate, avg_file_size
15	avg_words_per_query	2.4	[words/query]	Average words per query in Gnutella and Napster	www-db.stanford.edu/~byang/pubs/hybridp2p_med.pdf
16	avg_char_per_word	5	[chars/word]	Average chars per query in Gnutella and Napster	www-db.stanford.edu/~byang/pubs/hybridp2p_med.pdf
17	avg_chars_per_file	17	[chars]	Average chars per file inluding file extension	Assumption
18	username	10	[chars]	Average chars for a username	www-db.stanford.edu/~byang/pubs/p2psearch.pdf
19	password	10	[chars]	Average chars for a password	www-db.stanford.edu/~byang/pubs/p2psearch.pdf
ISP					
20	peer_network_size	1000	[peers]	Number of active peers or clients per Internet service provider	Assumption
21	isp_profit_margin	7	[%]	Profit margin of the ISP	Assumption
Server					
Outsourced Solution					
22	monthly_payment	278	[CHF]	Monthly standard server costs	www.metanet.ch
23	monthly_traffic	50	[GB]	Monthly traffic	www.metanet.ch
24	monthly_traffic_add	100	[CHF]	Additional 100GB traffic per month	www.metanet.ch
25	storage_capacity_hosting	80	[GB]	Storage capacity of the server	www.metanet.ch
26	installation_costs_hosting	149	[CHF]	Server installation costs (single payment)	www.metanet.ch
Inhouse Solution					
27	inhouse_storage_capacity	876	[GB]	Storage capacity of the server	www.dell.ch
28	inhouse_costs	19119	[CHF]	Price of one server	www.dell.ch
29	inhouse_monthly provided traffic	1.544	[Mbit/s]	Bandwidth provided by the local loop from the server to the PoP	Assumption
30	monthly_line_lease	1500	[CHF]	Monthly costs for a local loop	www.broadbandbuyer.com/chartbusiness.htm
Internet					
31	MTU	1000	[byte]	Maximum transfer unit	ipinspace.gsfc.nasa.gov/flatsat/docs/IP-Performance.doc
32	overhead_per_packet	40	[byte]	Overhead per IP packet	www.sans.org/resources/tcpip.pdf

Table 1: System variables and constants

4.1.1 Clients/Peers

In this work clients or peers are regarded as desktop computers, which are able to run the three file sharing applications explained above. The variables of clients and peers are classified into different sections.

Storage

In the case of Gnutella and Napster the clients and peers provide a certain amount of their harddisk storage capacity in terms of files to the public. In case of FTP the clients do not provide their harddisk storage to the public. If a client wants to share a file, it can upload the file on the FTP server, where other clients can access it. If a client wants a specific file, it can search the FTP server and if the search is successful, the client can download the file. In order to compare the three applications it is assumed that Gnutella peers and Napster clients have to buy one or more additional harddisks depending on the amount of files offered. The size of the harddisk is assumed to be 80 GB at a price of 179 CHF.

As it is stated above, peers and clients share files. According to [20] files have an average file size of 3.5 MB. In [21] the average number of files a user shares is specified as 340 files, so that the average number of unique files is calculated as number of client or peers multiplied with the average number of files a user shares, divided by the number of (replicas+1). Since files are shared and downloaded by other peers, files have replicas. It is assumed that a file has an average of four replicas distributed on other peers. The average number of chars per file name is assumed to be 17 including the file type extension. If an application needs a user name and a password their length in chars is 10 as estimated in [23].

Bandwidth

In order that clients and peers can perform functions as downloading files, they need a connection to the Internet. Today there are several ways how a desktop computer can get access to the Internet (e.g. dialup users, cable modem users, DSL users, T1 users etc.). In this work every client and peer is assumed to be connected to the Internet over a cable modem connection offered by an Internet service provider. It is assumed that every peer or client has a downstream connection of 512 kbps and an upstream connection of 128 kbps. The provided download traffic is unlimited and the price per month is 80 CHF including cable modem rental.

The monthly down and upload rate of a client or a peer is a variable and can have different values. This rate resembles the effective file transfer of a client or a peer. It is assumed that a peer or a client has the same download rate as upload rate, that means, that a client downloads as much files per month as it uploads. To calculate the number of down and upload requests the monthly average down and upload rate is divided by the average file size of 3.5 MB.

Queries

In [22] the query rate for a Gnutella peer is measured and is estimated to be around 412 queries per day. Therefore the query rate for Napster and FTP is also defined to be at 412 queries per day.

In [23] the average query length is assumed to be 2.4 words per query and 5 chars per word. This value is only used for queries in Napster and Gnutella, since in FTP it is not possible to search for terms.

4.1.2 ISPs

The Internet service providers are assumed to be gateways for the clients and peers to the Internet. The traffic, that is routed over an ISP network is assumed to be the cumulated traffic of peers and clients that are customer of this ISP. There will be no statement, which proportion of this traffic is internal traffic between peers of a special ISP network and which proportion has to be routed out of the network.[14] In this work, it is assumed that an ISP harbors 1000 active peers or clients. The price an ISP has to pay for Internet exchange carriers and its own network is not explicitly regarded. It is assumed that the monthly fee clients or peers are paying for their connection covers the expenditures of the ISP.

4.1.3 Servers

FTP and Napster need a server to offer a service to their clients. In case of FTP files are stored on a server, whereas Napster only uses the server to coordinate the file sharing activities of its clients. In order to evaluate the costs, below two different approaches for the server allocation are assumed - an outsourced solution and an inhouse solution.

Outsourced Solution

In this solution the server with its connection to the Internet and with a certain storage capacity is provided by a hosting company. Stakeholders who want to offer a service like FTP or Napster have to compensate the hosting company for their service.

For this work a typical offer of a hosting company[15] is used for the outsourced solution. This standard offer includes a server with 50 GB traffic per month[16], 80 GB harddisk, IP address, DNS service, service level agreements with guaranteed net availability and guaranteed support for a monthly price of 278 CHF. The initial installation costs are at 149 CHF. Additional 100 GB traffic can be bought at 290 CHF per month. If the demand of storage capacity is more than 80 GB an additional server has to be obtained.

Inhouse Solution

The inhouse solution assumes, that stakeholders who want to offer a service like FTP or Napster buy their own server and lease their own connection to the Internet. For the server solution a typical server offering company[17] has been chosen. The chosen server has the following main specifications. The total storage capacity is 876 GB. The network interface is equipped with an embedded Gigabit NIC. The server package includes 24x7 fully-integrated service, providing problem prevention and rapid resolution services to maximize system uptime on mission critical systems and the installation costs as well. This server solution has a price of 19119 CHF. If the demand of storage capacity is more than 876 GB, an additional server has to be bought.

For the Internet connection the stakeholders normally have to lease a local loop from an ISP with a specified offered bandwidth. This local loop is directly connected to the next Point of Presence (PoP), which then is connected to the ISP backbone. Since symmetric connections like a T1 connection[18] are strongly dependent on the distance to the next PoP, it is assumed that the server is co-located directly at the ISP's PoP. A T1 Internet

14.In [13] it is stated that only 2-5% of Gnutella peers have direct link connections within their ISP network.

15.METANET GmbH, see www.metanet.ch

16.50 GB per month traffic equals an offered bandwidth of ~162 Kbps

17.Dell Ltd., see www.dell.ch

18.If the T1 connection is not fractional, the offered bandwidth is 1.544 Mbps connection is chosen. Since most of the ISPs do not have standard offers for a T1 connection the price for a monthly connection to the PoP is approximately 1500 CHF[19]. For additional bandwidth it is assumed, that stakeholders can lease more T1 connections for 1500 CHF per month and T1 connection.

If more than one server is needed for a FTP or a Napster solution the problems that may occur, e.g. to synchronize the Napster index files on all servers or to distribute files onto different servers in case of FTP, are not part of this work. For both solutions the servers' CPU performance is not taken into account since it would enhance complexity of the model extremely. In case of Napster it is assumed, that one server of the above specification can handle the CPU instructions up to 1 million clients.

4.1.4 Internet

The analysis abstracts from the topological details of the Internet. There are no assumptions on how packets are routed from one peer or client to another or to a server. But since all of the three applications are built on top of TCP and the costs as well as performance issues of the different applications should be compared, the usage of TCP and IP is taken into account. To limit complexity, protocols used in the Data Link layer are not regarded, but it is assumed that an IP packet should not exceed the maximum transfer unit (MTU) of 1000 bytes as stated in [24]. The overhead of a TCP packet as well as the overhead of an IP packet is 20 bytes, which makes a total overhead per IP packet of 40 bytes (cp. [25]).

19.cf. BroadBandbuyer, see www.broadbandbuyer.com/chartbusiness.htm

4.2 Gnutella Application Simulation Model

As described in the Gnutella protocol specification [1], the Gnutella communication utilizes 5 different descriptors which are routed in a network of peers. Therefore, it is important to model this network first. The protocol says that a peer is connected directly with n_c neighbors and propagates the pings to all its neighbors except the one the ping came from. That makes n_c-1 connections. If n_c is assumed to be constant the cumulated number of pings created by an initial ping can be calculated as in equation (2).

$$n_{cumpings} = n_c \cdot \frac{(n_c-1)^{TTL}-1}{n_c-2}$$

Equation 2

This formula assumes that each ping will never hit the same peer more than once and that the network is infinite. Otherwise depending on the network size a ping message will have hit all peers after some hops and will be dropped by that peer, where it has been twice. That's why Schollmeier proposes in [26] to use a tree structure as shown in Figure27 to describe the Gnutella network.

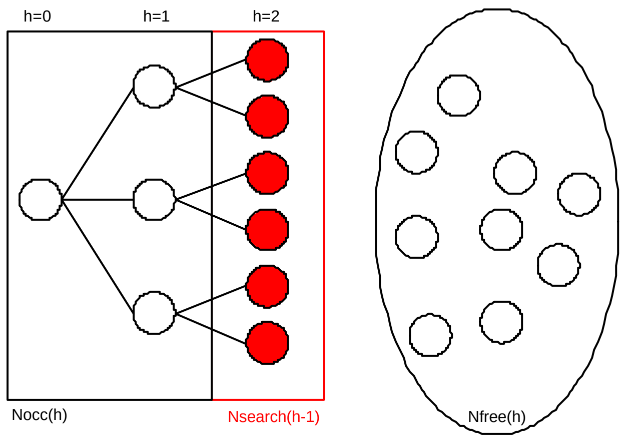


Figure 27: Gnutella tree structure

He considers the decreasing probability that a ping in a finite network can find a peer, which has never propagated this ping. For these calculations the following parameters are used:

- **No**, is the total number of active peers.
- $\mathbf{n_c}$, is the number of direct connections to other peers.
- **h**, is the maximum number of hops.
- $\mathbf{n_{search}(h)}$, is the number of peers that will receive a propagated ping at hop (h).

$$n_{search}(h) = n_{search}(h-1) \cdot (n_c-1) = n_c \cdot (n_c-1)^{h-1}$$

Equation 3

- **n_{occ}(h)**, is the total number of peers that have already been hit by a ping at hop(h).

$$n_{occ}(h) = 1 + n_c \cdot \frac{1-(n_c-1)^h}{2-n_c}$$

Equation 4

- **n_{free}(h)**, is the number of network peers that have not yet been hit by a ping at hop(h).

$$n_{free}(h) = n_{all} - n_{occ}(h)$$

Equation 5

- **p_{free}(h)**, is the probability to find a peer, which has not yet been hit by a ping at hop(h). The first part of the equation is the probability to find a free peer, whereas the second part of the equation calculates the probability that the same peer is not contacted by the remaining **n_{search}-1** peers.

$$p_{free} = \frac{n_{free}-(0.5 \cdot n_{search})}{n_{all}} \cdot \left[1-\frac{1}{n_{free}}\right]^{n_{search}-1}$$

Equation 6

Based on the parameter the cumulated number of pings as well as the total number of pongs can be calculated recursively[20], whereas the initial condition is:

```
>n_ping[h=1]:=n_c;
>p_free[h=1]:=1;
>n_search[h=1]:=n_c;
>n_free[h=1]:=no;
>n_search_pong[h=1]:=n_c;
>n_pong[h=1]:=n_c;
>con:=n_ping[h];
```

The cumulative number of pings and pongs is calculated as follows:

```
>for h from 2 to TTL while con < no do
   n_free[h]:=evalf(n_free[h-1]-n_search[h-1]);
   n_search[h]:=evalf(n_search[h-1]*p_free[h-1]*(n_c-1));
   n_ping[h]:=evalf(n_ping[h-1]+n_search[h-1]*(n_c-1));
   p_free[h]:=evalf(((n_free[h])/(no))*(1-(1/n_free[h]))^(n_search[h]-1));
```

20.Calculated in Maple.

```
n_search_pong[h]:=evalf(n_search[h-1]);
n_pong[h]:=evalf(n_pong[h-1]+n_search[h]*h);
con:=n_ping[h];
```

These calculations are only true, if the number of peers in the network is larger than the cumulated number of pings. To adjust this shortcoming an additional condition is inserted, since in a network of No peers the maximum number of cumulated pings can only be No-1 pings. For the sake of complexity reduction the number of pongs is calculated with linear approximation. For further details see section on page61.

The Gnutella protocol specification states that a peer only creates a ping during the login to probe the network, but since peers may come and leave the network and direct connections to neighbors can be lost and pings have to be sent to reconnect, the ping rate is difficult to estimate. In this work it is assumed, that every peer creates a ping every 15 minutes. The number of directly connected neighbors is 3.4 and the TTL value is 7 as mentioned in [13]. TTL is 7 for all of the descriptor types. The query rate, that means a peer makes x queries per minute, is a derived variable and has the value 0.286 [1/min] or 412 queries per day. Since queryhit messages are sent upon a query, which matches the defined conditions, query rate and queryhit rate are dependent. In [16] the proportion of measured query descriptors to queryhit descriptors is 12%. For further details see Appendix, Figure 49 on page61 and Table5 on page77.

With these equations, the knowledge of the protocol analysis and the system definition (see Table 1 on page 29 for system values and section 7.1.2 on page61 for a detailed view on the calculations) it is possible to calculate how many bytes per month are being transferred over the peer's Internet connection to or from the ISP in a worst case scenario, in which each packet is transmitted in an IP packet. It is assumed that the load of the network is distributed equally over all active peer connections. The byte volumina in [byte/month] for the different descriptors is generally calculated as follows:

$$\text{volumina}_{\text{descr}} = \text{descriptor}_{\text{rate}} \cdot n_{\text{descr}} \cdot (\text{Gnutella}_{\text{message}} + \text{TCP/IP}_{\text{overhead}}) \cdot 60 \cdot 24 \cdot 30$$

Equation 7

- **$\mathbf{n_{descr}}$**, is the cumulated number of created descriptors messages. This can be ping, pong or query messages.
- **$\mathbf{descriptor_{rate}}$**, is the descriptors' creation rate in [1/min], that can be query rate or ping rate.
- **$\mathbf{TCP/IP_{overhead}}$**, is 40 bytes for each message.
- **$\mathbf{Gnutella_{message}}$**, is the Gnutella's message size in [bytes], which is dependent on the descriptor's type.
- The last term of the equation considers the period of one month.

The formula for the direct download or upload volumina in [byte/month] is:

$$\text{volumina}_{\text{download}} = \text{peer\_downupload\_rate} + \text{ceil}\left(\frac{\text{peer\_downupload\_rate}}{\text{MTU}}\right) \cdot \text{TCPIP}_{\text{overhead}}$$

Equation 8

To measure a peer's bandwidth performance, that means the load of a peer's connection to its ISP, the volumina of all descriptors plus download traffic is calculated in Kbps. In order to to evaluate limits of the peer's connection the bandwidth usage in percentage is measured. It is assumed that half of the traffic is incoming traffic and the other half is outgoing traffic. For details see Appendix, Figure51 on page62.

Since in [16] a traffic breakdown by message type is provided, see Appendix, Figure 52 on page62, a crosscheck is implemented in the simulation model, which calculates the proportion of message types. The proportion of the created message types are close to the measured values, see Appendix, Figure 53 on page62.

4.3 FTP Application Simulation Model

In section 3.2.3 the functionality of FTP has been described. In contrast to Gnutella, FTP has not implemented a search function itself. In order to find a file, a client has to browse through the folders, that means that the folder information has to be transferred to the client first. One could assume that all files are located in the home directory, but since directory information has to be transmitted to the clients and assuming that the home directory contains a few ten thousand files, too many (mega)bytes have to be transferred to the clients only to login. Therefore it is assumed that the home directory contains 6 folders each 7 chars long, see Figure28. The second folder level consist of 26 folders with the length of 1 char, which should represent the alphabet. The third folder level consists of 100 folder with the length of 15 chars. It is assumed that all the files are distributed equally over the third folder level. The intention of this design is, if it concerns music files, that the home directory contains information about the music style, whereas the second folder level sorts the artist by name and the third folder level contains the artists. See Appendix Table 6 on page77.

So if a client logs into the server only the home directory with its 6 folders is submitted. Since FTP should be compared with Gnutella and Napster a query in FTP is defined as a "walk" from the home directory to the file level.

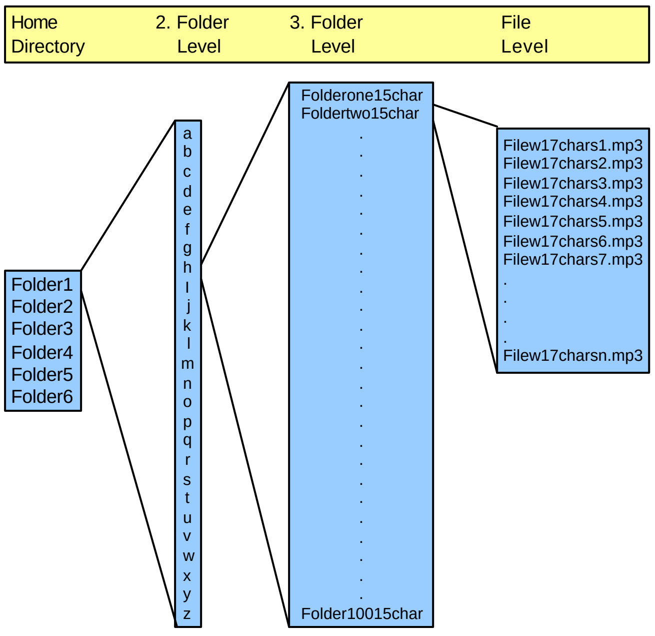


Figure 28: FTP folder structure

The query rate of FTP clients is a assumed to be the same as in Gnutella - 412 queries per day. Since most of the FTP client applications and also the FTP servers have implemented connection time-outs it is assumed, that an average client makes three server logins per day. It is not assumed, that clients cache the home directory information locally. Each time when they log in the information has to be resubmitted. Within the simulation model both control connection and data connection communication are taken into account. See Appendix, Figure 56 on page64 for details.

With this information it is possible to calculate how many bytes per month are being transferred over the client's Internet connection to or from its ISP in order to reach the FTP server. The byte volumina is calculated for three operations - login, search and download. The login volumina is calculated as:

$$\text{volumina}_{\text{login}} = \text{ftp\_connect} \cdot \text{ftp\_client\_login\_avg} \cdot 30$$

Equation 9

- **ftp_connect**, is the amount of transferred bytes, when a client logs into the server. This includes byte transfers on the data and the control connection.
- **ftp_client_login_avg**, is the number of client logins into the server (3 per day).
- **30**, considers the period of one month.

The search volumina, caused by "walks" through folders is calculated as:

$$\text{volumina}_{\text{search}} = \text{queries\_per\_day} \cdot \text{ftp\_query\_folder} \cdot 30$$

Equation 10

- **queries_per_day**, is the number of client queries (412 per day).
- **ftp_query_folder**, is the amount of bytes transferred to make a query. This includes byte transfers on the data and the control connection.
- **30**, considers the period of one month.

The volumina, caused by down and uploads can be calculated as follows:

$$\text{volumina}_{\text{download}} = \text{overhead\_control\_connection} + \text{file\_transfer}$$

Equation 11

- **overhead_control_connection**, resembles the amount of bytes, which are necessary to initiate x down and uploads per month over the control connection.
- **file_transfer**, is the amount of bytes needed per month on the data connection to transfer files.

For further details see Appendix, Figure 56 on page64.

To measure a client's bandwidth performance, that means the load of a client's connection to its ISP, all three volumina are calculated in Kbps. In order to evaluate limits of the peer's connection the bandwidth usage in percentage is measured. For details, see Appendix, Figure57 on page 65.

Since FTP is a solution, where data is stored centrally on a server, it has to be calculated how many servers are necessary to provide the same storage capacity as a Gnutella network does. Therefore in section 4.1.1 the average number of unique files was calculated in order to consider, that a Gnutella network contains many file duplicates, which are not necessary on a FTP server. With the total needed storage capacity for the server and the traffic that is being transferred from the clients to the server per month, it is possible to calculate how many outsourced server packages, each with 80GB of storage capacity, and how many extra bandwidth packages have to be bought. Therefore the traffic load of the server has to be known. The load is calculated as the product of the client's volumina in Kbps and the number of active clients. For details see Appendix, Figure58 on page 65. Analogically the number of inhouse servers and the number of additional T1 connections is calculated. Each server contains harddisk capacity of 876 GB and has a Gigabit NIC. For details see Appendix, Figure 59 on page66.

4.4 Napster Application Simulation Model

As it is stated in section 3.2.2 Napster works with different types of messages, which are exchanged between clients and server. Therefore the simulation model considers the message types, that has been described in section 3.2.2. Most of the message types, which are explained, contain a variable part regarding used bytes. In Appendix, Figure62 on page 68 an overview of the used values is given for each message type. In Appendix, Figure63 on page69, the different formulas for the message type costs, which are the costs to perform an action like to login or search for a file are listed in detail. This actions are measured in bytes and consider TCP/IP as well. The only Napster specific constants used in the simulation model, are the number of servers, the maximum number of search results and the number of client logins per day. The number of servers is limited to one server. The maximum number of search results is assumed to be 20. The number of logins per day is assumed to be the same as in FTP, what makes 3 logins per day. See Appendix Table 7 on page77.

With these formulas and the system definition it is possible to calculate how many bytes per month are being transferred over the client's Internet connection to or from the ISP as well as the traffic to the server. It is assumed that the load of the network is distributed equally over all active client connections. The byte volumina is calculated with regard to perform the three operations - login, search and download. Therefore the following calculations have been implemented in the simulation model:

The byte volumina per month for client search request and search response messages is calculated as follows:

$$\text{volumina}_{\text{message}} = \text{message}_{\text{rate}} \cdot \text{message\_type\_costs\_in\_bytes} \cdot 60 \cdot 24 \cdot 30$$

Equation 12

- **message$_{\text{rate}}$**, is the messages' creation rate per minute, this can be, e.g. the query rate.
- **message_type_costs**, these are the costs measured in bytes to send one client search request and one search response message.
- The last term of the equation considers the period of one month.

The byte volumina per month for download request, download ack, down and uploading file and down and uploading complete messages can be generally calculated as:

$$\text{volumina}_{\text{download setup}} = \text{download}_{\text{number}} \cdot \text{message\_type\_costs\_in\_bytes}$$

Equation 13

- **download$_{\text{number}}$**, is the number of file downloads a client initiates per month.
- **message_type_costs**, these are the costs measured in bytes, which are needed to setup a file download.

The byte volumina per month for the login procedure can be calculated as follows:

$$\text{volumina}_{\text{login}} = \text{logins\_per\_day} \cdot \text{message\_type\_costs\_in\_bytes} \cdot 30$$

Equation 14

- **logins_per_day**, is the number of client's logins per day.
- **message_type_costs**, are the costs to send login information to the server and the costs to submit information about shared files.
- The last term of the equation considers the period of one month.

The formula for the direct download or upload volumina is:

$$\text{volumina}_{\text{download}} = \text{client\_downupload\_rate} + \text{ceil}\left(\frac{\text{client\_downupload\_rate}}{\text{MTU}}\right) \cdot \text{TCPIP}_{\text{overh}}$$

Equation 15

The byte volumina for the server per month can be expressed as the sum of all clients volumina minus the volumina for direct download, since it does not bother the server connection. This result is multiplied with the number of active clients:

$$\text{server\_volumina} = (\text{total\_client\_volumina-volumina}_{\text{download}}) \cdot \text{no}$$

Equation 16

- **total_client_volumina**, is the sum of all clients volumina.
- **volumina$_{\text{download}}$**, is the volumina for direct downloads between clients.
- **no**, is the number of active clients.

For calculation details regarding the volumina, see Appendix, Figure 64 on page69.

To measure a client's bandwidth performance, that means the load of a client's connection to its ISP, all client volumina are calculated in Kbps. In order to evaluate limits of the client's connection the bandwidth usage in percentage is measured. Since Napster is a solution, where a server takes an intermediary part in the system, it has to be calculated how many extra bandwidth packages in case of an outsourced solution have to be bought to deliver a good service to the clients. Therefore the traffic load of the server has to be known. The load of the server is calculated as the server's volumina (see above), but in Kbps as unit.

The number of extra bandwidth packages then can be calculated as server volumina minus the standard bandwidth package divided by the extra bandwidth package size. Analogically the number of additional T1 connections in case of an inhouse solution is calculated.

For calculation details about the bandwidth performance see Appendix, Figure65 on page70.

4.5 Cost Distribution Analysis

In this section it is analyzed which stakeholders have to pay for what within the three compared file sharing applications. This analysis is done in order to distribute the costs truthfully on the stakeholders afterwards. Figure29 is provided as reference to illustrate which stakeholders are involved in the different systems.

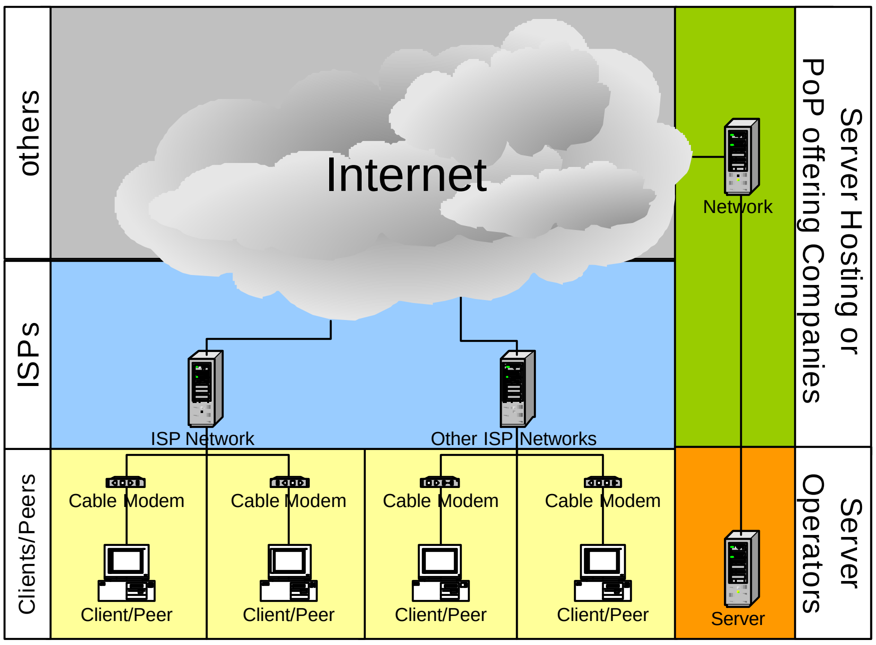


Figure 29: Costs distribution

4.5.1 Gnutella

Regarding Figure 29 one can point out three parties, which are normally involved when using Gnutella. These are the peers, which are in expectation to stand to benefit from running Gnutella, the ISPs, which are offering a network access to the peers and others. Others are meant to be Internet backbone companies. The Internet cloud is not fully covered by these companies since some ISPs maintain their own backbones.

In this work it is assumed, that the peers have to pay for their connection to their ISP and for storage capacity (see section 4.1.1). Since the ISPs only route traffic that is caused by the peers, the ISPs have to provide the infrastructure. In the case of traffic that has to be routed over a network of Internet backbone companies, the ISPs have to pay for their services. It is assumed that the monthly connection price of the peers covers the ISPs' expenses regarding own infrastructure and the use of Internet backbone services.

Due to the fact, that peers share files and allow other peers to download these files, they both have to pay for it, since both have to use their connection to the ISP to transfer a file.

All kind of messages (pings, pongs, queries) in the Gnutella network mean costs regarding bandwidth for peers that are sending/receiving such messages.

4.5.2 FTP

When using FTP as a file sharing solution there are five parties, which are normally involved, see also Figure 29:

The clients, which are using an FTP server for file sharing. The ISPs, which are offering a network access to the clients. FTP server operators, which offer a platform (file system) for file sharing to its clients and Server Hosting or PoP (Point of Presence) offering companies, which provide in the case of a hosting company the complete infrastructure (server, storage, connection, maintenance etc.). In the case of a PoP offering company, they provide only the connection from a server to the Internet. Others are again the Internet backbone companies.

As for Gnutella, it is assumed that the monthly connection price of the peers covers the ISPs' expenses regarding own infrastructure and the use of Internet backbone services.

FTP server operators have to pay either a monthly fee to a server hosting company or the operator has to invest in one or more servers, which then are connected to the Internet via a PoP offering company for a performance (bandwidth) depending fee. Different from Gnutella is, that clients only have to pay for their own actions, since other clients do not communicate directly with each other. In return the FTP server operators have to pay for each client action.

4.5.3 Napster

Regarding the usage of Napster one can say, that basically the same (five) parties are involved as when using FTP. The only difference are the Napster server operators, which offer a platform (index list) for file sharing to its clients, whereas a FTP server operator offers a file system.

The difference to Gnutella and FTP is, that clients only have to pay for their own action except, if a client has been selected as a download candidate. Then, if a download between clients has been initiated, both the client which is uploading as well as the client which is downloading have to pay for this action. Although the load regarding effective file transfer has been taken away from the server, the Napster operators have to pay for the rest of the traffic that basically consists of messages.

4.6 Costs Models

In this section the cost models for the three different applications are described. Basically the costs drivers focus on bandwidth and storage costs, which have to be paid by the stakeholders. Cost elements, which are applicable for all three applications are described right now:

In order to calculate the clients' or peers' bandwidth costs it is necessary to calculate the costs to transfer one byte of information from the peer to its ISP or vice versa. This can be done by applying the following equation, in which the dialup price per month is in [CHF] and the dialup speed in [Kbps].

$$\text{bytecost} = \frac{\text{dialupprice}}{\frac{(\text{dialupspeed} \cdot \text{kilo} \cdot (60 \cdot 60 \cdot 24 \cdot 30))}{8}}$$

Equation 17

As described in the system definition, see section 4.1, every peer (in the case of Gnutella and Napster) has to buy its own harddisk to participate in the network. This calculation is done to evaluate how much money a peer has to pay for sharing and storing files. It is assumed that the harddisk is depreciated (linearly) over a period of 4 years. This results in 3.72 CHF per harddisk, month and peer as “operating expenses”.

4.6.1 Gnutella Cost Model

In order to evaluate the costs for a peer, which is running Gnutella two cost elements are considered:

1 Bandwidth costs

2 Storage costs

With the knowledge of the byte costs as defined above, it is possible to evaluate the bandwidth costs per month and peer as the sum of ping-, pong-, query-, queryhit- and file transfer volumina multiplied with the byte costs. For calculation details, see Appendix, Figure54 on page page 63. The monthly storage costs for one peer are 3.72 CHF as described above.

4.6.2 FTP Cost Model

In order to evaluate the costs for a client, which is using FTP only one cost element is considered:

1 Bandwidth costs

The bandwidth costs consist of the sum of the byte volumina calculated in section 4.3, multiplied with the byte costs.

In order to evaluate the costs for a FTP service operator, which is running a FTP server two cost elements are considered:

1 Bandwidth costs

2 Storage costs

In section 4.3 it is described how the necessary server storage or the necessary server number is calculated for the outsourced and the inhouse solution. For the outsourced solution the standard package is as defined in section 4.1.3. The inhouse specifications are also described in section 4.1.3. In the case of the inhouse solution the investment into the server hardware is depreciated (linearly) over a period of four years. In section 4.3 it is explained how the server traffic load can be calculated and how many extra bandwidth packages in the case of an outsourced solution or how many extra T1 connections have to be leased in the case of an inhouse solution. With this knowledge it is possible to calculate the costs a FTP server operator has to pay for a certain number of active clients.

4.6.3 Napster Cost Model

In order to evaluate the costs for a client, which is using Napster two cost elements are considered:

1 Bandwidth costs

2 Storage costs

The bandwidth costs consist of the sum of the byte volumina, calculated in section 4.4, multiplied with the byte costs. The monthly storage costs for one peer are 3.72 CHF, just as for Gnutella.

In order to evaluate the costs for a Napster server operator, which is running a Napster server two cost elements are considered:

1 Bandwidth costs

2 Storage costs

Above it is stated that the Napster network gets along with only one server, therefore the storage costs have to be calculated for this server. In section 4.4 it is explained, how the server traffic load can be calculated and how many extra bandwidth packages in the case of an outsourced solution or how many extra T1 connections have to be leased in the case of an inhouse solution. With this knowledge it is possible to calculate the costs a Napster service provider has to pay for a certain number of active clients.

4.6.4 Cost Perspectives

The costs of Napster and FTP are broken down into clients' or peers', ISPs', server operators' and overall perspective. The Gnutella perspectives are the same except without the server operators' perspective, since Gnutella does not need a server.

The peers' or clients' total costs include the bandwidth and/or storage costs that they have to pay. With the assumption that an ISP routes the cumulated amount of traffic of the clients or the peers, which are directly connected to it and the assumption that the monthly connection fees cover the ISP's expenses, the total costs of an ISP could be calculated as the clients' or peers' bandwidth costs multiplied by the number of directly connected clients or peers minus a profit margin. The server operators' costs are calculated in case of an inhouse solution as the sum of monthly costs for all bought servers plus the costs for the local loop fee. In case of an outsourced solution the monthly costs consist of a basic fee for a certain storage capacity, a certain bandwidth and if necessary, more server or additional bandwidth packages. The overall costs are calculated as the peers' or clients' total costs, which can include bandwidth and/or storage costs, multiplied with the number of all active peers plus possible server operator costs. All three cost perspectives are calculated in a second step without considering the initial investment in hardware. This state is reached after 4 years, when all investments are depreciated. For the calculations of the cost perspectives, see Appendix, from section 7.1.2 until 7.1.4.

4.7 Limitations of the Models

Within the above sections three different cost models have been defined. Although the system definition has clearly set the scope of the models, the limits of the modeled application compared to the real life applications have to be assessed.

According to the measurements done in [4] the largest part of Napster clients are using a cable modem to participate in the network, approximately 33%. See Appendix, Figure68 on page73. No actual figures could be found for Gnutella, but it is assumed that the distribution would not differ very much. So the assumption within the system definition, that only cable modem users are included into the model is very optimistic, since at least 20% of connected Napster clients have reported bandwidths lower than 64 Kbps. This could negatively influence the real life behavior of an application like Napster or Gnutella, since direct downloads from such peers would be terribly slow and it is very likely that such peers will quit their connection before the download is finished.

Within the simulation models above, latencies, e.g. of replies, are not considered. It is also assumed that all generated traffic is distributed equally over a month. But in real life at certain times traffic peaks may occur at certain periods during the day. So bandwidth performance measurements, which are done above, have to be regarded carefully. From a cost perspective such performance problems in real life could cause additional costs, when a client is not able to find a file because of overloaded peers, the client will resend queries, which then create additional costs. In case of the Gnutella model, the number of created pings and pongs is a real theoretical value, since today there are several different applications on the market, which have implemented the Gnutella protocol. Some of them have made modifications, e.g. ping caching or the implementation of superpeers. Therefore the Gnutella simulation model must be regarded as a model, which has been strictly aligned to the protocol specification mentioned in [1].

It is also assumed that the monthly down and upload rate is equal per peer and client. Although in a real life closed system the overall download rates and upload rates per month must be equal, since data cannot be destroyed or leave the system. But the assumption that each peer or client downloads as much as it uploads cannot be verified in real life applications like in Napster or Gnutella, since many users of such systems participate in the network but do not allow other users to download files. This fact is also known as “freeloading”.

Although these are quite notable limitations, it is assumed that the cost models developed above resemble pretty well the costs, created by such applications under the defined assumptions.

4.8 Cost Scenarios (simulations)

Within this section cost scenarios are specified. In the system definition there are two variables defined: The number of active clients and peers and the monthly down and upload rate of clients and peers. Since these two variables seem to have large impact to the economics. Therefore two scenarios will be done, considering these variables.

Due to the fact that Napster and FTP are applications, that offer their clients access to all files at best, while Gnutella is restricted due to its TTL settings[21] an additional scenario will be performed, which should evaluate what happens, if Gnutella extended its “horizon” to compete with the other two applications.

First Scenario

The first scenario assumes a monthly down and upload rate of 100 MB. While this value is constant, the net size, that means the number of active peers and clients will run from 10 to 10^n while n<=6.

Second Scenario

The second scenario assumes a monthly down and upload rate of 10000 MB. While this value is constant, the net size will run from 10 to 10^n while n<=6.

Third Scenario

The third scenario should prove what happens, if Gnutella clients would increase their TTL setting from default 7 to 12. Twelve is chosen, since in [13] the longest Peer-to-Peer path is found to be 12. The monthly down and upload rate is assumed to be 100 MB. While this value is constant the net size will run from 10 to 10^n while n<=6.

21.Depending on the net size only parts of the whole network are reachable.

5 Stakeholder Analysis

Within this chapter a cost and a performance analysis for each application cost model is done. The analysis will contain the peer/client perspective as well as the ISP, server and overall cost perspective. In order to figure out, if there are any performance limitations regarding bandwidth, the uplink load [%] of clients/peers[22] and the server load [Mbps] compared with the server number is considered. The evaluation includes the analysis which application is the cheapest for which net size. It also considers, why the costs of an application are higher than of those of another application and which are the influencing factors.

5.1 Peer/Client

5.1.1 Cost for the next 4 years

The following figures represent the monthly costs a peer or a client has to pay for the usage of the regarded applications. Since from a costs perspective, clients and peers are not affected by a server operator's choice of inhouse or outsourced solution, the costs are equal from their perspective.

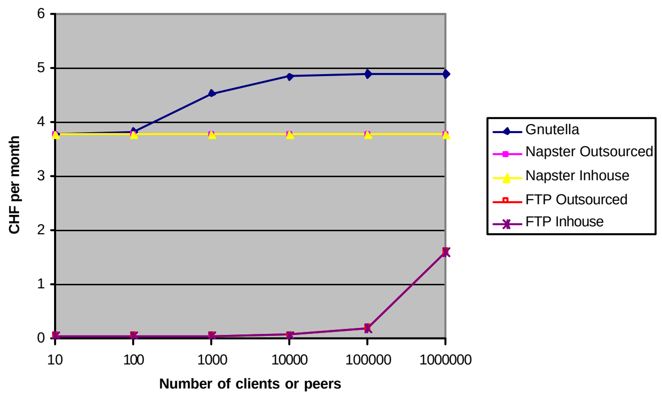


Figure 30: Scenario 1: Client/Peer cost perspective < 4 years

From a client cost perspective FTP as a solution for file sharing seems to be the cheapest application for all network sizes. This can be explained with the fact, that Gnutella and Napster participants have to invest in a new harddisk. This investment is not necessary in the case of FTP.

For scenario one, which has a low down and upload rate per month, it is interesting to observe that the costs for Gnutella resemble an s-curve. This can be explained with the TTL settings. TTL is responsible for the number of created pings and pongs as well as queries. The more descriptors are sent into the network, the more a peer has to pay. For a TTL of 7 the upper bound of created pings is around 1111. That means, that a TTL of 7 takes stronger influence to the costs up to a network size of 1111 peers. From there on the costs converge to an upper bound.

22.Since it is defined that a client/peer downloads as much as it uploads its' limiting factor is the upload bandwidth.

FTP has a steep curve for larger network sizes. This can be explained with the definition of the folder structure of the FTP server. Since every additional peer means additional files, these files have to be distributed equally over the folders. Every FTP query consists of a walk through the directories, so the more files are stored on the file level, the more file information (bytes) have to be submitted for each query. In the case of Napster clients can calculate with constant costs for every net size.

In Figure 31 a monthly down and upload rate of 10000 MB is considered. Basically the cost curves are like the ones in Figure30, except that they are shifted upwards. This can be explained with the additional traffic caused by the higher down and upload rate.

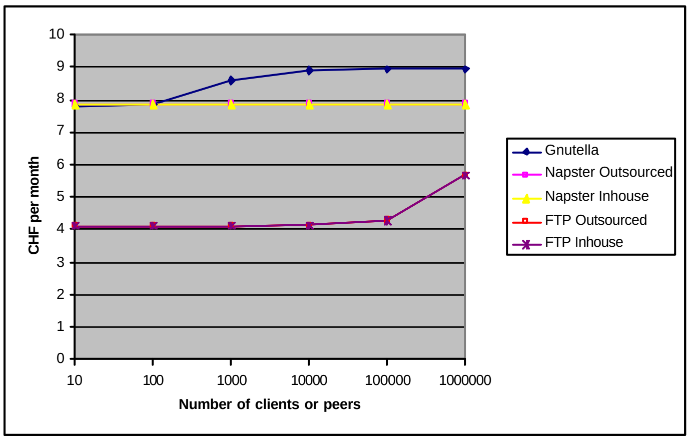


Figure 31: Scenario 2: Client/Peer cost perspective < 4 years

In Figure32 a monthly down and upload rate of 100 MB is assumed but the TTL setting of Gnutella is set to 12. The cost curves for Napster and FTP remain the same as in scenario one, see Figure30. The s-curve of Gnutella gets steeper compared to the other scenarios, but it also has a boundary value for higher net sizes. So peers have to pay a large amount of extra money to increase their network knowledge.

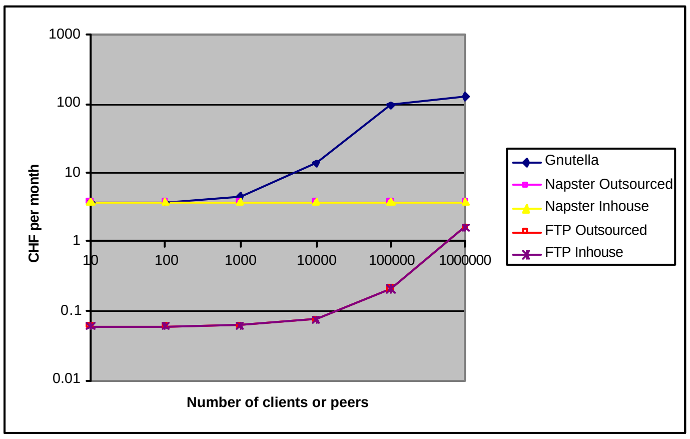


Figure 32: Scenario 3: Client/Peer cost perspective < 4 years

5.1.2 Costs without Investment

Regarding today's usage of Napster and Gnutella costs, without regarding the investment into infrastructure, represent the costs clients and peers pay, since they already have the proper infrastructure. Basically these costs only include the clients' and peers' bandwidth costs. As in Figure33 is visible for a monthly down and upload rate of 100 MB, all three applications have approximately equal costs for network sizes <100. Gnutella costs increase very steep until network sizes < 1111. From there on the costs remain the same. In contrast to the Gnutella curve, FTP has a flat slope at the beginning but due to the folder structure the costs of FTP will exceed the ones of Gnutella for large network sizes. Napster has again a constant cost distribution for different network sizes.

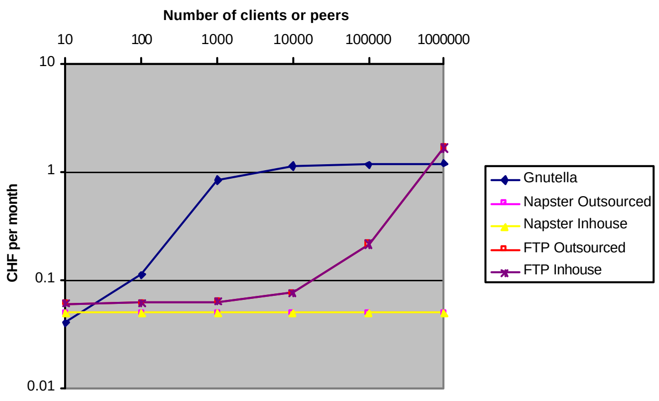


Figure 33: Scenario 1: Client/Peer cost perspective > 4 years

Figures for scenario two and three are available in the Appendix, section 7.2.1 on page73.

As conclusion for scenario two it can be said that the cost curves are basically alike the ones in Figure33, except that they are shifted upwards. This can be explained with the additional traffic caused by the higher monthly down and upload rate. For scenario three, where the TTL setting of Gnutella is set to 12 the same considerations mentioned in the explanations for Figure 32 can be applied here as well. The extra amount of network knowledge in the case of Gnutella makes this application the most expensive for the regarded network sizes.

5.1.3 Performance

The performance is measured in order to find out limits regarding the clients'/peers' bandwidth capacity. Therefore the uplink load is measured for different network sizes. Since the uplink statistic curves for scenario one and two show alike curves than the cost curves and do not show any concerns, that a client or peer cannot handle the amount of uplink traffic, these figures are shown in the Appendix, Figure 71 on page74 and Figure72 on page74. For scenario three, where the TTL setting of Gnutella is 12, it can be seen in Figure34, that from network sizes larger than 50000, the uplink capacity of the peers is exceeded. That means, regarding the system definition, that Gnutella peers would be totally overloaded with uplink traffic.

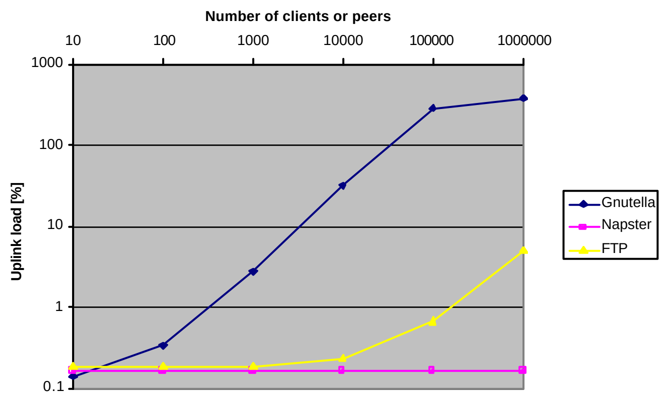


Figure 34: Scenario 3: Uplink load performance

5.2 Server

5.2.1 Cost for the next 4 years

The following figures represent the monthly costs a server operator has to pay, to offer a service like Napster or FTP. For both application the costs for an outsourced as well as for an inhouse solution are considered. Since Gnutella operates with no server the server costs for Gnutella are zero. Therefore the third scenario has no impact on the server costs.

Regarding Figure 35 it can be stated that a server operator would be less charged when using Napster as file sharing application. Up to a network size to ~1000 the costs for either the outsourced variants or the inhouse variants are constant. The constant costs can be explained with the over capacity of the specified server packages. In case of the FTP inhouse solution no additional server have to be bought until more than 876 GB of storage is needed. Also the standard bandwidth connection is sufficient up to 1000 clients. In case of the Napster inhouse solution only one server is needed. Therefore the break in the curve can be reduced to additional bought bandwidth (see section 5.2.3). Both outsourced solutions are cheaper for small networks but get more expensive than the inhouse solutions for larger networks, although the curves seem to converge slowly. The gap between outsourced and inhouse solution for larger network sizes could be explained with a profit margin, which a web hosting company normally has.

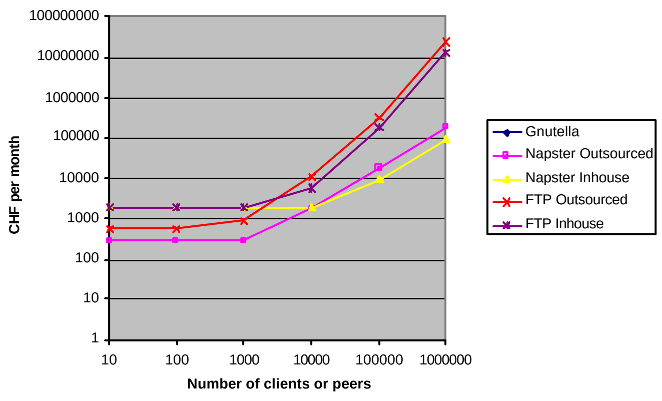


Figure 35: Scenario 1: Server cost perspective < 4 years

For higher down and upload rates, which are used in scenario 2, it can be generally stated, that Napster is the cheaper solution for a server operator, except the network size is smaller than ~60 clients. For this size, a FTP outsourced solution would be cheaper than a Napster inhouse solution. However, compared to Figure 35, it can be concluded, that the larger the down and upload rate is the bigger the cost gap between Napster and FTP from a server operator’s perspective.

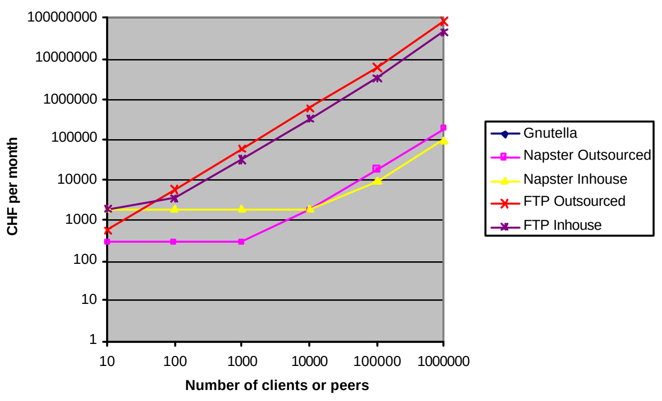


Figure 36: Scenario 2: Server cost perspective < 4 years

5.2.2 Costs without Investment

In case that server operators operate their system longer than four years, without buying new ones and without spending money into maintenance in case of an inhouse solutions, the costs that occur for the three scenarios can be seen in section 7.2.3 on page 74. It can be stated that a Napster server operator pays much less than a FTP server operator. When only regarding the inhouse solution, it is visible for both applications, that for larger net sizes the (bandwidth) costs will increase due to more traffic to the servers.

5.2.3 Performance

In Table2 the total server traffic[23] for scenario 1 and scenario 2 is listed in [Mbps]. The Napster traffic is basically the same in both scenarios. The little gap is caused by client messages, that are signalling the download status to the server. The FTP traffic differs in scenario 1 and scenario 2, since the server has 9900 MB of additional traffic per month and peer.

Scenario 1	Network size					
	10	100	1000	10000	100000	1000000
Napster [Mbps]	0.00097656	0.00947266	0.09472656	0.95605469	9.56738281	95.6767578
FTP [Mbps]	0.00476563	0.04780273	0.48828125	5.98535156	169.698242	12680.6611

Scenario 2	Network size					
	10	100	1000	10000	100000	1000000
Napster [Mbps]	0.00097656	0.00966797	0.09667969	0.97265625	9.7265625	97.2666016
FTP [Mbps]	0.32226563	3.22558594	32.2734375	323.838867	3348.22656	44465.9502

Table 2: Total server load

In Table 3 the total number of used servers, for both solutions is presented. It can be concluded that in case of the outsourced solution the number of servers increase more

23.This is the traffic, which has to bought from a server hosting company or a PoP offering company.

steep, due to the fact, that the servers do have less storage capacity, than the inhouse servers.

Number of servers	Network size					
	10	100	1000	10000	100000	1000000
Gnutella	0	0	0	0	0	0
Napster Outsourced	1	1	1	1	1	1
Napster Inhouse	1	1	1	1	1	1
FTP Outsourced	1	1	3	30	291	2906
FTP Inhouse	1	1	1	3	27	266

Table 3: Number of servers

Since in Table2 the total server load and in Table3 the number of used servers is presented Figure 37 and Figure38 show the total server load distributed over the number of used servers. This is done, to check, if a server is overloaded. Since the inhouse servers contain Gigabit NICs and the outsourced servers should contain at least 100 Megabit NICs, both scenario infrastructures can handle the traffic up to a network size of one million.

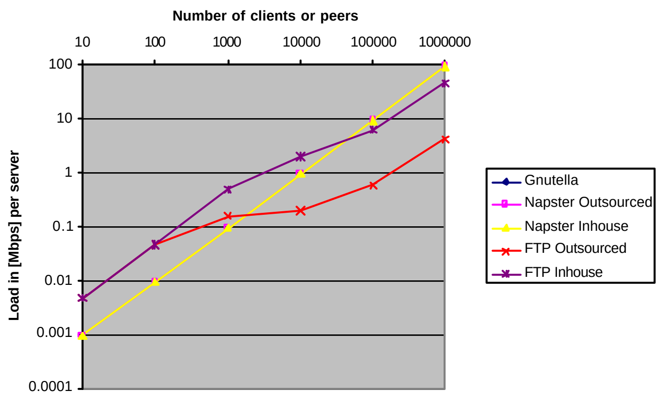


Figure 37: Scenario 1: Server load per used server

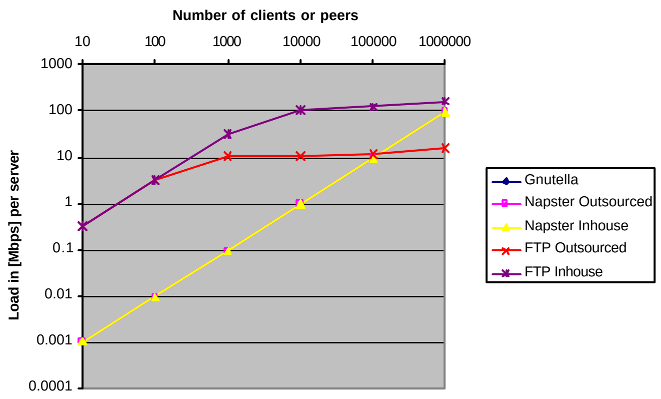


Figure 38: Scenario 2: Server load per used server

5.3 ISP

Within the system definition the number of active peers and clients in an ISP's network has been defined as 1000. The ISP costs therefore resemble the costs, an ISP has to pay, so that its' clients can use Gnutella, Napster or FTP. Since the ISP does not have to invest in storage, the costs basically include bandwidth costs.

5.3.1 Costs

It can be concluded out of the ISPs' cost figures, that for net sizes larger than ~100, Gnutella causes significant costs. This can be explained with the traffic caused by pings, pongs, queries and queryhit. This takes negative influence on the cost performance of an ISP. So for higher TTL values the ISP costs will increase significantly. The FTP traffic increases extremely in large net sizes and exceeds the Gnutella costs, but this can be traced back to the folder structure on the FTP server. Napster again shows a constant cost performance. Exemplarily of this behavior the results of scenario 1 are shown below. Results of scenario 2 and 3 can be found in the Appendix, Figure76 on page75 and Figure77 on page 76.

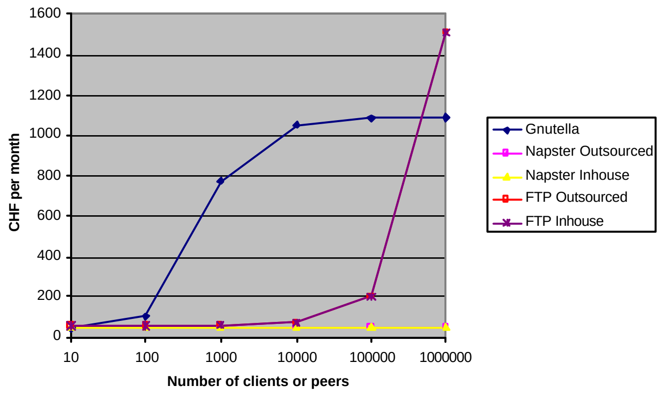


Figure 39: Scenario 1: ISP cost perspective

5.4 Overall

The overall costs resemble the costs which incurred by clients/peers and server operators, if a server is used. The overall costs are therefore interesting, to evaluate, which of the three compared applications is the cheapest due to its implementation. These costs are also interesting, in case a "closed" stakeholder group wants to implement a system like Napster, Gnutella or FTP and wants to know, which is the cheapest one for their needs.

5.4.1 Cost for the next 4 years

Regarding Figure 40 it can be concluded, that Gnutella seems to be the cheapest overall solution for network sizes up to ~200 and a monthly down and upload rate of 100 MB. From 100 until ~10000 clients both the outsourced and the inhouse FTP solution should be the preferred applications regarding their costs. When having higher network sizes Napster have cost advantages. The lower costs for Gnutella for small network sizes can be explained with the lower investment price of peers' harddisks compared to the investment into a central server. This advantage of the FTP architecture gets weaker with an increasing net size due to the traffic load of the server, which is mainly caused by effective file transfers and queries. The Napster architecture with its implementation takes the load for effective file transfers away from the server. This and the more efficient query function, seem to be the reasons, why Napster competes better regarding large network sizes.

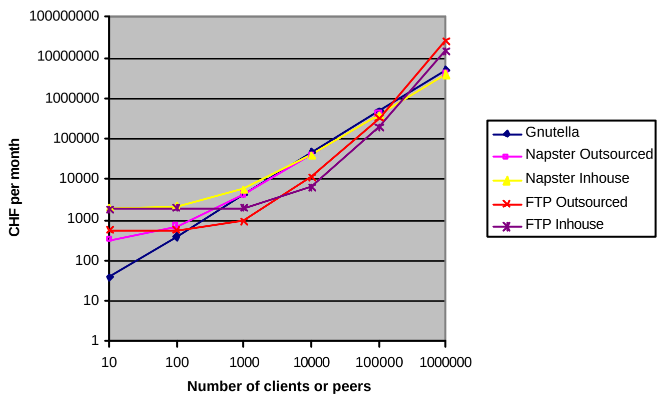


Figure 40: Scenario 1: Overall cost perspective < 4 years

In Figure41 the overall costs for scenario 2 are presented. For network sizes up to 1000 peers Gnutella is again the cheapest overall solution. For larger network sizes Napster has cost advantages regardless which solution (inhouse or outsourced) is used. Nevertheless the cost gap between Napster and Gnutella is only a fractional amount. FTP creates high costs, when the monthly down and upload rate is high. This is mainly caused by additional bandwidth, that has to be bought from hosting companies or PoP offering companies.

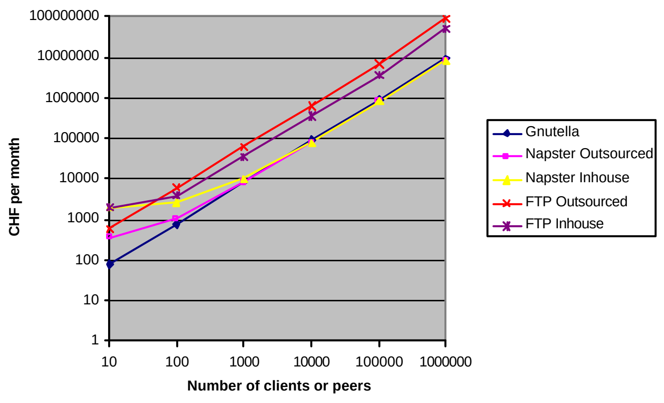


Figure 41: Scenario 2: Overall cost perspective < 4 years

In Figure42 the cost results of the third scenario are shown. The costs for Napster and Gnutella are the same as for scenario 1, whereas the costs for Gnutella are different. The third scenario has been defined to figure out what happens, if a Gnutella peer at least in theory could reach all other peers. Gnutella remains its' cost leadership for network sizes up to ~200. Then again FTP is the cheaper solution. Compared to scenario 1 the costs for Gnutella do not converge with the Napster costs. This can be explained with the TTL setting, which leads to a higher number of cumulated pings as well as queries descriptors and therewith to a higher number of pongs and queryhits.

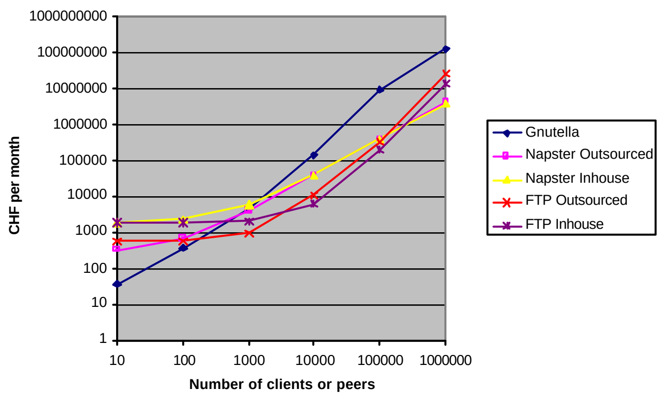


Figure 42: Scenario 3: Overall cost perspective < 4 years

Since the cost differences in some of the overall cost figures are not exactly visible the exact overall costs are listed in Figure 43 for all three scenarios. The costs are listed in [CHF/month].

Scenario 1	**Net size**					
	10	**100**	**1000**	**10000**	**100000**	**1000000**
Gnutella	37	384	4560	48570	489674	4900836
Napster Outsourced	318	659	4063	39556	395641	3957940
Napster Inhouse	1936	2276	5680	39723	387648	3874398
FTP Outsourced	571	577	905	11229	330302	24868033
FTP Inhouse	1898	1904	1960	6461	195975	14047075

Scenario 2	**Net size**					
	10	**100**	**1000**	**10000**	**100000**	**1000000**
Gnutella	78	790	8627	89246	896429	8968391
Napster Outsourced	359	1066	8131	80233	802706	8028599
Napster Inhouse	1976	2683	9748	80400	794424	7943657
FTP Outsourced	612	6204	63264	635394	6570504	87267730
FTP Inhouse	1939	3811	34529	356146	3691327	48996092

Scenario 3	**Net size**					
	10	**100**	**1000**	**10000**	**100000**	**1000000**
Gnutella	37	384	4629	139434	9480342	126385622
Napster Outsourced	318	659	4063	39556	395641	3957940
Napster Inhouse	1936	2276	5680	39723	387648	3874398
FTP Outsourced	571	577	905	11229	330302	24868033
FTP Inhouse	1898	1904	1960	6461	195975	14047075

Figure 43: Overall cost perspective: Accurate data

5.4.2 Costs without Investment

For the overall costs after all investments have been depreciated (after 4 years) and no more money is invested into server maintenance, in case of an inhouse solution, see Appendix, section 7.2.5 on page76. As bottom line from these figures it can be said, that Napster and Gnutella are the cheapest solutions for scenario one and two basically for all net sizes. This can be explained with the fact, that the traffic is distributed over the net, while the server operators still have to pay for additional bandwidth which is mostly used for effective file transfers. In scenario three it can be seen that the costs for a higher network knowledge of Gnutella peers exceed the costs of Napster and FTP. This can be reduced to the additional traffic caused by the higher TTL setting.

5.5 Economic Summary

Within the following tables only scenario one and two are considered. In Table 4 the result of the stakeholder analysis are summarized in order to propose the implementation of the cheapest application for a certain net size and monthly down and upload rate depending on the perspective. This summary considers the investment into hardware. As one can see, each perspective favors another application regarding costs, which could represent an area of conflict.

	Scenario One		**Scenario two**	
	Net size	**100 MB/month**	**Net size**	**10000 MB/month**
Client/Peer perspective < 4 years	0-10^6	FTP	0-10^6	FTP
Server perspective < 4 years	0-10^6	Gnutella	0-10^6	Gnutella
ISP perspective	0-10^6	Napster	0-10^6	Napster
Overall perspective < 4 years	< 100	Gnutella	< 900	Gnutella
	100-50000	FTP Outsourced	900-10000	Napster outsourced
	50000-300000	FTP Inhouse	10000-10^6	Napster Inhouse
	300000-10^6	Napster Inhouse		

Table 4: Proposed implementation regarding costs and net size (< 4 years)[24]

5.6 Qualitative Evaluation

Regarding the qualitative characteristics mentioned in section 3.2.4 the choice from a client perspective is clear - FTP. Although the comparison is a little bit unfair due to the fact that the cost gap between FTP and Napster, which would be the second choice, is mainly caused by the investment of a peer in the new harddisk. In case of FTP these costs have been burdened on the FTP service operator, which has to pay for the storage. But if the clients do not have to pay for the usage of an FTP server this is the best choice also for high down and upload rates. When regarding the costs without investment a client would be generally better served from a cost perspective with Napster for low and high monthly down and upload rates. But since the cost gap between Napster and FTP is not that significant, the client should figure out, if the shortcomings of Napster like low data consistency or a worse download performance can be accepted or if it is willing to pay more in order to be better served.

24.Why is Gnutella the proposed architecture from a server perspective? Since server operators do not want to spend much money in order to provide a file sharing solution, their best choice would be Gnutella, where they wouldn't have to pay for a server.

From an ISP perspective the choice should be Napster. The results show that Napster is the application that creates much less traffic than the other applications. For the ISP perspective qualitative characteristics are not taken into account, since its main task is to provide an Internet connection to the clients and peers.

For the overall perspective Gnutella seems to be the cheapest solution for small net sizes. And for small networks, e.g., for up to 900 peers with a monthly down and upload rate of 10000 MB, Gnutella with a TTL of 7 will reach every peer with its queries. Therefore, it has to be discussed if the shortcomings of data consistency and extensibility of hardware and the download performance can be accepted in order that money can be saved compared to Napster and FTP.

From ~100 until 300000 users and low down and upload rates (100MB), FTP is the proposed architecture. For higher down and upload rates FTP is much more expensive and therefore the usage of Gnutella or Napster should be discussed regardless of both applications’ shortcomings, since the costs gap could be over a couple of 100000 CHF per month on an overall perspective.

6 Conclusions, Recommendations and Outlook

In this chapter conclusions, recommendations and an outlook on possible future works regarding file sharing applications based on economical measurements are presented.

6.1 Conclusions and Recommendations

The initial question of this work was whether it is possible to replace all Client-Server systems with Peer-to-Peer system. In section 2.4.3 it was stated that it could be possible but associated with disadvantages and concessions in some classes of application. Since the focus of this work has been led on file sharing, simulation models for the different file sharing applications have been developed in order to quantify, if current Peer-to-Peer application perform better than comparable Client-Server applications. These simulation models have been made economically measurable, so that different file sharing applications could be judged by their created costs. It has been found out, that from an overall perspective pure Peer-to-Peer applications like Gnutella are economically interesting for small networks. Client-Server solutions like FTP have been identified as solutions, which deliver a high quality of service to its clients regarding data availability, consistency etc. The compared Peer-to-Peer applications could not achieve this level of quality, but hybrid Peer-to-Peer applications like Napster have been identified as highly cost effective systems, which beat the Client-Server (FTP) basically in all perspectives.

In section 5.5 an area of conflict was described, when regarding the proposed applications from the different perspectives. The client would prefer a FTP solution, since this is the cheapest one. Server operators would like to chose Gnutella, since then, they wouldn't have to invest into a server and could save money. ISPs favor Napster as a file sharing solution, since this is an application causes fewest traffic on the ISPs' networks, in contrast to Gnutella. Therefore ISPs should think about to subsidize their users, if the do not use Gnutella. If one assumes that an ISP has hosted 1000 clients in its network and these clients would participate in file sharing application like Gnutella, Napster and FTP with a total community of 10000 peers/clients, an ISP would have monthly costs of about 1048 CHF for Gnutella, 49 CHF for Napster and 71 CHF for FTP, see Figure44. So if all 1000 Gnutella peers which are hosted in the ISP's network, switch to the Napster solution, the ISP could subsidize the peers with 999 CHF per month at best.

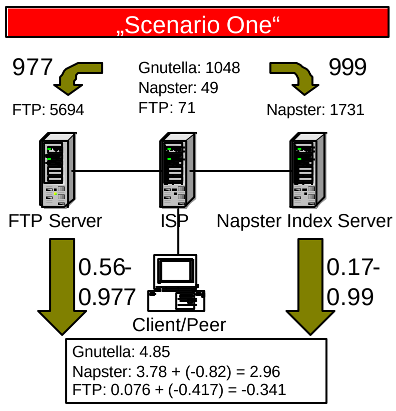


Figure 44: Resolve "Area of Conflict"

This would make 0.99 CHF per peer and month. Server operators have the problem that they offer a service to their clients/peers but have to carry the costs. Therefore, server operators should consider to pass their expenses to their clients/peers. In this specific example the Napster server operator has monthly costs of 1731 CHF. He could distribute these costs over the 10000 clients, which benefit from the server. This would make 0.17 CHF per month and peer. As one can see on Figure44. The same considerations can be done for FTP. Finally the client will decide which application will be the best for him, regarding cost issues as well as quality issues. In the example here, a client would benefit from using FTP, since he would even get paid for the usage.

But, if a scenario with higher down and upload rates like scenario two is considered, see section 4.8, server operator costs increase extremely, so that the costs passed by the operator to its clients for using this service, exceed the costs that clients would have to pay for a Napster solution, even if the Napster server operator passes the costs to its clients. Therefore a client will consider, if the advantages of a Client-Server system can be dropped in order to benefit from the cost advantage of a hybrid Peer-to-Peer architecture.

As bottom line of this work the following recommendation can be given, when thinking about to replace a Client-Server solution with Peer-to-Peer or vice versa in the class of file sharing:

Both architectures should be compared on an economical basis first, which then allows to compare cost advantages of one architecture against possible shortcomings regarding quality issues like data availability etc. Then stakeholders can decide which architecture they want to use. Normally the clients/peers will be the crucial factor when deciding which file sharing architecture will be used, since they benefit directly from the usage. But if, e.g., an ISP has a high traffic load (high costs) on its network due to Gnutella peers and this would lead to bandwidth capacity problems on its network an ISP should consider or should try to subsidize its Gnutella peers for using another file sharing solution in order to prevent an network extension.

6.2 Outlook

This work has proved, that it is possible to quantify Client-Server and Peer-to-Peer system economically, at least in the class of file sharing. The costs were mainly based on storage and bandwidth costs, but characteristics like security, data availability and data consistency were not included into the economic model.

As described in section 4.7, the cost models which were used to compare the three file sharing applications have a couple of limitations and assumption, like the assumption, that all peers are connected to the Internet via a cable modem, as well as it is assumed that all peers download as much per month as they upload. Therefore, future economic related works in the class of file sharing applications should first try to make the application models more accurate. This especially means, that the topology of such systems have to be taken into account, e.g., models should include the different peer connection types, as well as the number of “freeloaders”. A very important point for further works should focus on the ISPs. Since peers are spread all over the world data has to be routed over backbones to other ISP networks. Since interconnection traffic is normally more expensive for an ISP than traffic on its own network the worldwide topology of the peers must be included in further economic models. In [13] it is stated that only 2-5% of Gnutella peers have direct link connections within their ISP networks, therefore further research should focus on solutions for minimizing the interconnection traffic between ISPs while delivering the same quality to their users.

Another improvement proposal for the models presented in this work concerns FTP. FTP is an old protocol and it does not deliver a high functionality, e.g., missing search function. For a future economic comparison it should be considered to replace the FTP solution with a web server. On such a server a client can search, e.g., over a Google interface, for files. If there are files available, they could be downloaded over http. The author believes, that this action would have significant impact on the costs of such a Client-Server system.

As a second step, characteristics like data availability should be made quantifiable in order to include them into the economic model. In case of data availability different aspects should be made measurable, e.g., “what happens if a user cannot find a file or cannot download it?”. Which additional actions will occur from this user? And which costs arise?

By including such characteristics into an economic model, a comparison between Client-Server systems and Peer-to-Peer systems will become more objective and more powerful.

7 Appendix

7.1 Additional Figures

7.1.1 FTP Log Files

```
rmaly_up  > CWD new folder
rmaly_up  > 250 CWD command successful. "/new folder" is current directory.
rmaly_up  > PWD
rmaly_up  > 257 "/ new folder" is current directory.
rmaly_up  > PASV
rmaly_up  > 227 Entering Passive Mode (192,168,0,144,224,84).
rmaly_up  > LIST
rmaly_up  > 150 Data connection accepted from 192.168.0.144:2041; transfer starting.
rmaly_up  > 226 Transfer ok
```

Figure 45: FTP change folder log file

```
rmaly_up  > REST 1024
rmaly_up  > 350 REST supported. Ready to resume at byte offset 1024.
rmaly_up  > REST 0
rmaly_up  > 350 REST supported. Ready to resume at byte offset 0.
rmaly_up  > PASV
rmaly_up  > 227 Entering Passive Mode (192,168,0,144,76,101).
rmaly_up  > RETR new.txt
rmaly_up  > 150 Data connection accepted from 192.168.0.144:2043; transfer starting for new.txt (0 bytes).
rmaly_up  > 226 Transfer ok
```

Figure 46: FTP download log file

```
rmaly_up  > TYPE I
rmaly_up  > 200 Type set to I.
rmaly_up  > PASV
rmaly_up  > 227 Entering Passive Mode (192,168,0,144,198,187).
rmaly_up  > STOR wsftpsi.dll
rmaly_up  > 150 Data connection accepted from 192.168.0.144:2045; transfer starting for wsftpsi.dll.
rmaly_up  > 226 File received ok.
rmaly_up  > TYPE A
rmaly_up  > 200 Type set to A.
rmaly_up  > PASV
rmaly_up  > 227 Entering Passive Mode (192,168,0,144,250,16).
rmaly_up  > LIST
rmaly_up  > 150 Data connection accepted from 192.168.0.144:2046; transfer starting.
rmaly_up  > 226 Transfer ok
```

Figure 47: FTP upload log file

7.1.2 Gnutella Model: Maple Code and Additional Figures

```
> n_ping[h=1]:=n_c;
> p_free[h=1]:=1;
> n_search[h=1]:=n_c;
> n_free[h=1]:=no;
> n_search_pong[h=1]:=n_c;
> n_pong[h=1]:=n_c;
> con:=n_ping[h];
> for h from 2 to TTL while con < no do
   n_free[h]:=evalf(n_free[h-1]-n_search[h-1]);
   n_search[h]:=evalf(n_search[h-1]*p_free[h-1]*(n_c-1));
   n_ping[h]:=evalf(n_ping[h-1]+n_search[h-1]*(n_c-1));
   p_free[h]:=evalf(((n_free[h])/(no))*(1-(1/n_free[h]))^(n_search[h]-1));
   n_search_pong[h]:=evalf(n_search[h-1]);
   n_pong[h]:=evalf(n_pong[h-1]+n_search[h]*h);
   con:=n_ping[h];
   TTL_max:=h;
   od;
> a_ping:=n_ping[TTL_max-1];
> b_ping:=n_ping[TTL_max];
> a_pong:=n_pong[TTL_max-1];
> b_pong:=n_pong[TTL_max];
> if (n_ping[TTL_max] > no)
   then
   n_ping[TTL_max]:=evalf(a_ping+(no-(a_ping+1)));
   n_pong[TTL_max]:=evalf((a_pong-b_pong)*((n_ping[TTL_max]-b_ping)/(a_ping-
   b_ping))+b_pong);
   else
   n_ping[TTL_max]:=n_ping[TTL_max];
   n_pong[TTL_max]:=n_pong[TTL_max];
   end if;
```

Figure 48: Cumulated pings and pongs

```
> TTL:=7;
> n_c:=3.4;
> pingrate_eval:=15;
> queries_per_day:=412;
> query_proportion:=12;
> queryrate:=evalf(queries_per_day/(24*60));
> pingrate:=1/(pingrate_eval);
> download_request_rate:=evalf(ceil((peer_downupload_rate*mega)/(2*avg_files_size*mega))/
   (30*24*60));
```

Figure 49: Gnutella constants and derived variables

```
> ping:=evalf(n_ping[TTL_max]*(overhead_per_packet+23)*pingrate*(60*24*30));
> pong:=evalf(n_pong[TTL_max]*(overhead_per_packet+37)*pingrate*(60*24*30));
> query:=evalf((n_ping[TTL_max]*(overhead_per_packet+25+
  avg_words_per_query*avg_char_per_word)*queryrate*(60*24*30)));
> queryhit:=evalf((n_ping[TTL_max]*(query_proportion/
  100)*(overhead_per_packet+58+avg_chars_per_file)*queryrate*(60*24*30)));
> download:=evalf((peer_downupload_rate*mega)+
  (ceil((peer_downupload_rate*mega)/MTU)*(overhead_per_packet)));
```

Figure 50: Gnutella monthly byte volumina

```
> peer_overhead:=evalf((ping+pong+query+queryhit);
> connectionload:=evalf((peer_overhead+((peer_downupload_rate)+(
  ceil((peer_downupload_rate*mega)/MTU)*(overhead_per_packet))/mega))*mega*8/
  ((60*60*24*30)*kilo));
> bandwidth_usage_128kbps:=evalf(((connectionload/2)/128)*100);
> bandwidth_usage_512kbps:=evalf(((connectionload/2)/512)*100);
```

Figure 51: Gnutella connection performance

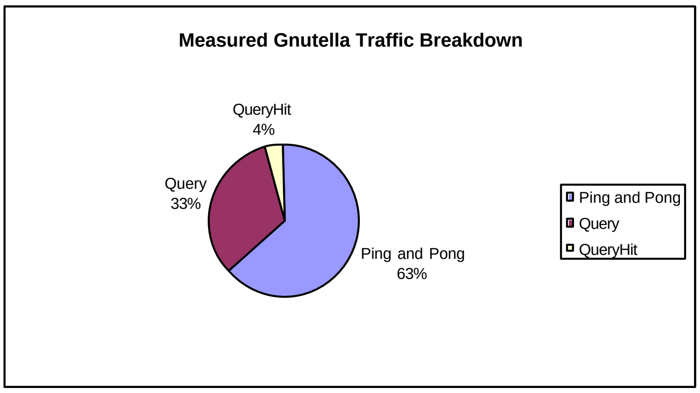


Figure 52: Measured Gnutella Traffic Breakdown

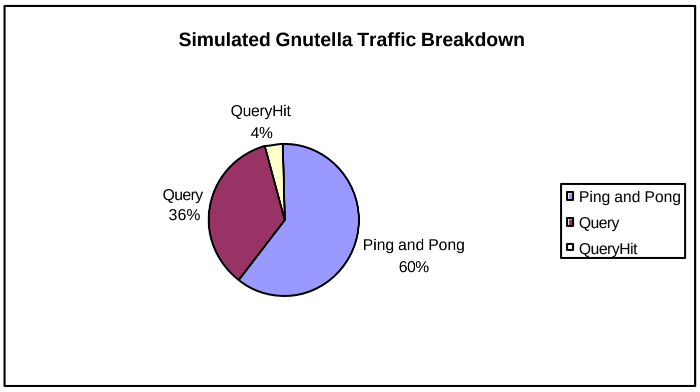


Figure 53: Simulated Gnutella Traffic Breakdown

```
>ping_costs:=(n_ping[TTL_max]*(overhead_per_packet+23)*pingrate*(60*24*30)*byte_costs);
>pong_costs:=(n_pong[TTL_max]*(overhead_per_packet+37)*pingrate*(60*24*30)*byte_costs);
>query_costs:=((n_ping[TTL_max]*(overhead_per_packet+25+avg_words_per_query*avg_char_per_
   word)*queryrate*(60*24*30)*byte_costs));
>queryhit_costs:=((n_ping[TTL_max]*(query_proportion/
   100)*(overhead_per_packet+58+avg_chars_per_file)*queryrate*(60*24*30)*byte_costs));
>download_costs:=((peer_downupload_rate*mega*byte_costs)+(ceil((peer_downupload_rate*
mega)/MTU)*(overhead_per_packet))*byte_costs);
>bandwidth_costs:=ping_costs+pong_costs+query_costs+queryhit_costs+download_costs;
```

Figure 54: Gnutella monthly bandwidth costs

```
Peer's perspective
Total Costs per month and peer for the next 4 years
   > total_cost:=evalf(bandwidth_costs+storage_costs);
Total Cost per month and peer without investment
   > total_cost_plus:=evalf(bandwidth_costs);

ISP's perspective
Total Costs for the ISP per month excluding storage costs
   > isp_total_cost:=evalf(total_cost_plus*peer_network_size*(100-isp_proft_margin)/100);

Gnutella overall perspective
Total Costs per month and all peer for the next 4 years
   > total_cost_all_peers:=evalf((bandwidth_costs+storage_costs)*no);
Total Cost per month and all peer without investment
   > total_cost_plus_all_peers:=evalf(bandwidth_costs*no);
```

Figure 55: Gnutella total costs

7.1.3 FTP Model: Maple Code and Additional Figures

```
FTP constants and derived variables
> ftp_client_login_avg:=3;
> ftp_number_folder_1:=6;
> ftp_number_folder_2:=26;
> ftp_number_folder_3:=100;
> ftp_number_folder_4:=100;
> ftp_char_folder_1:=7;
> ftp_char_folder_2:=1;
> ftp_char_folder_3:=15;
> FTP_server_name:=10;
> File_size_in_bytes:=7;
> ftp_number_files:=evalf(avg_unique_files_in_system);
> ftp_login_overhead:=evalf((311+3*username+password+FTP_server_name)+
  18*overhead_per_packet);
> ftp_change_folder:=evalf(236+9*overhead_per_packet);
> ftp_file_download_overhead:=evalf((284+2*username+password+avg_chars_per_file+
  file_length_in_bytes)+9*overhead_per_packet);
> ftp_file_upload_overhead:=evalf((352+2*avg_chars_per_file)+18*overhead_per_packet);
> ftp_downupload_requests:=evalf(ceil((peer_downupload_rate*mega)/
  (avg_files_size*mega)));
> MTU_folder_1:=evalf(ftp_change_folder+3*ftp_char_folder_1+ceil(ftp_number_folder_1*(
  57+ftp_char_folder_2)/MTU)*(overhead_per_packet));
> MTU_folder_2:=evalf(ftp_change_folder+3*ftp_char_folder_2+ceil(ftp_number_folder_2*
  (57+ftp_char_folder_2)/MTU)*(overhead_per_packet));
> MTU_folder_3:=evalf(ftp_change_folder+3*ftp_char_folder_3+ceil(ftp_number_folder_3*
  (57+ftp_char_folder_3)/MTU)*(overhead_per_packet));
> MTU_folder_4:=evalf(ceil((ftp_number_files/
  (ftp_number_folder_1*ftp_number_folder_2*ftp_number_folder_3)*
  (57+avg_chars_per_file))/MTU)*(overhead_per_packet));
> ftp_connect:=evalf(ftp_login_overhead+((ftp_number_folder_1*(57+ftp_char_folder_1)+
  MTU_folder_1)));
> ftp_query_folder:=evalf((ftp_number_folder_2*(57+ftp_char_folder_2)+
  MTU_folder_2+ftp_char_folder_3*(57+ftp_char_folder_3)+MTU_folder_3+
  (ftp_number_files*(57+avg_chars_per_file)/
  (ftp_number_folder_1*ftp_number_folder_2*ftp_number_folder_3))+MTU_folder_4));
  (ftp_number_folder_1*ftp_number_folder_2*ftp_number_folder_3)*
  (57+avg_chars_per_file))/MTU)*(overhead_per_packet));
>  ftp_connect:=evalf(ftp_login_overhead+((ftp_number_folder_1*(57+ftp_char_folder_1)+
  MTU_folder_1)));
> ftp_query_folder:=evalf((ftp_number_folder_2*(57+ftp_char_folder_2)+
  MTU_folder_2+ftp_char_folder_3*(57+ftp_char_folder_3)+MTU_folder_3+
  (ftp_number_files*(57+avg_chars_per_file)/
  (ftp_number_folder_1*ftp_number_folder_2*ftp_number_folder_3))+MTU_folder_4));
> storage_capacity_server:=evalf((avg_unique_files_in_system*avg_files_size*mega)/giga);

Byte Volumina
> ftp_connect:=evalf(ftp_connect*ftp_client_login_avg*(30));
> ftp_search:=evalf((queries_per_day*30*ftp_query_folder));
> ftp_download:=evalf(ftp_downupload_requests/
  2*(ftp_file_download_overhead+ftp_file_upload_overhead)+
  (peer_downupload_rate*mega)+(ceil((peer_downupload_rate*mega)/
  MTU)*(overhead_per_packet)));
```

Figure 56: FTP constants, derived variables and byte volumina

```
> ftp_client_bandwidth_total:=evalf(ftp_download+ftp_search+ftp_connect);
> ftp_client_overhead_download_kbps:=evalf(ftp_client_bandwidth_total*8/
    ((60*60*24*30)*kilo));
> ftp_bandwidth_usage_proportion_128kbps:=evalf(((ftp_client_overhead_download_kbps/2)/
    128)*100);
> ftp_bandwidth_usage_proportion_512kbps:=evalf(((ftp_client_overhead_download_kbps/2)/
    512)*100);
```

Figure 57: FTP bandwidth performance

```
> ftp_server_bandwidth_total:=evalf((ftp_client_bandwidth_total*no)/giga);
In [GB/month]
> ftp_server_bandwidth_total_kbps:=evalf((ftp_server_bandwidth_total*giga*8)/
    ((60*60*24*30)*kilo));
Number of required Servers
> if (storage_capacity_hosting > storage_capacity_server)
    then server_number:=1
    else server_number:=evalf(ceil(storage_capacity_server/storage_capacity_hosting))
    end if;
Traffic, that is provided by the necessary standard server packages
> ftp_server_monthly_traffic_provided:=evalf(((server_number*(monthly_traffic*
    (8*giga/((60*60*24*30)*kilo))))-ftp_server_bandwidth_total_kbps));

Additional Traffic which have to be bought from the Hosting Company:
> if (0 < ftp_server_monthly_traffic_provided)
    then traffic_add:=1
    else
    traffic_add:=evalf(-1*ceil(ftp_server_monthly_traffic_provided/
    ((monthly_traffic)*8*giga/(60*60*24*30*kilo))))
    end if;
```

Figure 58: FTP: Number of outsourced packages

```
Number of required Servers
> if (inhouse_storage_capacity > storage_capacity_server)
   then server_inhouse_number:=1
   else server_inhouse_number:=evalf(ceil(storage_capacity_server/
   inhouse_storage_capacity))
   end if;
> ftp_server_inhouse_monthly_traffic_provided:=
   evalf((inhouse_monthly_provided_traffic*kilo)-(ftp_server_bandwidth_total_kbps));

Additional Traffic which have to be bought:
> if (0 < ftp_server_inhouse_monthly_traffic_provided)
   then traffic_inhouse_add:=0
   else traffic_inhouse_add:=evalf(-1*ceil(ftp_server_inhouse_monthly_traffic_provided/
   (inhouse_monthly_provided_traffic*kilo)))
   end if;.

Server Performance Check. If result < 0 then Server perfomance ok
> ftp_inhouse_performance_check:=evalf(ftp_server_bandwidth_total_kbps*kilo/
   (server_inhouse_number*giga));
```

Figure 59: FTP: Number of inhouse servers and connections

```
Outsourced Solution
> traffic_add_costs:=evalf(traffic_add*ftp_monthly_traffic_add);
> ftp_server_hosting_costs:=evalf(server_number*monthly_payment);
> ftp_server_installation_costs:=evalf(server_number*installation_costs_hosting);
> ftp_server_installation_costs_monthly:=evalf(ftp_server_installation_costs/
   (storage_depreciation*12));

Inhouse Solution
> traffic_inhouse_add_costs:=evalf(traffic_inhouse_add*inhouse_monthly_line_lease);
> ftp_inhouse_server_costs:=evalf((server_inhouse_number*inhouse_costs)/(
   storage_depreciation*12));
> ftp_inhouse_performance_check:=evalf(ftp_server_bandwidth_total_kbps*kilo/
   (server_inhouse_number*giga));
```

Figure 60: FTP inhouse and outsourced costs

```
Client's perspective

> ftp_client_total_costs:=evalf(ftp_client_bandwidth_total_costs);

Server perspective

Outsourced: Total Costs per month for the next 4 years
> ftp_server_total_costs:=evalf(traffic_add_costs+ftp_server_hosting_costs+
   ftp_server_installation_costs_monthly);

Outsourced: Total Costs per month after 4 years (without investment)
> ftp_server_total_costs_plus:=evalf(traffic_add_costs+ftp_server_hosting_costs);

Inhouse: Total Costs per month for the next 4 years
> ftp_inhouse_server_total_costs:=evalf(traffic_inhouse_add_costs+
   ftp_inhouse_server_costs+inhouse_monthly_line_lease);

Inhouse: Total Costs per month after 4 years (without investment)
> ftp_inhouse_server_total_costs_plus:=evalf(traffic_inhouse_add_costs+
   inhouse_monthly_line_lease);

ISP's perspective

Total Costs for the ISP per month excluding storage costs
> ftp_isp_total_cost:=evalf(ftp_client_total_costs*peer_network_size*(100-
   isp_proft_margin)/100);

Overall perspective

Outsourced: Total Costs per month for the next 4 years
> ftp_overall_costs:=evalf((ftp_client_total_costs*no)+ftp_server_total_costs);

Total Costs per month after 4 years (without investment)
> ftp_overall_costs_plus:=evalf((ftp_client_total_costs*no)+ftp_server_total_costs_plus);

Inhouse: Total Costs per month for the next 4 years
> ftp_inhouse_overall_costs:=evalf((ftp_client_total_costs*no)+
   ftp_inhouse_server_total_costs);

Inhouse: Total Costs per month after 4 years (without investment)
> ftp_inhouse_overall_costs_plus:=evalf((ftp_client_total_costs*no)+
   ftp_inhouse_server_total_costs_plus);
```

Figure 61: FTP Total costs

7.1.4 Napster Model: Maple Code and Additional Figures

Message	**Message receiver**	**Total bytes (incl. message header and "blanks")**	**Description**
Login	Server	46 bytes	Nick: 10 bytes Password: 10 bytes Port: 4 bytes Client Info: 10 bytes Link-Type: 2 bytes
Client notification of shared files	Server	79 bytes	Filename: 17 bytes MD5: 32 bytes Size: 7 bytes Bitrate: 4 bytes Frequency: 4 bytes Time: 4 bytes
Client search request	Server	137 bytes	Filename contains: 32 bytes Max_Result: 13 bytes Filename contains: 32 bytes Linespeed contains: 22 bytes Bitrate: 15 bytes Freq: 12 bytes
Search response	Client	106 bytes	Filename: 19 bytes MD5: 32 bytes Size: 7 bytes Bitrate: 4 bytes Frequency: 4 bytes Length: 4 bytes Nick: 10 bytes IP: 10 bytes Link-Type: 2 bytes
Download request	Server	36 bytes	Nick: 10 bytes Filename: 19 bytes
Download ack	Client	90 bytes	Nick: 10 bytes IP: 10 bytes Port: 4 bytes Filename: 19 bytes MD5: 32 bytes Link-Type: 4 bytes
Downloading file	Server	5 bytes	No payload
Uploading file	Server	5 bytes	No payload
Download complete	Server	5 bytes	No payload
Upload complete	Server	5 bytes	No payload

Figure 62: Napster message type costs: Length in bytes

```
Client Variables
> napster_max_search_results:=20;

Client Derived Variables:
> napster_client_login_avg:=ftp_client_login_avg;
   Message type costs:
> napster_login:=evalf((46+(79+(avg_chars_per_file))*avg_files_per_peer)
   +ceil((46+(79+(avg_chars_per_file))*avg_files_per_peer)/MTU)*(overhead_per_packet));
> napster_query:=evalf(109+2*((avg_words_per_query*avg_char_per_word)+2)+
   ((overhead_per_packet)));
> napster_download_request:=evalf(17+((avg_chars_per_file)+2)+((overhead_per_packet)));
> napster_updownload_start_complete:=evalf((4+(overhead_per_packet))*2);
> napster_download_p2p:=evalf((peer_downupload_rate*mega)+
   (ceil((peer_downupload_rate*mega)/MTU)*(overhead_per_packet)));

Server Variables
> napster_server_number:=1;

Server Derived Variables:
> napster_server_inhouse_number:=evalf(napster_server_number);
   Message type costs:
> napster_query_server_response:=evalf((87+((avg_chars_per_file)+2))*
   napster_max_search_results+(ceil((87+((avg_chars_per_file)+2))*
   napster_max_search_results)/MTU)*(overhead_per_packet));
> napster_download_request_server_response:=evalf(71+((avg_chars_per_file)+2)+
   ((overhead_per_packet)));
```

Figure 63: Napster constants and derived variables

```
Client Byte volumina
> napster_connect:=evalf(napster_login*ftp_client_login_avg*(30));
> napster_search:=evalf((queryrate*(60*24*30)*(napster_query+
   napster_query_server_response)));
> napster_download_request_replies:=evalf(download_request_rate*(60*24*30)*
   (napster_download_request+napster_download_request_server_response));
> napster_download:=evalf((peer_downupload_rate*mega)+(ceil((peer_downupload_rate*mega)/
   MTU)*(overhead_per_packet)));
> napster_finish_message:=evalf(ceil(peer_downupload_rate/
   avg_files_size)*napster_updownload_start_complete);
> napster_client_overhead:=evalf((napster_connect)/mega);
> napster_client_bandwidth:=evalf(napster_download+napster_search+
   napster_download_request_replies+napster_finish_message+
   (napster_client_overhead*mega));

Server byte volumina
In [bytes/month]
>napster_server_bandwidth_total:=evalf(((napster_client_bandwidth-napster_download)*no));
```

Figure 64: Napster byte volumina

```
Client Overhead in [MB/month]:
> napster_client_overhead:=evalf((napster_connect)/mega);
Traffic consumption in [Byte/month] and client:
> napster_client_bandwidth_total:=evalf(napster_download+napster_search+
   napster_download_request_replies+napster_finish_message+
   (napster_client_overhead*mega));
Server Traffic Consumption in [Byte/month]:
> napster_server_bandwidth_total:=evalf(((napster_client_bandwidth-napster_download)*no))

Client total traffic:
> napster_client_overhead_download_kbps:=evalf(napster_client_bandwidth_total*8/
   ((60*60*24*30)*kilo));
   All bandwidth usage figures in [%]
> napster_bandwidth_usage_proportion_128kbps:=
   evalf(((napster_client_overhead_download_kbps/2)/128)*100);
> napster_bandwidth_usage_proportion_512kbps:=
   evalf(((napster_client_overhead_download_kbps/2)/512)*100);

Napster Server traffic:
> napster_server_bandwidth_total_kbps:=
   evalf((napster_server_bandwidth_total*8)/((60*60*24*30)*kilo));

Dedicated Server Hosting - Outsourcing
> napster_server_monthly_traffic_provided:=evalf(((napster_server_number*
   (monthly_traffic*(8*giga/((60*60*24*30)*kilo))))-
   napster_server_bandwidth_total_kbps));
   Additional Traffic which have to be bought from the Hosting Company:
> if (0 < napster_server_monthly_traffic_provided)
   else napster_traffic_add:=
   evalf(-1*ceil(napster_server_monthly_traffic_provided/((monthly_traffic)*8*giga/
   (60*60*24*30*kilo))))
   end if;

Server In house solution
> napster_server_inhouse_monthly_traffic_provided:=
   evalf((inhouse_monthly_provided_traffic*kilo)-(napster_server_bandwidth_total_kbps));
   Additional Traffic which have to be bought from a PoP offering company:
> if (0 < napster_server_inhouse_monthly_traffic_provided)
   then napster_traffic_inhouse_add:=0
   else napster_traffic_inhouse_add:=evalf(-1*
   ceil(napster_server_inhouse_monthly_traffic_provided
   (inhouse_monthly_provided_traffic*kilo)))
   end if;
```

Figure 65: Napster bandwidth performance

```
Outspurced Solution
> napster_traffic_add_costs:=evalf(napster_traffic_add*ftp_monthly_traffic_add);
> napster_server_hosting_costs:=evalf(napster_server_number*monthly_payment);
> napster_server_installation_costs:=
   evalf(napster_server_number*installation_costs_hosting);
> napster_server_installation_costs_monthly:=
   evalf(napster_server_installation_costs/(storage_depreciation*12));

Inhouse Solution
> napster_traffic_inhouse_add_costs:=
   evalf(napster_traffic_inhouse_add*inhouse_monthly_line_lease);
Costs for additional traffic in [CHF] per month
> napster_inhouse_server_costs:=
   evalf((napster_server_inhouse_number*inhouse_costs)/(storage_depreciation*12));

Client Storage Costs
> napster_storage_costs:=evalf(((harddisk_price)/(storage_depreciation*12)));
```

Figure 66: Napster inhouse and outsourced costs

```
Client's perspective

Total Costs per month and peer for the next 4 years
> napster_client_total_costs:=
   evalf(napster_client_bandwidth_total_costs+napster_storage_costs);
Total Costs per month and peer after 4 years (without investment)
> napster_client_total_costs_plus:=evalf(napster_client_bandwidth_total_costs);


Server perspective

Outsourced: Total Costs per month for the next 4 years
> napster_server_total_costs:=
   evalf(napster_traffic_add_costs+napster_server_hosting_costs+
   napster_server_installation_costs_monthly);

Outsourced: Total Costs per month after 4 years (without investment)
> napster_server_total_costs_plus:=
   evalf(napster_traffic_add_costs+napster_server_hosting_costs);

Inhouse: Total Costs per month for the next 4 years
> napster_inhouse_server_total_costs:=
   evalf(napster_traffic_inhouse_add_costs+napster_inhouse_server_costs+
   inhouse_monthly_line_lease);

Inhouse: Total Costs per month after 4 years (without investment)
> napster_inhouse_server_total_costs_plus:=
   evalf(napster_traffic_inhouse_add_costs+inhouse_monthly_line_lease);

ISP's perspective

Total Costs for the ISP per month excluding storage costs
> napster_isp_total_cost:=evalf(napster_client_total_costs_plus*peer_network_size);

Overall perspective

Outsourced: Total Costs per month for the next 4 years
> napster_overall_costs:=
   evalf((napster_client_total_costs*no)+napster_server_total_costs);
Outsourced: Total Costs per month after 4 years (without investment)
> napster_overall_costs_plus:=evalf((napster_client_total_costs_plus*no)+
   napster_server_total_costs_plus);

Inhouse: Total Costs per month for the next 4 years
> napster_inhouse_overall_costs:=evalf((napster_client_total_costs*no)+
   napster_inhouse_server_total_costs);

Inhouse: Total Costs per month after 4 years (without investment)
> napster_inhouse_overall_costs_plus:=evalf((napster_client_total_costs_plus*no)+
   napster_inhouse_server_total_costs_plus);
```

Figure 67: Napster total costs

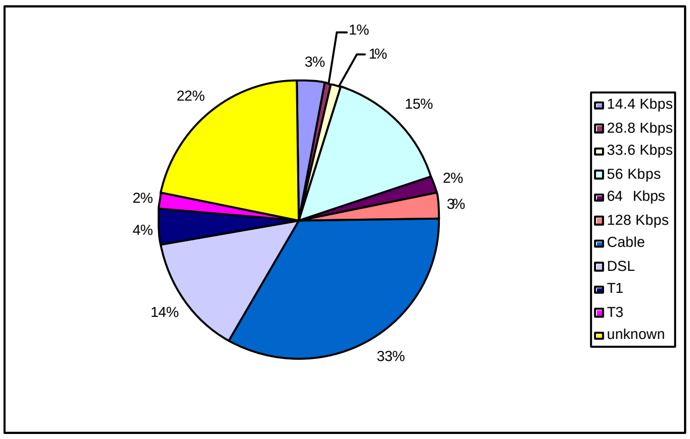


Figure 68: Reported bandwidth for Napster clients

7.2 Stakeholder Analysis

7.2.1 Peer/Client Cost Perspective without Investment

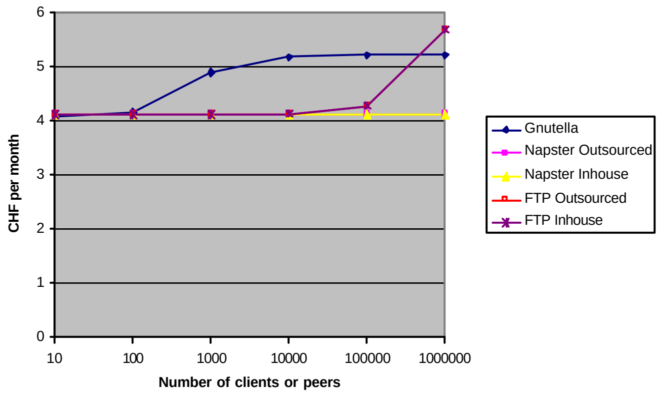


Figure 69: Scenario 2: Client/Peer cost perspective > 4 years

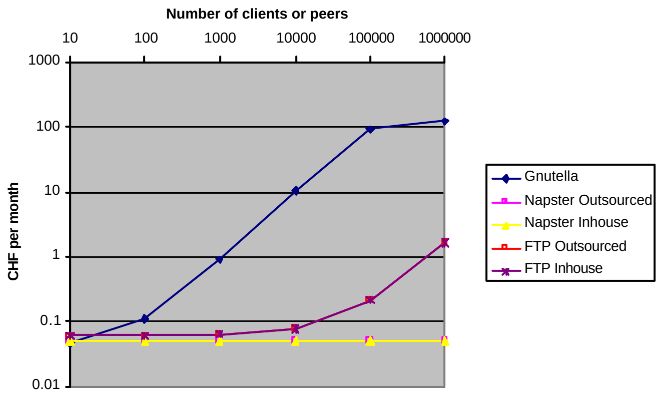


Figure 70: Scenario 3: Client/Peer cost perspective > 4 years

7.2.2 Peer/Client Load Performance

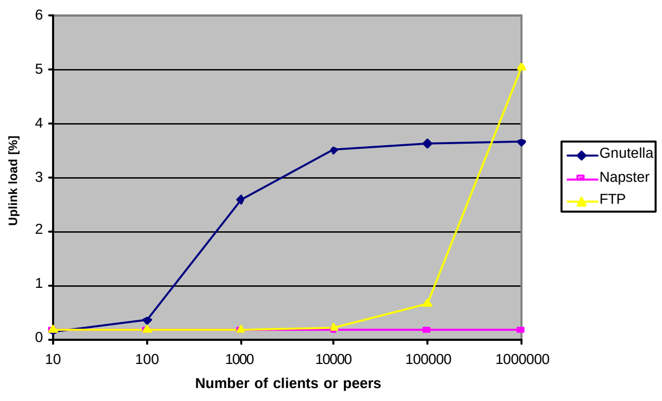


Figure 71: Scenario 1: Uplink load performance

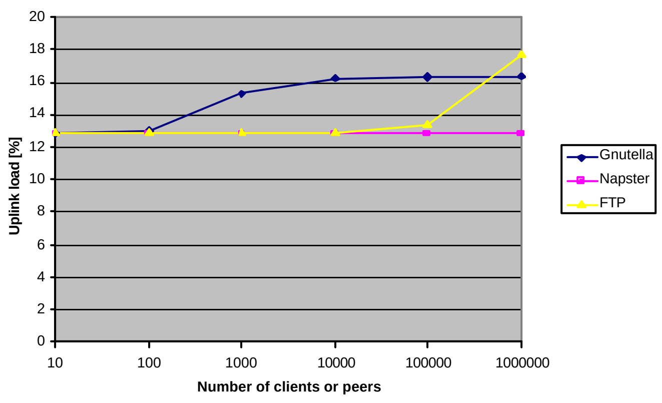


Figure 72: Scenario 2: Uplink load performance

7.2.3 Server Cost Perspective without Investment

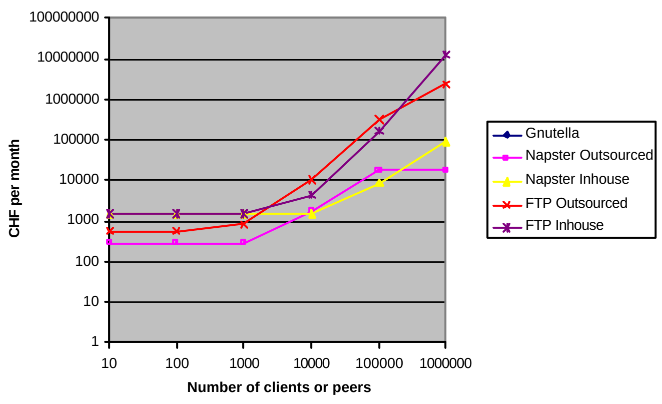


Figure 73: Scenario 1: Server cost perspective > 4 years

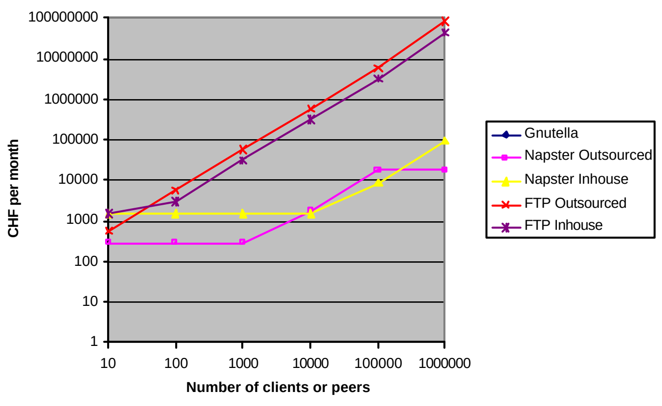


Figure 74: Scenario 2: Server cost perspective > 4 years

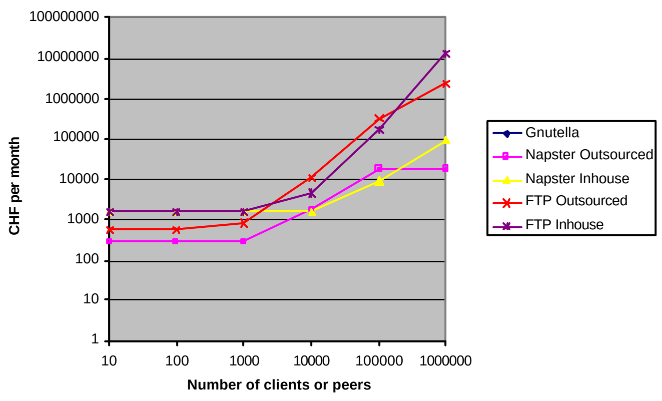


Figure 75: Scenario 3: Server cost perspective > 4 years

7.2.4 ISP Cost Perspective

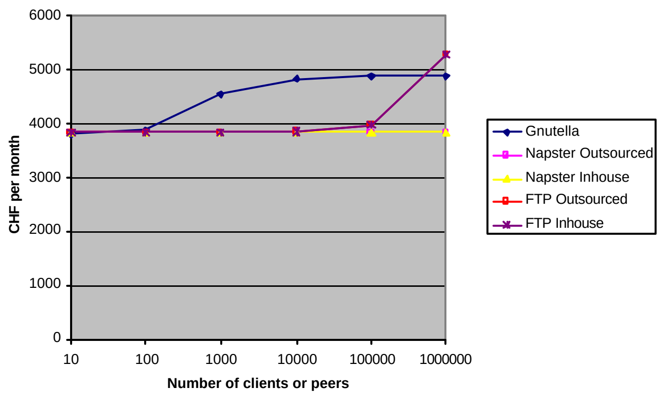


Figure 76: Scenario 2: ISP cost perspective

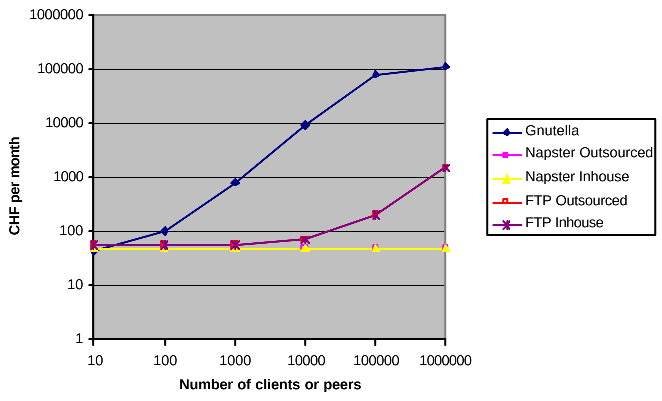


Figure 77: Scenario 3: ISP cost perspective

7.2.5 Overall Cost Perspective without Investment

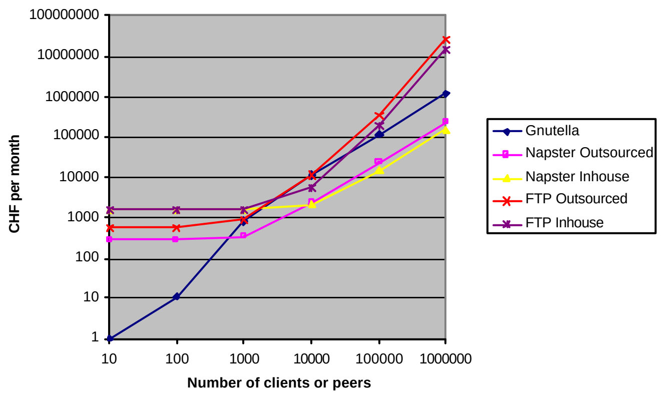


Figure 78: Scenario 1: Overall cost perspective > 4 years

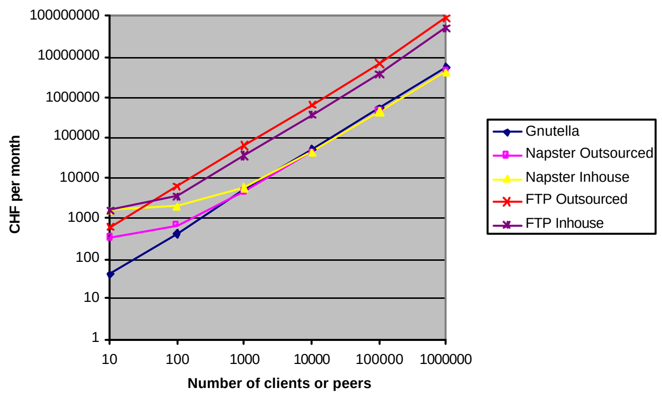


Figure 79: Scenario 2: Overall cost perspective > 4 years

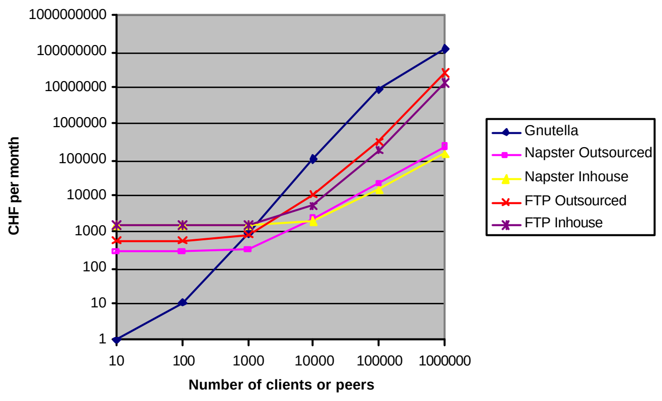


Figure 80: Scenario 3: Overall cost perspective > 4 years

7.3 Application Parameters

Gnutella					
No.	**Variable name**	**Value**	**Unit**	**Explanation**	**Source**
33	pingrate_eval	15	[minutes]	Ping creation period of a peer in minutes	Assumption
34	pingrate	derived	[1/minute]	Average ping creations per minute and peer	see pingrate_eval
35	query_proportion	12	[%]	Percentage of queryhit descriptors compared to queries	www.cs.ucr.edu/~csyiazti/courses/cs204/project/html/final.html
36	n_c	3.4	[]	Average node connectivity	people.cs.uchicago.edu/~matei/PAPERS/ic.pdf
37	TTL	7	[]	Time to live	people.cs.uchicago.edu/~matei/PAPERS/ic.pdf

Table 5: Gnutella variables and constants

FTP					
No.	**Variable name**	**Value**	**Unit**	**Explanation**	**Source**
38	ftp_client_login_avg	3	[]	Total number of client logins per day	Assumption
39	ftp_chars_per_file	derived	[chars]	Chars per file	see avg_chars_per_file
40	ftp_number_folder_1	6	[folders]	Number of folders within this folder level	Assumption
41	ftp_number_folder_2	26	[folders]	Number of folders within this folder level	Assumption
42	ftp_number_folder_3	100	[folders]	Number of folders within this folder level	Assumption
43	ftp_char_folder_1	7	[chars]	Number of chars for ftp_number_folder_1 folders	Assumption
44	ftp_char_folder_2	1	[chars]	Number of chars for ftp_number_folder_2 folders	Assumption
45	ftp_char_folder_3	15	[chars]	Number of chars for ftp_number_folder_3 folders	Assumption
46	ftp_number_files	derived	[files]	Number of files stored on the server	see avg_unique_files_in_system

Table 6: FTP variables and constants

Napster					
No.	**Variable name**	**Value**	**Unit**	**Explanation**	**Source**
47	napster_max_search_result	20	[]	Is the maximum number of search replies from the server	Assumption
48	napster_server_number	1	[]	Number of central servers	Assumption
49	napster_client_login_avg	derived	[]	Number of logins per day	see ftp_client_login_avg

Table 7: Napster variables and constants

8 References

[1] Limewire: The Gnutella Protocol Specification vO.4; http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf (in March 2003).

[2] A. Oram (ed.): Peer-To-Peer: Harnessing the Power of Disruptive Technologies; O'Reilly&Associates, Sebastopol, U.S.A., 2001.

[3] A. Michalove: Amdahls Law; http://home.wlu.edu/~whaleyt/classes/parallel/topics/amdahl.html (in March 2003).

[4] S. Saroiu, P. Gummadi, S. Gribble: A Measurement Study of Peer-to peer File Sharing Systems; Technical Report # UW CSE-01-06-02, Department of Computer Science & Engineering, University of Washington, Seattle, U.S.A., 2002.

[5] C. Shirky: What Is P2P… And What Isn't?; http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html (in November 2000).

[6] R. Schollmeier: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications; Proceedings of the First International Conference on Peer-to-Peer Computing (P2P.01).

[7] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Lab, 2002.

[8] Webopedia: Client/server architecture; http://www.webopedia.com/TERM/C/client_server_architecture.html (in February 2003).

[9] Peer-to-Peer working group: What is peer-to-peer?; http://www.peer-to-peerwg.org/whatis/index.html (in February 2003).

[10] ETHOS: Client Servers; http://www.ethoseurope.org/ethos/Techterm.nsf/All/CLIENT+SERVERS (in February 2003)

[11] Peer-to-Peer working group: Glossary for Peer-to-Peer; http://www.peer-to-peerwg.org/tech/glossary.html (in February 2003).

[12] A. Oram: Peer-to-Peer makes the Internet interesting again; http://linux.oreillynet.com/pub/a/linux/2000/09/22/p2psummit.html (in February 2003).

[13] M. Ripeanu, A. Iamnitchi, I. Foster: Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal 6, 1 (2002), 50-57.

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer to peer networks. In ICS'02, New York, USA, June 2002.

[15] M. Findeli: Peer-to-Peer (P2P) Networking; Lehrstuhl für Kommunikationsnetze an der Technischen Universität München, München, Deutschland, 2001; http://www.onlinejunkie.de/docs/p2p.pdf (in March 2003).

[16] D. Zeinalipour, Y.T. Folias: A Quantitative Analysis of the Gnutella Network Traffic; http://www.cs.ucr.edu/~csyiazti/courses/cs204/project/html/final.html (in February 2003).

[17] Limewire Homepage: http://www.limewire.com/index.jsp/size (in March 2003).

[18] Dr. Scholl: Napster protocol specification, April 7, 2000; http://opennap.sourceforge.net/napster.txt (in March 2003).

[19] J. Postel, J. Reynolds: File Transfer Protocol (FTP), RFC 959, October 1985.

[20] CenterSpan: Mediated Peer-to-Peer (P2P) Enables Viable, Cost Effective Delivery of Digital Entertainment; http://www.centerspan.com/technology/cscc_p2pwhitepaper.pdf (in March 2003).

[21] B. Yang and H. Garcia-Molina: Improving Search in Peer-to-Peer Networks; 22nd International Conference on Distributed Computing Systems (ICDCS'02), July 2002.

[22] K. Sripanidkulchai: The popularity of Gnutella queries and its implications on scalability, http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html (in March 2003).

[23] B. Yang, H. Garcia-Molina: Comparing Hybrid Peer-to-Peer Systems, Proceedings of the 27th Intl. Conf. on Very Large Data Bases.

[24] E. Criscuolo: Performance Efficiency of Internet Protocol (IP) in Space Applications; Computer Science Corp. for the Goddard Soace Flight Center; http://ipinspace.gsfc.nasa.gov/flatsat/docs/IP-Performance.doc (in March 2003).

[25] Sans Institute: TCP/ IP and tcpdump Flyer; http://www.sans.org/resources/tcpip.pdf (in March 2003).

[26] R. Schollmeier: Why Peer-to-Peer (P2P) does scale: An analysis of P2P traffic patterns; Proceedings. Second International Conference on Peer-to-Peer Computing (P2P’02), 2002.

9 List of Figures

10 List of Tables