Matthieu Weber

Advertising Peer-to-Peer Networks over the Internet

29th June 2006

Licentiate's Thesis

UNIVERSITY OF JYVÄSKYLÄ 2006

ABSTRACT

Weber, Matthieu
Advertising Peer-to-Peer Networks over the Internet
Jyväskylä: University of Jyväskylä, 2006, 120 p.
(Jyväskylä Licentiate Theses in Computing
ISSN 1795-9713; 7)
ISBN 951-39-2329-0
Finnish summary

Most peer-to-peer networks nowadays are decentralized or even fully distributed, meaning that they do not require a central authority for proper operation. Joining such networks is however often performed by using a central directory of its members, thus breaking their decentralized characteristic. This work proposes an advertisement system for peer-to-peer networks that does not rely on a central service, nor requires any dedicated infrastructure to be setup, using only already existing, fully distributed messaging networks, such as IRC or Usenet. The efficiency of the new system is discussed, as well as its impact on the networks it relies on.

Keywords: P2P, advertisements, IRC, Usenet, fully distributed

Author

Matthieu Weber
Department of Mathematical Information Technology
University of Jyväskylä, Finland
P.O. Box 35 (Agora), FI-40014 University of Jyväskylä
mweber@mit.jyu.fi

Supervisor

Jarkko Vuori
Department of Mathematical Information Technology
University of Jyväskylä, Finland

Examiners

Rolland Vida
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics, Hungary

Jarmo Siltanen
School of Information Technology
Jyväskylä University of Applied Sciences, Finland

ACKNOWLEDGEMENTS

I would like to thank my supervisor Jarkko Vuori for his help. I would also like to thank Mikko Vapa for his useful comments and for the discussion during which the original idea for the thesis came up. Finally, I would like to thank my wife Ritva for proofreading the text of the thesis and translating the summary into Finnish.

Jyväskylä, 29th June 2006
Matthieu Weber

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

Peer-to-peer networking has become extremely popular in the past few years, in the information technology community as well as in the general public. [Sch01] defines the term *peer-to-peer* as follows:

> *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) and requesters (Servent-concept).*

Current peer-to-peer systems are mostly (but not only) file sharing systems, i.e., distributed software that allows users to download files stored on other users' computers. Peer-to-peer systems are self-organized into networks, and any user running the required software can join (and leave) those networks. Prior to joining a network, it must however be located, meaning that it must somehow be advertised. Most peer-to-peer systems are provided with their own solutions to advertise their network, but these are specific to each network and, more importantly, they are centralized, thus breaking all the benefits brought by the decentralized architecture of peer-to-peer systems. The goal of this thesis is to design a generic, decentralized system for advertising peer-to-peer networks to the users that want to join it.

1.1 Peer-to-Peer Networks

1.1.1 Historical Background

Until the late 1990s, sharing files over the Internet was done using client/server systems. [Sch01] defines *client/server* as follows:

> *A Client/Server network is a distributed network which consist of one higher performance system, the Server, and several mostly low 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.*

The best-known examples of client/server systems are FTP and the WWW, where servers hold the files, and the users connect to these servers with the appropriate clients to retrieve them (and sometimes upload new files). Finding the right file was not an exact science, and users had to rely on WWW and FTP search engines, or even the knowledge of other users to locate the proper server. And when the content of the files was copyrighted, it was a good idea not to openly distribute the address of the server, in order not to attract too much attention.

Sharing a file in practice with a client/server is done as follows: the sender of the file to be shared stores it onto a file server, and the recipient downloads it from that server. Many different types of servers could be used for that purpose, such as BBS (which existed before the Internet and were accessed using a modem), systems based on IBM's SMB protocol [Sha02], such as *LAN Manager* or *Samba* [sam05], NFS [Sa03], and WWW or FTP servers connected to the Internet. To some extent, one can also put e-mail into this category, where the file is temporarily stored in the recipient's mailbox on a server, and Usenet News [Sal99], where the files are replicated over multiple servers for a limited amount of time. All the above methods (except e-mail) allow to share files in a one-to-many fashion, i.e., that the file can be retrieved by more than one recipient. E-mail is essentially a one-to-one type of communication, but it is however possible to send several copies of the file to several recipients at once.

Client/server was the main way to share files between two users, but not the only one. A real P2P approach existed before the words "peer-to-peer" became notorious, using IRC [OR93]: most IRC clients allow client-to-client connections via a set of so called *DCC* (Direct Client-to-Client) [ZRM94] commands, and the "DCC send" and "DCC get" commands are designed for respectively sending to and receiving from a specified user. File sharing with DCC is strictly a one-to-one type of communication.

As will be shown later, P2P is not only about file sharing, even if it is probably the first use coming to mind when P2P is mentioned: person-to-person messaging is also an area in which P2P concepts have largely been used. For illustration purposes, here is a (non-exhaustive) list of non-P2P messaging systems:

- E-mail, which is clearly a client-server system, where the message is considered as a file and temporarily stored on a server, as explained above.

- Usenet, which resembles e-mail and uses the same approach.

- IRC, where the user connects, using a specific client, to a network of servers (which can be, in some cases, reduced to one single server).

- Jabber [Jab05], which is a protocol for instant messaging based on the principle of e-mail: all messages transit through servers, but if the recipient is online when a message is sent, it is pushed toward it instead of being stored on the server until the message is retrieved from there.

The common denominator behind all these systems (excepting DCC) is the client/server architecture, which creates a hierarchy between the computers connected to the Internet: servers are large and expensive machines, with a lot of storage space and a lot of processing power, clients are small and not-too-expensive computers. With the rise of the personal computer, the power of which has never stopped increasing year after year, and especially since the spreading of affordable broadband home-Internet connections such as ADSL or Cable-TV networks that allow anybody's computer to be reachable from the Internet without interruption, the distinction between client and server disappears slowly. The logical distinction still exists ("the client requests, the server replies"), but the discrimination based on the power of the machines, their availability and their network connection has no more reason to be. Nowadays, every home user's PC is powerful enough to be a (reasonably sized) server. Even though the power of server computers increases at the same rate as the power of personal computers, the latter have in the past few years reached the minimum performance level required to make suitable file servers.

The notion of P2P communication arises from the above statement: if any computer can become a server, then any computer can be at the same time client and server. Based on this idea, Napster [Ora01] was released in fall of 1999. The novelty in the Napster system was double:

- It made the search for the proper server much easier, by providing one single directory of all the users and all the files shared by these users.

- It bypassed the need for a file server by allowing each user to share his or her own files, turning the user's workstation into a computer that is at the same time a client (for retrieving files from other Napster users) and a server (for sharing one's own files with other Napster users). Instead of being uploaded to the server and then downloaded, the files are copied from user computer to user computer in a P2P fashion.

Since then, the proportion of P2P traffic on the Internet has rapidly and constantly increased [LRW03, SW02].

1.1.2 The P2P Revolution

Besides giving a formal definition of P2P, [Sch01] also defines the term of *servent*, which is the contraction of "server" and "client". And this is exactly what a node in a peer-to-peer network is: depending on the circumstances, it plays the role of server or client (or both at the same time).

Although the concept of P2P has become well-known with the advent of Napster, it is already quite old (and did not bear the name of P2P at that time). The following systems are all at least several years older than Napster, and yet, they are peer-to-peer systems:

- the Unix *talk* [man] command, which allows two users to establish a TCP connection between each other using a specific client and exchange text messages;
- IRC's "DCC chat" command, that allows a peer-to-peer dialog; the messages exchanged by the users are therefore not sent through the IRC server, as it is the case for normal message exchange;
- the above mentioned DCC file sharing features of IRC clients;
- the networks formed respectively by the Usenet, IRC and e-mail servers use P2P communication between their nodes;
- IP networking is also a P2P system, in the sense that, in an IP network, all the nodes are equal and exchange IP packets in a P2P fashion. The Internet, at network level, is actually a P2P network by design.

ICQ [Rei01], released in 1996, has revolutionized the instant messaging the same way Napster did in the field of file sharing. Here are the common features of both systems:

- the presence of a global index of all the users (and for Napster, of all the files shared by the users as well);
- the possibility for the users to query these indexes based on various meta-information (keywords in the file name, type of file . . . in Napster; name, nickname, location, addresses, phone numbers, hobbies . . . in ICQ);
- the possibility in ICQ to deliver a message to the recipient with only little manipulation on his side, or the possibility in Napster to download a file from someone's computer with no need for that computer's user to do any manipulation whatsoever (simplicity of use).

These features were available for the first time all at once within one single tool and they made these tools extremely popular. It is important to notice that in Napster and ICQ, although the exchange of information (message or file) is done in a P2P manner, the location of meta-information (the source of the file or the destination of the message) is purely client-server. Other systems exist however,

where the location of the meta-information is also done in a P2P fashion (see below).

The concepts that made the success of Napster and ICQ were reused, with different kinds of implementations, in various software which attempted to clone or improve the operations of the "original" tools.

Modern instant messaging systems such as AOL Instant Messenger (from version 2.0) [aim04], Yahoo! Messenger [yah04], MSN Messenger [msn04]... allow peer-to-peer connections for person-to-person dialog. They must however be considered as clones of ICQ, since they use the same concepts.

Post-Napster file sharing systems, however, are, at least for some of them, trying to improve the way Napster works: Gnutella [Ora01], Freenet [fre] and Overnet [ove02] (among many others) aim toward the suppression of the need for an index (each node of the network actually stores a partial index), whereas Kazaa [Sha03] and eDonkey attempt to combine the efficiency of an index with the reliability of the absence of index, by using a distributed and partially replicated index.

1.1.3 A Matter of Perspective

It was mentioned above that personal computers, which were until recently restricted to the role of clients in the client/server architecture, have become powerful enough to play the role of the servers. Besides allowing the creation of modern P2P systems, it has also the side effect of blurring the limit between P2P and non-P2P systems. Since any computer running a Web server, an FTP server, Samba or NFS can share resources, a network composed of computers which are all running these systems and which are actually sharing some of their resources can be considered as a P2P network, according to the definition.

In some cases, the P2P-characteristic appears only at a given level of description of the system. For example, e-mail, Usenet and IRC have a client/server architecture from the point of view of the user (the user uses a client software and connects to a server), but these three systems use internally a P2P architecture: the system relies on a network of servers, which communicate with each other in a P2P fashion.

1.2 Overlay Networks

From an abstract point of view, the nodes of the P2P network form an *overlay network* lying on the top of the Internet. As a classical, concrete network, the Internet is composed of three categories of components:

- endpoint equipments, i.e., the workstations, servers, etc. that allow the users to produce data which is sent or retrieved over the Internet,
- routing equipments, i.e., the routers, switches, proxies, wireless base stations, etc. that allow the data to travel across the network,

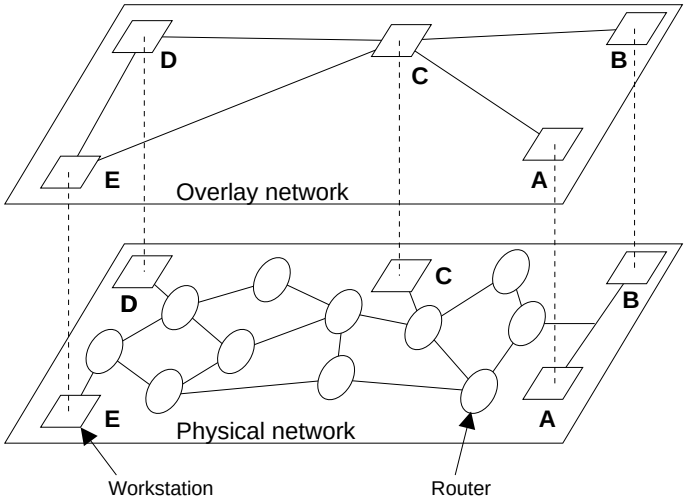


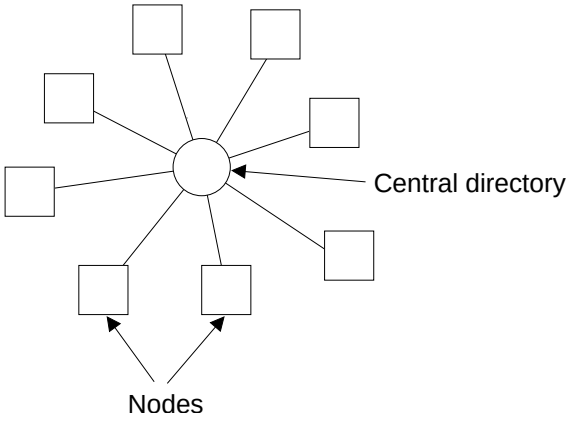
FIGURE 1 Example of an overlay network and its underlying physical network

- links, i.e., copper wires, optic fiber and radio signals that connect the various equipments.

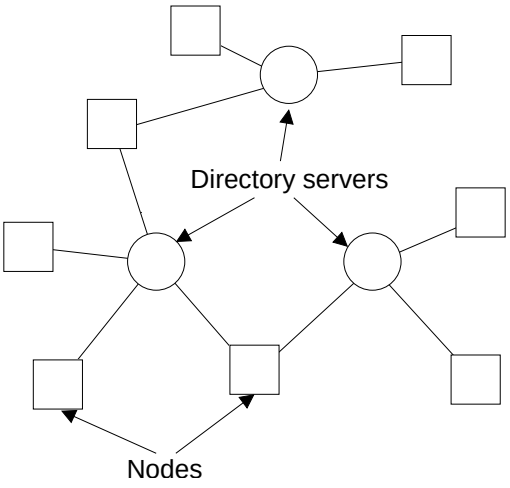
These components all have a physical existence; the network they are part of is therefore called the *physical network*.

In a typical P2P network, the only physical components are the workstations. They play at the same time the role of endpoint equipments and routing equipments, whereas the links are virtual, and are most of the time TCP (sometimes UDP) sockets. A socket does not physically connect two computers like a wire or even a radio signal does, but rather represents, from the point of view of the operating systems on both computers, a connection between them; the physical location on the network of both ends of a socket is thus hidden from the applications using them. This is illustrated in Figure 1: the workstations A and B are on the same local, physical network, but they are not connected straight to each other in the overlay network; if A and B want to exchange messages using the overlay, these messages must go through the workstation C, which plays in this case the role of a router between A and B within the overlay network. On the opposite, workstations D and E have a direct connection in the overlay network, but the map of the underlying physical network clearly shows that messages sent from D to E will need to go through two routers in order to reach their destination.

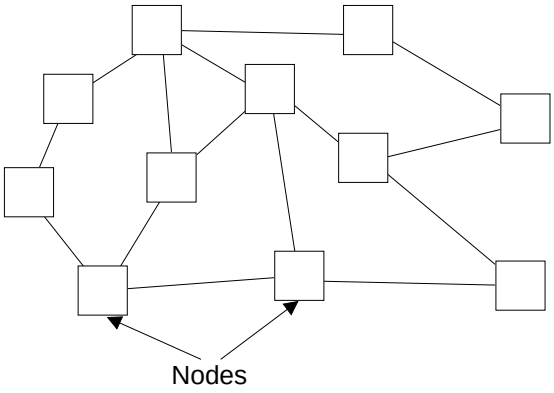
The P2P network is thus composed only of workstations (called *nodes*) which have established TCP *connections* between each other. The P2P network is not able to transport data in itself; it relies for this purpose on the underlying physical network that has been described above. The P2P network can therefore be described as a *virtual network* and as an overlay network of the Internet. However, most of the P2P systems described above actually are composed of two distinct overlay networks: one is used for signaling, which is, in most cases, resource location, and the other for data transport. Both can go from simple star-shaped networks to complex meshes and they do not necessarily have the



(a) Star-shaped network



(b) Mesh network without routing



(c) Mesh network with routing

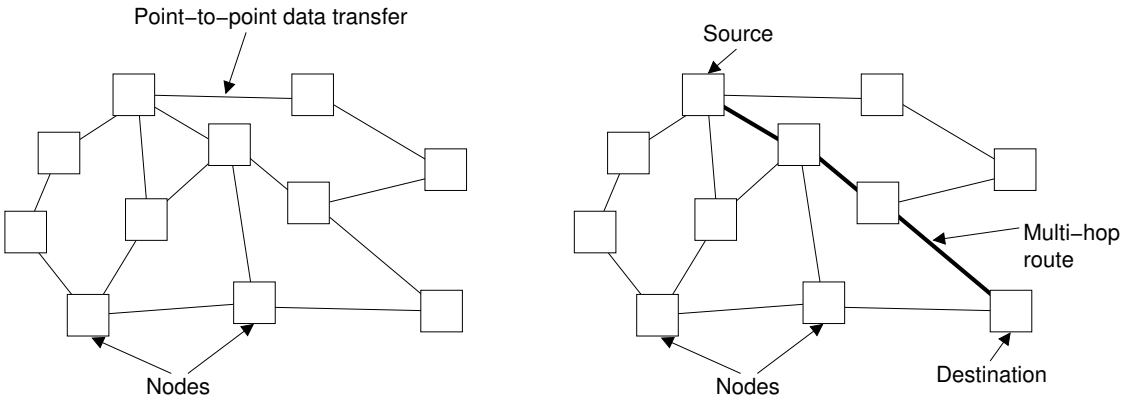
FIGURE 2 Various topologies of overlay networks for resource location

same structure. Here are examples of overlay networks for resource location:

star-shaped network, where all the nodes connect to one central directory and query it (see Figure 2(a)); this is clearly not a P2P structure. Napster, ICQ and its clones use it.

mesh network without routing, where nodes are connected to at least one directory server (see Figure 2(b)); this is not a P2P structure. There is no need for routing here, since the server is always only one hop away from the requester node. eDonkey uses this structure, as well as, to some extent, the network formed by e-mail or Jabber servers: the directory servers are in the latter case the DNS servers.

mesh network with routing, where the nodes are connected to each other (see Figure 2(c)); queries for resources are routed through the network until one



(a) Mesh network without routing

(b) Mesh network with routing

FIGURE 3 Various topologies of overlay networks for data transport

node can reply to it. This structure is clearly P2P. Gnutella, Freenet and Kazaa use this structure.

Here are examples of overlay networks used for transporting data (file transfer in the case of file-sharing networks, or message transfer in the case of instant messaging applications):

mesh network without routing, where the recipient of the data connects directly to the sender of the data (see Figure 3(a)); this makes it a P2P structure. The communication is essentially point-to-point, but since each node can be at the same time sender and recipient for multiple data transfers, the resulting network is a mesh. Napster, Gnutella, Kazaa, ICQ and its clones use this structure.

mesh network with routing, like the previous one, but the same data can be transferred across several nodes before reaching its destination (see Figure 3(b)); this structure is used in Freenet. The topology of the transport networks used in eDonkey and BitTorrent can be assimilated to "mesh with routing" since one can start to download a file from a node which is itself in the process of downloading the same file from another node; the data is actually routed from one node to the other across the network, in a multicast manner, each upstream node in the multicast tree acting as a cache for its downstream nodes. The networks formed by IRC, Usenet, Jabber and e-mail servers have also "mesh with routing" structures.

1.3 Research Problem

When a user wants a workstation to join a given P2P network which does not have any known-beforehand, central directory, the user must find at least one

other computer which is already part of that network and establish a TCP connection with it. In short, one needs to find connection parameters (i.e., an IP number and a TCP (or UDP) port number to which to connect).

Because currently working P2P systems provide mechanisms allowing new nodes to join the network, most researchers concentrate on the problem of locating and retrieving resources from the network, and most, if not all, of them consider that the acting nodes are already part of the network.

Some systems, like Napster or ICQ are using central indexes (sometimes called "databases" or "directories") referencing all current users of the system (along with their connection parameters) in order to allow new users to locate other peers and thus communicate with them. The mean of accessing the index (usually the same kind of connection parameters as mentioned above) is known before using the system (e.g., from a configuration file distributed with the software), and thus represents an easy way to join the network. Recent history [Ano02b] has proved that this central index is a potential point of failure, and that bringing down the index will in time bring down the entire system: current users are still members of the network, but they can't rejoin it after leaving, and no new user can join the network. In time, the network will disappear. Backup servers are not always a solution in this case: if a company is running the servers (like e.g., Napster) and is stopped by a decision of justice, the backups are usually stopped at the same time. Similarly, if only a limited set of backups exist, it is possible to stop the Internet traffic to and from those servers to censor the whole system (*denial of service* attacks).

In order to avoid this central point of failure, fully distributed P2P systems like Gnutella have been developed. However, when a P2P system is designed in such a way that there is no central control over the network, users must find a way to join that network prior to being able to use the system. But finding an entry point into the network becomes a problem when one doesn't know where to start looking for it.

This thesis proposes a solution for this problem, using existing decentralized systems such as IRC or Usenet to efficiently distribute information about existing nodes of a P2P network.

1.4 Structure of the Thesis

In this first chapter, we have introduced the history of peer-to-peer networks, from which an informal definition is derived, then defined overlay networks and finally defined the problem of finding an entry point into the P2P network one wants to join, which is a necessary step before actually joining that network.

The second chapter presents solutions for finding the entry point, based on examples from everyday life, then proposes one possible protocol that formalizes the exchanges between computers in order to implement these solutions.

The third chapter defines a formal terminology for some of the concepts which will be used in the rest of the thesis.

The fourth chapter describes the simulator that has been developed in order to study the behavior of the protocol described in the second chapter, and the fifth chapter describes in more details the rules that define the behavior of individual nodes within the simulated network.

The sixth chapter presents the results of the simulations and the analysis of these results, and the seventh chapter concludes the thesis.

2 FINDING THE P2P NETWORK

This chapter presents solutions to the problem of finding a way to join a P2P network, as well as the software design of one possible solution. The protocol described at the end of this chapter has not been implemented as is in the simulator (see Chapter 4), but the simulator rather implements the behavior of virtual nodes that would implement the aforementioned protocol. Moreover, the simulator focuses only on the variants of the protocol that uses IRC and Usenet as a communication layer (see below for details).

2.1 The Real-World Example

A hint of a solution to the problem described in Section 1.3 can be found in the real world. When one person wants to know about a given service, there are several ways to find information about that service, using well-known sources of information, i.e., sources that are general knowledge. Each of those ways can be transposed in the world of P2P systems:

- Ask friends if they know something about the service: this is equivalent to already being member of a P2P network.

- Look into the phone book, and especially the yellow pages, into the local newspaper or into specialized magazines for advertisements: this can be compared to using an already existing, well-known system, like a search engine (e.g., Google [Goo03a]), Usenet News or IRC.

- Knock randomly on people's doors, until one find the required information: this can be assimilated to random network scan.

The first example is not possible in a P2P network, since the user is not yet a member of any P2P network, but the two other examples are worth being investigated.

Finding a source of information is one more step that must be taken in order to find the desired service, and which only displaces the problem. However, the fact that the source of information is *well-known*, i.e., part of the *general knowledge*, makes this step possible for anybody.

2.2 Existing Systems

Existing systems already try to address the problem of entering the network. Several solutions have been found, that can be divided into three categories, depending on the way their resource index is organized [AW03]:

- Systems with centralized indexes, where a list of potential entry points are available at one source only. The address of the index is known by the user or hard-coded into the software by the software publisher. The latter is often true with commercial systems.
- Systems with decentralized indexes, where the lists of entry points are scattered over several sources, but where the location of those sources is to be found in well-known places. The address of the sources is known by the user, either directly or as e.g., the result of a query sent to a Web search engine, using pertinent keywords.
- Systems with no index, where the lists of entry points are directly available in well-known places.

2.2.1 Centralized Indexes

The first P2P systems to be released were (or still are) using centralized indexes. A centralized index has several advantages:

- simple and straightforward to design,
- easy to locate: only one address is necessary, which can be hard-coded into the software which is going to use that index,
- easy to control: all the data is in one place, there is no risk of inconsistencies like there can be in distributed or replicated indexes.

The centralized index concept also has drawbacks:

- if the system grows too large, the available processing power and/or network bandwidth available for the users to use the index might not be enough,

- as mentioned before, a centralized index is a favorite target for attacks aimed at bringing the whole system down. These attacks can be technical, such as Denial of Service (DoS) attacks, where a large number of computers, controlled by the attacker, attempt to saturate the server hosting the index with incomplete connection requests, in order to prevent it to reply to normal queries, or legal attacks, aimed at forcing the entity running the index to cease its activity, thus stopping the index to work.

Napster, Kazaa and most Instant Messaging services, such as ICQ, Yahoo! Messenger and MSN Messenger all obviously belong to this category: their centralized index are maintained by the companies developing the software, and the address of the index is hard-coded into the source code of the software. In here, the well-known source of information is the company itself, since the information about the index is delivered to the user at the same time as the user acquires the software. The fact that the address of the index is hard-coded into the software also participates to the business model of the companies: it makes the software easy to use, therefore attractive to the end-user, and it keeps the user captive since there is no way to use an alternative index.

eDonkey is another commercial software, which differs from the previous ones in using a decentralized index (i.e., several independent indexes, called *servers*, run by independent entities), but distributes a list of of servers along with their software. In this case like in the above ones, the company providing the software is the well-known source of information. The user of eDonkey has, however, and as opposed to the above commercial systems, the possibility to add more servers to its list of known server, and to completely disregard the list provided by the company. Besides, the servers know each other's existence, and can provide the software with an up-to-date list of active servers. Moreover, several clones of the eDonkey client, implementing the same protocol, have been developed (such as eMule [emu04] and MLdonkey [mld04]), which allow users to join the eDonkey network without any need for the original eDonkey software publisher.

Gnutella at some point of its existence also fell into the current category, since the early Gnutella cache [Ora01, p. 113–115] was a centralized index; the address of the index's host server was mentioned in various documents available on the Web (whose search engines were the well-known sources of information). Even though the index was here centralized, the way to get to know the index was not centralized.

2.2.2 Decentralized Indexes

More recent configurations of Gnutella [Däm03] belong to the second category: there are several independent lists of peers available on the World Wide Web (so if one list disappears, there are still several others), but finding those lists depends on a Web search engine. There are however several major and independent search engines available (so this is not a potential point of failure either) representing the well-known sources of information. The risk of global failure is smaller than in the previous case thanks to this redundancy. However, the use of those lists still requires human intervention, because of the need for manipulating a Web search engine (Section 2.3.3 discusses the automatization of that task).

2.2.3 Without an Index

JXTA [Gon01], the Universal Ring [CDKR02], GIrcCache [Ano02a] are in the third category: the needed information is available from numerous well-known sources (RendezVous servers [Gon01] of JXTA, each node of the Universal Ring, the local IRC server in GIrcCache). Once one source is known, the other ones can easily be found through the first one. As with decentralized indexes, the risk of failure is low, thanks to the high redundancy of the well-known sources of information.

JXTA is however only dodging the issue: all JXTA users are members of one peer group called the World Group, and joining this group in order to know other RendezVous servers requires to know a first RendezVous server, hosted by Sun Microsystems, which is hard-coded in the software distribution of JXTA. Because of that, as long as JXTA is not widely used (i.e., the address of the closest RendezVous server is a well-known piece of information for everybody), it will actually fall into the category of centralized systems. In the same way, the Universal Ring expects the system to be widely distributed, and that each user uses a well-known node as entry point.

GIrcCache[1], on the contrary, is fully distributed: it relies on IRC, which is a 10 years old, well-established and fully distributed chat system. Accessing GIrcCache and finding Gnutella nodes only requires to know the local IRC server, which is assumed to be a well-known piece of information (as are the Internet addresses of the local mail server and Domain Name server). GIrcCache, however, allows finding only Gnutella nodes.

2.3 Toward a Universal Solution

The existing solutions presented above are designed for being used within a specific system (except the Universal Ring), and often assume that the system is already sufficiently widely used in order for the source of information to be

[1] GIrcCache is not documented, but is implemented in Gnucleus [Ano02a]

well known. The latter is however not true, and, although the Universal Ring is one possible solution for the future, none of those systems is nowadays widely spread.

A transitional solution needs to be set up so that one (or maybe several) system can in future become as common as the e-mail, the Web, Usenet or IRC are nowadays. This transitional system should be widely available as soon as possible, without the need of heavy infrastructure deployment, in order to make it attractive to the peer-to-peer software developers. One easy way to achieve this is to rely on already existing infrastructures, which are already part of the general knowledge of the Internet. Usenet, IRC, and, to some extent, the Web can be used that way.

2.3.1 Usenet

Usenet, sometimes called Internet News, is a distributed, completely decentralized system that broadcasts messages across a large set of servers. As Usenet is as old as the Internet itself, it is considered as one of the basic services of the Internet, and should be available from any Internet Service Provider. However, given the absence of central administration in Usenet, getting an accurate picture of how many Usenet servers are available in the world is difficult. Some Web sites propose lists of public servers [Ano03a, Ano03b], but there is no way to infer from those data the availability of Usenet servers for everybody. It is however assumed that most of the people having an access to the Internet are also given an access to a Usenet server by their Internet Service Provider.

Messages in Usenet are sorted by topic into newsgroups, usually one newsgroup for one topic. Some groups, however, have special roles, like informing Usenet server administrators about newly created groups and groups that must be destroyed, discussing about whether a group should be created or destroyed, and test groups, where users can verify that their news reading software works properly (especially for posting messages to a server). The content of the test groups is also propagated among servers, in order to check if propagation works properly, but users do not usually read these groups.

It would thus be possible to use those test groups to post adequately formatted messages describing a peer-to-peer network and a node which is a member of that network, in order to allow other users to find an entry point into that network (the format of that message will be discussed later).

An alternative to using test groups for that purpose would be to create a dedicated group in the alt.* hierarchy[2]. This solution might be cleaner, but if alt.* groups can be easily created, they can also easily be destroyed (actually ignored by administrators), thus making the whole system vulnerable to attacks.

[2]Creating a new group is usually a long process, involving discussions about the usefulness of the new group and ending with a vote from the persons who took an interest in the discussion. The alt.* hierarchy is different in this respect since anyone is allowed to create a new group into it.

2.3.2 IRC

IRC (Internet Relayed Chat) is a real-time chat system based on a network of servers which are exchanging messages in a peer-to-peer way, without any central control. Users connect to their favorite server to join the system. They are then uniquely identified by a nickname, and can send messages either to another user or to a channel. Channels are virtual rooms, which the users can enter or leave. Every message sent to a channel is forwarded to all members of that channel; it can thus be assimilated to a multicast group. Any user can create a new channel, simply by "joining" a non-existing one.

It is thus possible for some nodes of a peer-to-peer network to connect to IRC in order to find other nodes to which they could connect. This is already implemented in GIrcCache for the Gnutella network, but we propose here to extend this to any network and to design a lightweight protocol for that purpose.

2.3.3 WWW Search Engines

GWebCache [Däm03] relies partly on the WWW search engines, which represent the well-known sources of information for finding GWebCache servers. A search engine is in essence a centralized system, which is subject to failure (in this case, a failure can be an attack, but also the disappearance of the company running the search engine or censorship). As there are several search engines available on the Web, this problem can be somewhat minimized, but not completely forgotten.

However, a search engine can be relied on in case none of the above-mentioned systems is available or known by the user.

2.3.4 Random Network Scanning

If none of the above solutions is available for a given user, the last resort possibility would be to try to connect to random IP addresses, until a peer is found. This solution should however be avoided, since this behavior can be assimilated to network scanning, which is considered as an attack by some network administrators.

Actual random scanning might prove inefficient; one optimization would be to first scan the IP network to which the scanner belongs (a list of these IP addresses can be computed from the scanner's IP address and network mask address). If no suitable node is found this way, the scanner can try random or systematic[3] scan in A-class IP networks; most of these have been attributed to large organizations, such as universities and Internet Service Providers, in the early day of the Internet.

[3]The decision to choose between random and systematic depends on the distribution of attributed IP addresses within the address class, and is out of the scope of this document, as well as is the algorithm used to implement the systematic scan.

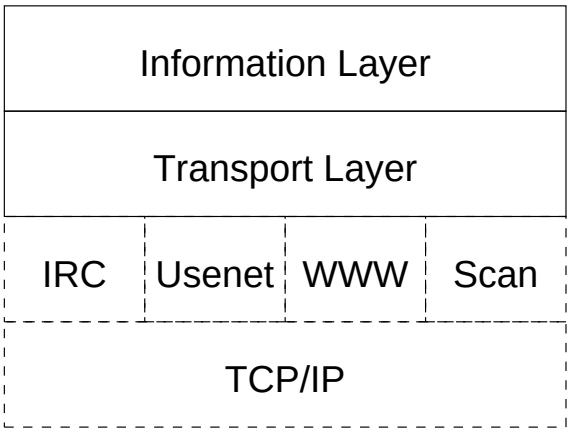


FIGURE 4 Protocol stack

2.4 Protocol Architecture

If we consider that IRC, Usenet, a WWW search engine and random network scan are several data communication media, we can design a two-layer system (Figure 4) composed of an Information Layer and a Transport Layer, running on the top of the various media which would allow a peer-to-peer application to use all of them in a transparent way.

We also describe basic considerations on the behaviors that must be observed by the peers when using those media in order not to overload them: IRC and Usenet are not designed for the purposes described here, and a massive use of these systems by peer-to-peer networks may rapidly overload them. The greatest care must thus be taken when implementing this system to minimize the risk of provoking a denial of service in these systems.

2.4.1 Information Layer

In the following, we call *binding information* any kind of information that is sufficient for establishing a connection with the node. In the case of TCP/IP, the IP address of the peer and the TCP port number to connect to form together the binding information. In the case of other protocols, the type of information that is needed may be different.

When the node wants to join a network, its own binding information is sent over the Internet through the Information Layer. The Information Protocol is XML based and has only one primitive: the *ad* primitive. The Document Type Definition of the messages is shown in Figure 5.

The messages are designed so that almost any kind of information can be sent. The only mandatory information is the name of the network which identifies the network among all peer-to-peer overlay networks (and which must be decided by the designers of the network) and the type of binding information contained in the message (in the case of the Internet, the type could be *TCP/IP*

```
<!ELEMENT ad (network,bind-data)>
<!ELEMENT network ANY>
<!ATTLIST network
          name #CDATA REQUIRED >
<!ELEMENT bind-data ANY>
<!ATTLIST bind-data
          type #CDATA REQUIRED >
```

FIGURE 5 The Document Type Definition of the XML messages of the Information Protocol

```
<?xml version="1.0"?>
<!DOCTYPE ad SYSTEM "p2padvertisements.dtd">
<ad>
  <network name="abc"/>
  <bind-data type="TCP/IP">
    <ip>192.168.123.123</ip>
    <port>12345</port>
  </bind-data>
</ad>
```

FIGURE 6 Example of an advertisement message for a P2P network named *abc* and using TCP/IP binding information

and the *bind-data* element could contain an *ip* element and a *port* element. In the case of Bluetooth or IrDA protocols, this information might be completely different). Figure 6 shows an example of such a message.

ad messages are then given to the Transport Layer, which will multicast them to any peer which is willing to receive them, according to the media which are available to the Transport Layer.

ad messages can also be received from the Transport Layer. The Information Layer must forward to the upper layer only information relevant to the desired network. All other information is dropped.

2.4.2 Transport Layer

The Transport Layer will send the messages it receives from the Information Layer over all the media it can access: Usenet, IRC, Web search engine and random network scanning. The messages will be fitted into a suitable form, depending on the media.

Media are assigned priorities: the medium having the highest priority is used first. If it does not yield any result after a given timeout, the next medium in the priority list is tried (this does not necessarily mean that one gives up the previous one), and so on until there is no medium left. If no result has been reached at this point, joining the network doesn't fail, but is considered as being delayed. Once all media have been given up, the network connection fails. It is however possible to remain connected to a given media for days (e.g., IRC) or

to poll information at regular intervals over a long period of time (Usenet, Web search engine). Joining a network might thus take a lot of time, but failure will occur only if none of the media is available or if the user interrupts the procedure.

IRC Medium

Before sending a message using the IRC medium, a TCP/IP connection must be made to an IRC server. The address and port number for establishing that connection is considered as common knowledge and is given by the user who installs the peer-to-peer software. Once the connection has been established, the *#p2padvertisement* channel will be joined and only one message will be sent to the channel, providing the users already connected to the channel with the binding information of the current node.

All messages sent by the other users over that channel will be forwarded to the Information Layer.

The use of IRC medium gives a naturally limited lifetime to the advertised information: messages will only be visible to the nodes already connected to the channel at the time of the sending of the message. There is no need to re-send the message: when a newcomer arrives, it sends its own information, and one can react on it (i.e., try to establish a connection with it) if it is found suitable.

Usenet Medium

Before sending a message using the Usenet medium, a TCP connection must be made to an Usenet server. As in the IRC medium, the address and port number for establishing that connection is considered as common knowledge. Once the connection has been established, the *alt.test* newsgroup will be requested and some message headers will be read (this group usually has a lot of traffic, so reading the messages are limited to the last N ones, N being given as a configuration parameter by the user, typically 100 to 500 messages). Only the messages whose *Subject* field is set to "p2padvertisement" will be read (i.e., their bodies will be requested from the server). The bodies of those messages are then passed up to the Information Layer. If no suitable information is found, the message coming from the upper layer is posted to the newsgroup.

Usenet servers associate each message with an index number which is growing monotonically. It is advisable to cache the last index number which has been requested, in order to avoid reading the same messages again during the next poll.

The use of the Usenet medium gives a naturally limited lifetime to the messages: in order to save disk space, Usenet servers delete older messages. The usual lifetime of a message is between a day and a month, depending on the newsgroup and on the server.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>p2padvertisement</title>
  </head>
  <body>
    <p><a href="binding-data.xml">binding-data.xml</a></p>
  </body>
</html>
```

FIGURE 7 Example of an HTML document publishing binding information on the WWW

Web Search Engine

If IRC and Usenet are not available at a given location (because the local network administrator is blocking the necessary TCP ports with a firewall), the WWW might still be reachable, even if its use as a broadcast medium is more complex.

Compared to the two previous media, the use of a Web search engine requires an additional Web server for publishing the binding information. Moreover, different web search engines have different user interfaces, which require specific programming interfaces.

However, the use of a Web search engine as a diffusion medium differs little in principle from the Usenet medium. Before sending a message using this medium, a TCP/IP connection must be made to the Web server hosting the search engine. The address and port number for establishing that connection is considered as common knowledge and is given by the developer who made the programming interface. Once the connection has been established, a request can be sent using HTTP protocol and the resulting HTML page can be parsed in order to find URLs of Web pages containing binding information. Each of those pages must then be parsed to get the necessary information, which can finally be passed to the upper layer.

Advertising the binding information is however less straightforward than in the previous cases: it must be published on the Web, in a form that is easily exploitable by search engines, and the Web document must be registered into the search engine. Simply publishing an Information Protocol message might not work with all search engines, since the message is in XML, not HTML. However, the registration needs to be done only once for a node.

The proposed way to publish the binding information on the Web is shown in Figure 7. The actual binding data is located in the `binding-data.xml` file, to which the HTML document links. The *title* element of the HTML document is mandatory and used as a filter criterion for selecting the results given by the search engine. Maintaining the information up-to-date consists in maintaining the content of the `binding-data.xml` file up-to-date.

One can notice that the use of Web search engine medium does not give a natural lifetime to the binding information. This information must be kept up-to-date, by means which are out of the scope of this document.

Moreover, compared to the previous media the use of a Web search engine as a diffusion medium for binding information is thought not to be a good choice for automated network binding, for the following reasons:

- There is no standard programming interface between search engines (the layout of the result page of search engine A is different from the one of search engine B).
- There is no standard programming interface within one search engine (the layout of the result page can change at any time without warning).
- The request might yield unwanted results, which makes the parsing of the result page less efficient.

However, some search engines might provide an automaton-friendly search interfaces (like the SOAP [ea00] interface of Google [Goo03b]). Some others might be developed in the future.

Random Network Scan

If all previous media fail to return the desired information, or if the user who installs the software is not able to give any of the needed information (IRC/Usenet server IP address, Web search engine interface), the last resort solution is to connect to some random or pseudo-random IP addresses in order to find a peer to which to connect.

This medium is not a diffusion medium in the sense that it advertises the node's information to other nodes, but it is nonetheless potentially able to return binding information to the upper layer.

Heuristic methods that can be applied are, among others:

- Trying IP addresses of nodes to which the current node was connected earlier (this requires for the node to keep a history of its neighbors).
- Trying IP addresses of the same sub-network than the current node.
- Trying random IP addresses inside the network of great domestic or foreign institutions (e.g., universities). A list of these institutions is asked from the user.
- Trying random A class IP addresses from large Internet Service Providers (these addresses or networks are asked from the user).

None of these methods might yield satisfactory results, and they can even be considered as illegal in some countries. The use of this medium should be disabled by default, and enabled only on the user's request.

3 FORMAL TERMINOLOGY

A formal terminology is necessary in order to clarify explanations; the one used in this work is mainly based on the well-accepted vocabulary of graph theory [DM03].

3.1 Network

The system is composed of one *broadcast channel* B, a set of nodes $\mathcal{V}$, a *network* $\mathcal{G}$ and a *knowledge network* $\mathcal{K}$.

$\mathcal{G}$ is in this work defined as a couple composed of a set of *nodes* $N \subset \mathcal{V}$ and a set of undirected *connections* C; therefore $\mathcal{G} = (N, C)$. Two nodes N_0 and N_1 are said to be connected to each other if $\exists c \in C$ that links N_0 and N_1; c is noted $c : N_0 \leftrightarrow N_1$. $\mathcal{G}$ is such that all its nodes are connected to at least one other node.

The nodes have the following properties:

- Every node knows the existence of the broadcast channel.
- The nodes that have at least one *connection* to another node form the *network*. This differs from the traditional definition of a network, which can contain also nodes that are not connected to any other node. In this work, we consider that a node which is not connected to at least one other node is *not* a part of the network. However, it is permitted to add nodes from $\mathcal{V}$ to N, on the condition that corresponding connections are added to C. In other words, that means that any node which is added to the network must also be connected to the network.

- A *path* is a finite, ordered set of nodes $\mathcal{P} = \{N_i\}, \mathcal{P} \subset N$ where $0 \leq i \leq n, n \in \mathbb{N}$, each node N_i is connected to node N_{i-1} if $i \geq 1$ and to node N_{i+1} if $i \leq n-1$. $\mathcal{P}$ is then said to exist between node N_0 and node N_n.
- The network is said to be *partitioned* into two disjoint *connected components* $\mathcal{C}_1 \subset \mathcal{G}$ and $\mathcal{C}_2 \subset \mathcal{G}$ if $\forall (N_1, N_2) \in (\mathcal{C}_1, \mathcal{C}_2)$, there exists no path between N_1 and N_2.
- A node N_i connected to node N_0 is called a *neighbor* of N_0. The set of neighbors of N_0 is noted C_{N_0}.
- The number of neighbors of N_0 is called the *degree* of N_0. We note $d(N_i)$ the degree of node N_i for $N_i \in \mathcal{V}$. Moreover, $d(N_i) = \left|C_{N_0}\right|$.
- The maximum number of neighbors that N_0 is ever allowed to hold is called the *maximum degree* of N_0. We note $d_{max}(N_i)$ the maximum degree of node N_i for $N_i \in \mathcal{V}$. The ratio $d(N_i)/d_{max}(N_i)$ is sometimes referred to as the *filling* of node N_i.

$\mathcal{K}$ is defined as a a couple composed of $\mathcal{V}$ and a set of directed *knowledge relations* K; therefore $\mathcal{K} = (\mathcal{V}, K)$. The node N_0 is said to know the node N_1 if $\exists k \in K$ that relates unidirectionally N_0 to N_1; k is noted $k : N_0 \rightarrow N_1$.

- A node N_i that N_0 knows about is called a *known node*. The set of nodes that N_0 knows about is noted K_{N_0}.
- Each node N_0 defines a threshold for the number of nodes it wants to know about at a time; this threshold is noted $\left|K_{N_0}\right|_{max}$. It is in practice defined as a percentage of $d_{max}(N_0)$ and called *minimum known nodes before leaving* (see Section 4.2).

3.2 Nodes

When the behavior of a given node is described, the following terminology will be used:

- The network is called *the network* or $\mathcal{G}$.
- Interactions between nodes (either direct or through the broadcast channel) allow the nodes to discover the existence of other nodes; in other words, nodes get to know other nodes. This is formally defined by the addition of a knowledge relation to K.
- The node from the point of view of which a behavior is being described is from now on called *the node* or N_0. A node $N_i \in \mathcal{V} - (C_{N_0} \cup \{N_0\})$, i.e., a node that is neither N_0 nor a neighbor of N_0 is called *the other node*. If N_i is not known by N_0, i.e., $N_i \notin K_{N_0}$, it is called an *unknown node*.

3.3 Broadcast Channel

The following terminology will be used to describe the properties of the broadcast channel B:

- Connections to B are represented by a network $\mathcal{B}$ composed of $\mathcal{V} \cup B$ and a set of connections J; therefore $\mathcal{B} = (\mathcal{V} \cup B, J)$. If N_0 is connected to the broadcast channel, then $\exists s \in J$ that links N_0 to B.
- *Joining the broadcast channel* is synonym to connecting to the broadcast channel, in other terms one adds one element to J which represents a connection between $\mathcal{B}$ and N_0.
- *Leaving the broadcast channel* is synonym to disconnecting from the broadcast channel, on other terms one removes the element of J which represents the connection between $\mathcal{B}$ and N_0.
- The broadcast channel has a maximum capacity; this means that $|J|$ cannot be higher than a given value $|J|_{max}$: we always have $|J| \leq |J|_{max}$.

4 THE SIMULATOR

In order to study the formation of the network according to the node behaviors described in Chapter 6, a simulator has been developed. It is not a network simulator in the usual acceptance of the term, i.e., it does not simulate as accurately as possible a network topology and network protocols. The use of such a network simulator (like e.g., NS-2 [FV03]) would have required to simulate a real network topology, making assumptions on the links' capacities and latencies, define the presence (or absence) of routers or switches and their characteristics, network protocols between the nodes (the use of IP and TCP is obvious, but lower protocols layers can have a strong influence on the performance of TCP/IP), to name only some of the decisions that should have been taken. These decisions would have been arbitrary, and would probably have influenced the results of the simulation. Besides, it would also have been necessary to implement an actual peer-to-peer protocol to be used in the simulation, which would have been outside of the goal of this work.

Two separate simulators have been implemented, one using IRC as the broadcast channel, and another one using Usenet for that purpose. The actual implementations have most of the code in common, and only the broadcast channel-specific functions are different.

4.1 Broadcast Channels

The replacement of the central index, used in many systems for the nodes to be able to join the network, requires a broadcast channel, shared both by the

nodes which are members of the network and by the nodes which try to join the network. This channel is the only possible way a node outside the network can find information about the nodes inside, and the nodes inside can advertise their presence to the outsiders.

The broadcast channel is, by definition, a communication channel that is available to all nodes in a network. Any message sent to the broadcast channel will be received by all the nodes in the network. Wireless network devices such as WLAN [IEE99] or Bluetooth [IEE02] benefit of an implicit broadcast channel: any node can broadcast on the air information to any other node which is within range of its radio transmitter. This feature is used by the *Mobile Chedar* middleware [KVW+04], where the mobile nodes use Bluetooth's Service Discovery Protocol to locate the other nodes that are within range of their transceiver.

IP networks also have a broadcast feature. It is however rarely used, since, given the number of nodes of the Internet, it would produce enormous amounts of traffic; that is why broadcast messages are filtered out by gateways [Ste94, page 171]. Besides, whereas in radio networks nodes within range are likely to be interested in establishing communication, computers located on the same local network are not necessarily interested in joining a peer-to-peer network. For that reason, the broadcast channel must have the following properties:

- it must be like a multicast channel, rather than a broadcast channel, i.e., a transmission channel that is shared only by the nodes which have explicitly registered to it; this limits the sending of the "broadcast" information only to the nodes which are actually interested in that information.

- its existence and the information that one must know in order to access it must already be known to the node which wishes to join it.

Following the aforementioned guidelines, two different existing infrastructures are considered while selecting a broadcast channel: IRC and Usenet.

Moreover, even though the name "broadcast channel" is not appropriate, it will be used in this work, because of the connotation of the word "broadcast" which conveys the notion of a medium shared by all users of the system (even though, in practice, only the nodes that connect to IRC or to Usenet will receive the information).

4.1.1 IRC

IRC (for Internet Relayed Chat) is a worldwide distributed system that allows users to exchange instant text-mode messages. IRC relies on a network of servers linked together using TCP connections in a peer-to-peer manner, which route messages from the source user to the destination user. The topology of the network of IRC servers, a tree, is carefully planned by the community of the IRC admins (i.e., the administrators of IRC servers). Each user is identified by a *nickname*, which is an identifier, unique within the system. Messages are sent either to specific users (the user's nickname being the destination address) or to groups

of users called *channels*. A channel is similar to an IP multicast [Ste94, section 12.4] address: users register to the channel at any time, and the messages sent to the channel are forwarded to all the registered users. Channels are identified by a name, which is unique within the system. The nickname namespace is disjoint from the channel namespace.

Once a TCP connection has been established with an IRC server, the following operations are available:

nick: identify oneself with a given nickname. If another user is already connected with this nickname, an error message is issued; the user must choose another nickname. This command is also used to change one's nickname during a session.

join: register to a channel. If the channel does not exist yet, it is created.

privmsg: send a message to a user or a channel.

part: unregister from a channel. If the channel has no member any more, it is destroyed.

quit: disconnect from the server. The user is unregistered from any channel he/she was member of, and his/her nickname is available for another user to use.

This list is not exhaustive, and voluntarily limited to the operations useful within the scope of this work.

IRC is essentially synchronous, since messages are pushed toward the user. The delivery delay of a message is of the order of a second.

4.1.2 Usenet

Usenet is a worldwide distributed system that stores text messages, called *articles*, for a limited amount of time. It relies on a network of servers linked together in a peer-to-peer manner. When a user posts an article to one server, it is slowly propagated to the other servers. After some delay, the article is available on all servers of the network. Other users can then retrieve it from any server. A given amount of time after having been received, articles are removed from the server in order to save disk space; this process is called *expiration* of the articles. Messages are sorted by groups. Each group has a unique name within the system, and names are hierarchically organized [Wri99].

Once a TCP connection has been established with a server, the following operations are available:

group: select one group from which to retrieve articles.

article, head, body: retrieve one article (respectively the article's head, the article's body).

post: send one article to a server; the server must permit the user to post articles.

quit: disconnect from the server.

This list is non-exhaustive, and voluntarily limited to the operations useful within the scope of this work.

Usenet is essentially an asynchronous system, since the users have to explicitly poll new messages from the server. The propagation delay of a message from one server to another depends on the configuration of the server (i.e., how often a given server synchronizes with its neighbors in the network), and may be of the order of several hours.

4.1.3 Other Possible Broadcast Channels

Web servers were mentioned in [WVV03] as another possibility for a broadcast channel: a node advertises its own existence on a dedicated Web service (e.g. [Däm03] or gNetCache[1]), which publishes on the Web all these advertisements. This system is similar to Usenet, but with the following distinctions:

- the advertisement publishing system is not distributed, and thus more prone to failure
- the removal of out-of-date advertisements is not guaranteed; in practice, such available services do not take into consideration any expiration time for advertisements.

This work will however be limited to the study of the systems using IRC or Usenet as broadcast channels.

4.2 Design of the Simulator

The design of the simulator has been kept as simple as possible, and the following decisions have been made:

- All the nodes are equal, except in terms of their maximum degree. This value synthesizes the limitations of each node in terms of processing power, memory and network capacity. It means that, if a node has a maximum degree of 20 in the simulation, the node is considered to have such processor, memory and network link characteristics that it can accept up to 20 connections.
- The simulation has no time counter. There is no way to determine how much time has elapsed between two events. There is however an iteration counter, which can be used to tell that an event happened after another.

[1] gNetCache is currently available only as an archived source code package. It is not maintained anymore and probably not in use at the current time

- The simulator, as a program, is single-threaded, and all nodes are given a chance to run in a first-come-first-served fashion. The nodes are not programmed as state machines, but have a behavior which is equivalent to the state machine described in Section 4.3.

  All the nodes have the same priority, and are run in turns. When a node is running, it cannot be preempted by another one: the next node will have a chance to run only when the previous one yields control of the processor. This will affect the results of the simulator, but all simulators are inherently inaccurate on this point, because of their nature. However, since no assumption has been made on the underlying network topology and characteristics, there is no way to tell e.g., how much time the establishment of a connection between two nodes would take, or how much time the processing of received data by the node would take; this scheduling method is therefore as good as any.

- The use of random numbers (or rather pseudo-random numbers) has been minimized, in order to be able to reproduce easily one given simulation and obtain the same results every time.

- In order to simulate the progressive arrival of nodes into the system, all nodes are inactive (i.e., idle) when the simulation starts. At the beginning of each iteration, a given number of inactive nodes become active (i.e., they take actions, such as establishing connections with the broadcast channel or with each other); only active nodes are scheduled to run. All nodes are therefore progressively becoming active, thus simulating the introduction of new nodes into the system.

  The number of nodes that become active at each iteration is a percentage of the total number of nodes in the system. It is not a random number, in order to minimize the influence of randomness on the system.

The simulator, as a computer program, takes parameters which are set when the simulator is started, and which do not change during the simulation. These parameters are:

Number of nodes: the total number of nodes in the system.

Number of iterations: the number of times the simulator will run each node, allowing it to perform some action (i.e., perform the equivalent of one or more state transition(s) in a simulator that would have been implemented as a state machine). This value can be dynamically increased if there are still nodes performing actions at the end of the simulation, i.e., if the nodes have not finished aggregating themselves when the predefined number of iterations has been executed.

Maximum neighbor distribution: the statistical distribution used for generating each node's maximum degree, which determines what random number

generator is used. The distribution which is used is the Gnutella Distribution, described in Appendix 2, since it is the actual distribution of the degree of the nodes in the Gnutella network. Other distributions are also available: normal, exponential and single value (all the nodes have the same maximum degree).

Activation probability: the percentage of the total number of nodes that become active at the beginning of each iteration.

Minimum desired neighbor filling: the degree (expressed as a percentage of the maximum number of neighbors) above which the node will stop trying to discover other nodes using the broadcast channel. This value is the same for every node.

Minimum known nodes before leaving: the number of known nodes (as a percentage of the maximum number of neighbors) above which the node will leave the broadcast channel and try to establish actual connections with its known nodes. This value is the same for every node.

Random number generator's seed: the seed which is used to initialize the uniform random number generator.

Simulations based on Usenet as a broadcast channel also take the following parameter:

Advertisement expiration age: the age (expressed as a number of iterations) of an advertisement after which it is removed from the channel (i.e., cannot be read anymore). This simulates the real behavior of Usenet servers, which delete messages after some time (see Section 4.1.2) or at the date indicated by the *Expires* message header [HA87]. In the simulator, the expiration time is calculated from two parameters, a and b using the formula $a \times d_{max}(N_0) + b$. This allows to set longer expiration times for nodes which are expecting many neighbors, and shorter times for the nodes that need less neighbors.

The following parameters are available but not used:

Incoming connection acceptance probability: the probability (as a percentage) for a node to accept an incoming connection, provided that nothing else forbids the node to accept it. This value is by default 100% for all simulations.

Connection decision probability: the probability (expressed as a percentage) for a node to request a connection to another node, provided that nothing else forbids the node to send this request. This value is by default 100% for all simulations.

Channel join probability: the probability (expressed as a percentage) for a node to join the broadcast channel, provided that nothing else forbids the node to join it. This value is by default 100% for all simulations.

Channel leave probability: the probability (expressed as a percentage) for a node to leave the broadcast channel, provided that nothing else forbids the node to leave it. This value is by default 100% for all simulations.

These parameters are not used, because on the one hand, they have no physical justification (except maybe simulating random network or random node failures such as lost IP packets or rejected connections in case of an overloaded node), and on the other hand, they induce more randomness in the simulation, which is something one tries to avoid.

Each simulation is then composed of three phases:

1. the initialization phase,
2. the simulation itself,
3. the synthesis of the results.

4.2.1 Initialization

During the initialization, the uniform random number generator[2] (on which all other random number generators are based) is initialized with the *random number generator's seed* parameter. Then the nodes are instantiated in the simulator and assigned a maximum degree, using the *maximum neighbor distribution*, which is by default the *Gnutella Distribution random deviate* described in Appendix 2. This value is fixed for the whole length of the simulation.

4.2.2 Simulation

The simulation is run as two nested loops:

- The outer loop runs each simulation step a given number of times (as specified by the *number of iterations* parameter). If in the last simulation iteration, there are nodes which want to perform more actions, the number of iterations is increased by 10%. This allows the simulation to run to an end even if the original number of iterations was underestimated.
- The inner loop gives each node the possibility to become active, and each active node the possibility to perform the following actions, in this order (each action is described in detail in Chapter 6):
    1. Attempt to connect to known nodes. When a connection actually takes place, the simulator immediately runs the destination of the connection in order to finalize the connection; this node, however, does not take any other action at this point.

[2] Three different uniform random number generators are available to the simulator, which are of uneven quality. The one used in the simulation must be chosen when the simulator is compiled from its source code. See Appendix 2.2 for more details on those.

When a connection has been established, the two peers exchange a list of their current neighbors, which is stored into each respective node's *P2P queue* (see Chapter 6 for a definition of the P2P queue).

2. Join the broadcast channel, if needed.
3. Discover other nodes using the broadcast channel. This is done by:
   - Retrieving advertisement stored on the newsgroup in the case of the simulator using Usenet as a broadcast channel
   - Advertising itself to all nodes connected to the channel in the case of the simulator using IRC as a broadcast channel. All the nodes that are connected to the channel are processing the advertisement immediately (this means that the simulator lets the recipient nodes run in order to process this information; no other action is taken by the recipient of the advertisement). If the target of the advertisement has enough known nodes in its *channel queue* (see Chapter 6 for a definition of the channel queue) when the node advertised itself, the recipient node leaves the channel immediately at this point.

   The information about the advertisement is stored in the node's channel queue.
4. Leave the broadcast channel, if applicable.
5. Post an advertisement in the case of the simulator using Usenet as a broadcast channel (in the simulator, it is not necessary to actually be on the newsgroup to post a message, even though it is of course needed in real life).

- the simulator using Usenet as a broadcast channel expires old advertisements.

When the node attempts to connect to known nodes, it selects preferably nodes from the P2P queue. The reason for this is to keep the channel queue as full as possible, in order to reduce the usage of the broadcast channel. If the P2P queue is empty, the channel queue is selected. When the selected queue has been emptied, the other queue is not selected, even if it is non-empty. If the node succeeds in connecting to another node, there will be no other attempt to connect another node during this running turn of the node: the next node is run immediately. The reason for this behavior is double:

- It prevents the node from making as many connections as possible at once while holding back all other nodes. In a real environment, all active nodes are attempting to make connections concurrently, thus this behavior mimics the concurrency of the nodes' actions.
- It prevents the node which has been able to make one connection from further using the broadcast channel, thus reducing the network traffic on the channel.

4.2.3 Synthesis

Once the simulation has been run, the simulator gathers data about the simulation into a file (see Figure 23 for an example of such a file).

The following technical data about running the simulation on the computer is collected:

- used time
- virtual memory usage
- number of iterations necessary in order to complete the simulation

The following data related to the disjoint connected components[3] which have formed during the simulation is collected:

- Number of disjoint connected components (*Number of disjoint networks*)
- Number of nodes in each disjoint connected component (*Nodes forming a network with node N*)
- Maximum of the degrees of all the nodes in each of the connected components (*Biggest max neighbors*)
- Distribution of the maximum degrees of the nodes of each connected component (*Connectability distribution*)
- Distribution of the actual degree of the nodes of each connected component (*Connectivity distribution*)
- Distribution (for each connected component) of the "filling" percentage of the nodes, i.e., ratio between the actual degree over the maximum degree, which shows how close the actual distribution of the degree at the end of the simulation matches the distribution of the maximum degrees assigned to the nodes at the beginning of it (*Neighborhood filling level*)

The following data related to the usage of the broadcast channel during the simulation is also collected:

- Number of joins to the channel and leaves from the channel (*Total channel joins/leaves*)
- Total amount of traffic to and from the broadcast channel, in bytes (*Global traffic on channel*, see Chapter 5 for a detailed calculation of the traffic)

[3] Each connected component is identified by one of the nodes which belong to it. The simulator searches the connected components starting from node 0 and finds all the nodes which belong to the same component (i.e., there is a path from that node to node 0). It then searches the first node which does not belong to the first connected component (this node will be the identifier for the second connected component) and applies the same method, and so on until all the nodes have been associated to one connected component.

- Maximum number of nodes simultaneously connected to the channel (*Max users on channel*)

- Number of retrieved and posted advertisement in the case of the Usenet simulator (*Number of advertisements retrieved*, *Posted advertisements*)

- Number of peer-to-peer connections that have been possible thanks to the channel (i.e., using the channel queue) and number of peer-to-peer connections that have been possible thanks to the exchange of neighbor lists between two peers (i.e., using the P2P queue) (*Channel/P2P connections*)

External scripts can then be used to calculate statistics about the simulations (especially, if several simulations have been run with the same parameters but with different seeds for the random number generators, one can compute the average and the standard deviation of the above described results) and produce graphics showing the evolution of one result depending on the values of one parameter.

The simulator is also capable of calculating values characterizing the connected components of the network. These values are:

- Average clustering coefficient, which represents the average "density" of connections between the neighbors of a node.

- Longest shortest path, which represents the maximum "diameter" of the connected component; routing a message between two nodes of the component will require at most this number of hops.

- Average shortest path, which represents the average "distance" between two nodes, i.e., the average number of hops needed for routing a message between two nodes of the network.

These values take however a long time to compute (depending on the size of the network, it can take longer than the simulation itself); therefore they must be explicitly required when running a simulation (in practice, the code must be activated in the simulator at compile time).

4.3 The Simulator as a State Machine

Traditional network simulators implement the nodes as state machines, since the nodes are actually implementing specific networking protocols, which are often designed as state machines. Figure 8 represents a state machine which has a behavior equivalent to the nodes in the simulator. The flowchart is simplified compared to the simulator, since it does not take into account the fact that there are two distinct queues of known nodes. Moreover, the state machine assumes that the broadcast channel is an IRC channel.

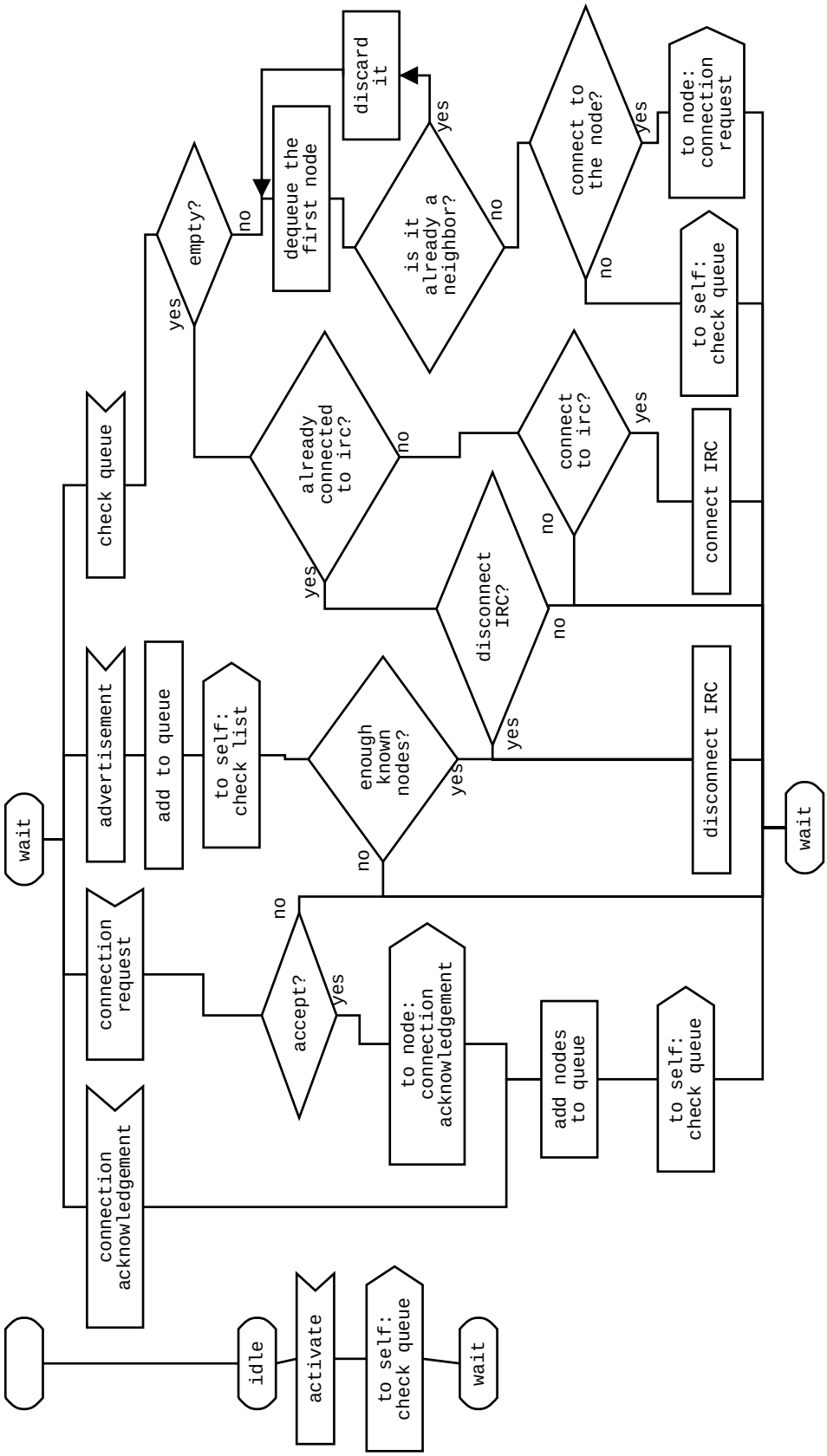


FIGURE 8 Flowchart of the state machine equivalent to the nodes in the simulator

The state machine has one idle state, which is the state in which the node is when it has not yet been activated. The *activate* message, sent by e.g the user of the node, will change the state from idle to wait. During this transition, the node also sends a message to itself to check the queue of known nodes. This will immediately trigger a state transition, where the node will check the queue, notice that it is empty, and join the broadcast channel. At this point, the node has nothing else to do besides waiting for incoming advertisements from the broadcast channel or incoming connection requests from other nodes.

A simulator based on this state machine would schedule the next node after the node has made one state transition, in order to simulate multitasking. On the opposite, the simulator described in this work runs the whole process described above in one shot. Both simulators would however be in the same state after the above steps have been performed.

Since both the hypothetical and the actual simulator would be single-threaded and running on one single processor, two events E_1 and E_2 (triggering the sending of a message in the state machine simulator or actions in the actual simulator) targeted at node T cannot happen concurrently while the simulator is running. Moreover, let us suppose that E_1 occurs before E_2. We can then show that the two types of implementation yield equivalent results:

- In the state machine simulator, the message sent on E_1 will be placed in the input queue of T, then the message sent on E_2 will be placed in the same queue. When T will actually process these messages, it will process them in the order they appear in the queue, i.e., first the message from E_1, then the one from E_2. There is no way the messages would be placed in the queue in an order different from which they were sent (i.e., different from the order in which the events happened).

- In the actual simulator, when E_1 occurs, an action is taken and its result is immediately applied to T. Then when E_2 occurs, the result of the action is applied to T also. The two actions have occurred in the same order as the events have happened. Since the execution of the action is immediate, there is no way the processing of E_1 can be held back by the processing of E_2.

In both simulators, the order of the events and their associated processing would be the same; it is therefore reasonable to state that they would yield the same results.

5 TRAFFIC ESTIMATION

Although the simulator described later in this work does not implement any protocol as such, it is possible to estimate the traffic generated by the nodes toward the broadcast channel, by estimating what protocol request and reply messages would be used if a protocol (IRC protocol or NNTP) was actually implemented and how much bandwidth these messages would require.

Each node needs to realize the following four operations in order to use the broadcast channel:

joining the channel: this operation consists in establishing a TCP connection to the server that handles the broadcast channel (i.e., an IRC server or an Usenet server), and whatever protocol-specific initialization is necessary to actually join the broadcast channel.

sending an advertisement: this operation consists in sending an advertisement, as described in Section 2.4.1. For further reference, the length of this message is 171 characters (once the extraneous white space has been removed).

receiving an advertisement: this operation consists in passively receiving an advertisement that is pushed towards the node by another node (like it is the case in IRC), or actively poll an advertisement that is stored on the server (like it is in Usenet).

leaving the channel: this operation consists in leaving the broadcast channel and disconnect from the server at TCP level.

5.1 IRC Protocol

The IRC protocol [OR93] is a text-mode, line-oriented protocol. Each protocol message is one line of text which is composed of one optional prefix (which always starts with a colon), a command, a list of parameters, and an end-of-line marker. The different elements of a message are separated with spaces, and the end-of-line marker is the ASCII "Carriage Return" character followed by the "Line Feed" character.

The protocol needs the following parameters:

Nickname: each user connected to an IRC network is required to have a unique nickname. The simplest way for the nodes to choose a nickname is to generate it at random, and in order to minimize the risks of nickname collision (i.e., two different users choosing the same nickname), the length of the nickname will always be the maximum length, i.e., 9 characters.

Server name: this is the domain name of the IRC server the node connects to. Its length can be anything, therefore the chosen value is 10 characters.

Channel name: the name of the channel is *#p2padvertisement* (as described in Section 2.4.2) , therefore its length is 17 characters.

Real name: this is a mandatory parameter for the USER command, but it can be left empty. Its length is therefore 0 characters.

Username: this is normally the username of the user that connects to the IRC server. It has to be unique within the node's computer, and can be as simple as one character. Its length is therefore 1 character.

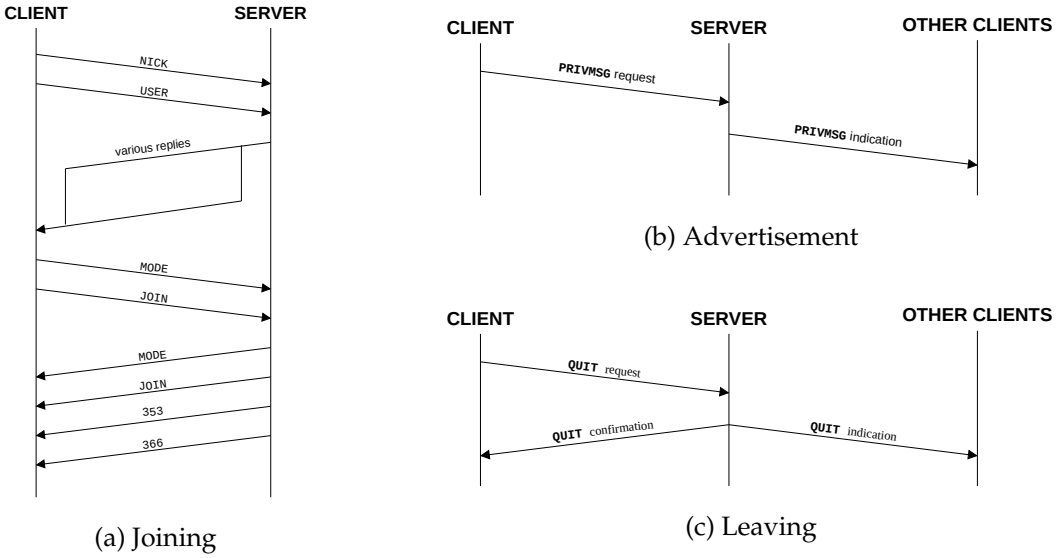
Hostname: this is the name of the machine from which the node connects to the IRC server. Assuming that it is always an IP number, its length is therefore 15 characters.

Client mode: it is advised that the nodes set their user mode to "invisible" (noted `i`), in order not to be seen by robots that send unsolicited messages to IRC users. The length of the client mode is therefore 1 character.

Quit message: When quitting IRC while on a channel, one can send a message to other users. This can be left empty, and has therefore a size of 0 characters.

5.1.1 Joining the Broadcast Channel

The connections process requires the client to connect to the IRC server at TCP level. Once this connection has been established, mandatory messages are exchanged as shown in Figure 9(a): the client sends a `NICK` request and a `USER` request (see Figures 24(a) and 24(b) respectively). The server then replies with various messages, including a greeting, information about the state and features of



(a) Joining

(b) Advertisement

(c) Leaving

FIGURE 9 Minimal IRC protocol sequences

the server, and the "message of the day" (usually rules defined by the administrator of the server about its utilization); these messages are all in the form of numerical replies (see Figure 25(a)) and amount to $3488 + 50 \times \mathit{nickname} + 51 \times \mathit{channel}$ bytes for the IRC server `irc.jyu.fi`. The client then sends `MODE` and `JOIN` requests (Figures 24(c) and 24(d)), and the server replies with the corresponding confirmations (Figures 25(b) and 25(c)) and sends the list of the members of the channel. Since an IRC message cannot exceed 512 bytes overall, the server may send several *List of names* replies (see Figure 25(d)); the last *List of names* reply is followed by the *End of names* reply (see Figure 25(e)). Given the limited length of the IRC message, one can fit at most 46 names (of length 9 bytes) in one *List of names* reply message. The traffic generated by the list of names is thus equal to $508 \times (\mathit{members}/46) + 48 + 10 \times (\mathit{members} \bmod 46)$ bytes.

The total traffic T_j (in bytes) generated by a node connecting to the broadcast channel is thus given by the following formula:

$$T_j = 4978 + 54 \times \mathit{members} + 508 \times (\mathit{members}/46) + 10 \times (\mathit{members} \bmod 46) \quad (1)$$

The number of members on the channel, *members*, includes the node that is currently joining the broadcast channel.

5.1.2 Advertising

Once the node has joined the broadcast channel, it can immediately advertise itself to the other members of the channel, by sending a `PRIVMSG` request (see Figure 24(e)). This command is then forwarded to all other members of the channel, which receive a `PRIVMSG` indication (see Figure 26(a)), as illustrated in Figure 9(b).

The total amount of traffic T_a generated by one node advertising itself on the broadcast channel is therefore:

$$T_a = 199 + (members - 1) \times 228 \quad (2)$$

The number of members on the channel, *members*, includes the node that is sending the advertisement.

5.1.3 Leaving the Broadcast Channel

When the node decides to disconnect from the broadcast channel, it sends a `QUIT` request (see Figure 24(f)) and receives an error message from the server (Figure 26(c)); the server forwards at the same time a `QUIT` indication to all other members of the channel.

The total amount of traffic T_l generated by one node leaving the broadcast channel is therefore:

$$T_l = 62 + 39 \times (members - 1) \quad (3)$$

The number of members on the channel, *members*, includes the node that is currently leaving the channel.

5.2 NNTP

Like the IRC protocol, NNTP [KL86, Bar00] is a text-mode, line-oriented protocol. Each protocol message is made of ASCII characters and composed of one command (for the queries) or one 3-digit code (for the responses), followed by a white-separated list of arguments and terminated by a "Carriage Return-Line Feed" character pair.

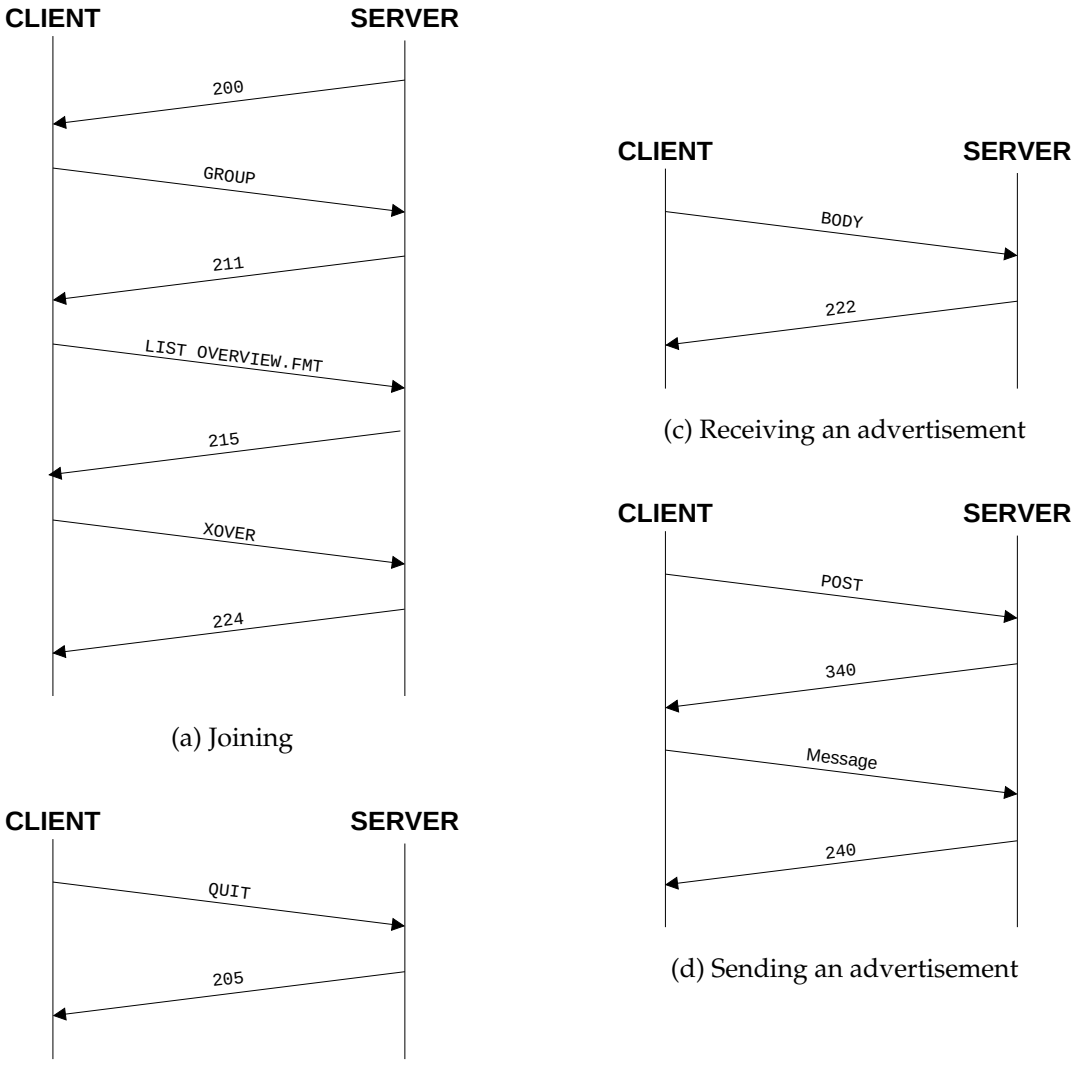
Some messages, especially the `POST` query and the reply to the `BODY` query, contain multiple lines, and are then terminated by a period on a single line.

The protocol needs the following parameters:

group name: the name of the newsgroup where the advertisements will be sent to and retrieved from. The newsgroup used in this case will be `alt.test`; the length of the group name is therefore 8 characters.

numbers: numbers are used in NNTP to identify articles locally on the server. Since the numbers are represented in text format, their length (in terms of characters) may vary. Since alt.test is a group with a lot of traffic, we can estimate that the numbers will be coded on 7 positions in average, i.e., that the length of any number will be of 7 characters.

overview record: the Usenet servers maintain, for each newsgroup, a so-called *overview database* that summarizes the content of the group. Each record of that database concerns one article and is a tabulation-separated list of fields, such as the subject, the author and the posting date, as well as its length and



(a) Joining

(b) Leaving

(c) Receiving an advertisement

(d) Sending an advertisement

FIGURE 10 Minimal NNTP sequences

its relation to other articles (especially which other article(s) it is a reply to). The length of such records is variable, but an acceptable average length of the record can be computed after taking a snapshot of the `alt.test` group; the result of this calculation is about 212 characters.

5.2.1 Joining the Broadcast Channel

The connection process requires the client to connect to the Usenet server at TCP level. The following sequence of protocol commands is illustrated in Figure 10(a). First, the client receives a greeting message from the server (see Figure 28(a)), which specifies to the client if posting is allowed or not (if the client is not allowed to post, the node cannot advertise itself; however, it can still receive the

advertisements from other nodes). The client then sends a `GROUP` command (Figure 27(a)) in order to select the appropriate newsgroup, and the server replies with the estimated number of articles in the group, the indexes of the first and last articles and the name of the group (Figure 28(b)). The client then sends the `LIST OVERVIEW.FMT` command (Figure 27(b), in order to receive the list of the fields of the overview database (Figure 28(c)). Once it has received it from the server, it can send the `XOVER` command (Figure 27(c)) to receive a complete list of articles between a given range of indexes (see Figure 28(d)). The first value of that range will be the index of the last article previously read by the node from that server (the node must of course remember the value), or the index of the first available article, as returned by the `GROUP` reply, it is the first time the node connects to that server. The second value will be the index of the last available article, as indicated by the `GROUP` reply. This way, the node is ensured not to read several times the same article, in the case it joins the broadcast channel several times in a row. Once the node has received the overview of the group, it can filter out the articles which are irrelevant to it (i.e., whose subject is not "p2padvertisement") and prepare to receive the relevant ones.

However, the newsgroup that has been chosen to store the advertisements is not dedicated to this purpose. In fact, `alt.test` has an average traffic of 424 articles a day (computed from a snapshot taken January 19th, 2005), which will add overhead to the reply of the `XOVER` command. Since the simulator does not take time into consideration at all, it is impossible to evaluate the amount of that overhead. Therefore, it will simply be ignored, even though the nodes querying the overview database will have an impact in terms of network traffic on the server.

The total traffic T_j generated by a node connecting to the broadcast channel depends on R, which is the number of articles requested by the `XOVER` command, is therefore:

$$T_j = 327 + 214R \tag{4}$$

5.2.2 Retrieving an Advertisement

The retrieval of an advertisement is described in Figure 10(c). The node must decide, based on the content of the overview database it received, what article it will request. Thereafter, the article can be retrieved with the `BODY` command, which returns the index of the article, its *Message-Id* and the actual body of the article. Headers are not useful for the node, since all the needed information is located in the body of the article; retrieving only the body is therefore sufficient.

The total amount of traffic T_r that is generated by one node retrieving one article is therefore:

$$T_r = 244 \tag{5}$$

5.2.3 Sending an Advertisement

Sending an advertisement to the newsgroup is done as follows (see also Figure 10(d)): the client sends a `POST` message (Figure 27(e)), and the server replies with a confirmation message and a recommended *Message-Id* value (Figure 28(f)). If the client does not have posting permission, a different message is sent, telling the client that posting is not allowed; in the latter case, the client must abort the posting procedure. We assume here that the node is allowed to post. After receiving the confirmation message, the client sends the article composed of 3 mandatory headers (*Subject*, *From* and *Newsgroups*) and containing the advertisement in its body (see Figure 27(f)). The server then replies with a confirmation message that the article has been posted (Figure 28(g)). If the message is not valid, an error message is replied instead of the confirmation.

The amount of traffic T_s generated by the sending of an advertisement is:

$$T_s = 376 \tag{6}$$

5.2.4 Leaving the Broadcast Channel

When the node decides do disconnect from the broadcast channel (see Figure 10(b), it sends a `QUIT` command (Figure 27(g)), to which the server sends a confirmation message (Figure 28(h)).

The amount of traffic T_l generated by the leaving process is:

$$T_l = 13 \tag{7}$$

6 NODE BEHAVIORS

This chapter describes in more detail the different actions that a node can perform while it is run by the simulator, as was described in Section 4.2.2.

From a programming point of view, each node maintains a list of neighbors (C_{N_0}) and two queues of known nodes (whose union forms K_{N_0}):

- The *P2P queue* contains the known nodes that the node has learned about directly from neighbors: when a connection is established between two nodes, they both exchange a list of their neighbors (as described in Section 6.1.1). These lists are then used to populate the P2P queue of each node respectively.
- The *channel queue* contains the known nodes that the node has learned about from the broadcast channel: the purpose of the broadcast channel is to get to know other nodes. Other nodes that are advertised on the broadcast channel are used to populate the channel queue.

6.1 Common Behaviors

The following behaviors are the same whatever the type of the broadcast channel.

6.1.1 Neighbors Lists Exchange

Once a new connection has been established between two nodes, they exchange the lists of their respective neighbors. Since one of the goals of this work is to

find a way to minimize the use of the broadcast channel, this behavior, which is a common practice in fully distributed P2P file sharing systems such as Gnutella, allows to get to know more nodes without using the broadcast channel. Its implementation in a P2P protocol could be done in two ways:

- After the connection has been established, each node sends the list of its neighbors to the neighbor using a specific protocol command.
- The list of neighbors are sent as a part of the connection request and connection acknowledgment commands; this could increase the load of the network if connections are often rejected.

The simulator does not implement any specific P2P protocol; however, when a connection is established, the list of neighbors are copied from the node to its neighbor's P2P queue and vice versa.

6.1.2 Connection Requests

Connection requests are always sent when there is a possibility to do so. It is assumed that the nodes are eager to aggregate themselves into a network and to have as many neighbors as possible. The only case when a node will not send a connection request to another node is when it holds enough neighbors, i.e., when its degree has reached a given threshold (which may be equal to or lower than its maximum degree).

6.1.3 Incoming Connections

Incoming connection requests are accepted based on Algorithm 1. If the node cannot afford to get one more neighbor, i.e., its degree is equal to its maximum degree, then the node will not accept any incoming connection. Moreover, in order to prevent the creation of a partitioned network (that is, a network composed of two or more distinct connected components), the node will reject a connection request issued by another node which can accept only one more neighbor (i.e., the other node's degree is equal to its maximum degree minus 1) if the node itself can also accept only one more neighbor. For that purpose, a hypothetical P2P protocol following these requirements would require that the connection request message contains a flag stating that requester can accept only one more neighbor.

6.1.4 Joining the Broadcast Channel

The broadcast channel will be joined based on Algorithm 2. The node will join the broadcast channel if it still wants to connect to other nodes, but does not know any other nodes (because e.g., all the known nodes it knew about have rejected its connection requests); by doing so, it may get to know other nodes that want to establish new connections. However, if the broadcast channel has reached its full capacity, the node cannot join it.

Input: N_0 the node that receives the connection request
N_1 the node that has sent the connection request
Output: accept or reject the connection
if $d(N_0) = d_{max}(N_0)$ **then**
reject the connection
else if $d(N_0) = d_{max}(N_0) - 1$ and $d(N_1) = d_{max}(N_1) - 1$ **then**
reject the connection
else
accept the connection
end if

ALGORITHM 1 Incoming request acceptance

Input: N_0 the node making the decision
B the broadcast channel
J the set of connections to the broadcast channel
threshold the minimum desired neighbor filling
Output: join or do not join the broadcast channel
if N_0 is connected to B **then**
do not join
else if $d(N_0) = d_{max}(N_0)$ **then**
do not join
else if $|J| = |J|_{max}$ **then**
do no join
else if $\left|K_{N_0}\right| = \emptyset$ and $d(N_0)/d_{max}(N_0) <= threshold$ **then**
join
else
do not join
end if

ALGORITHM 2 Channel joining decision

6.1.5 Leaving the Broadcast Channel

Leaving the broadcast channel is done based on Algorithm 3. The node will leave the broadcast channel when it has come to know enough other nodes, i.e., the number of nodes it knows is equal to or greater than a given threshold. Besides, in order to prevent any node staying indefinitely on the broadcast channel, the node will leave the broadcast channel if it has been connected to it for a number of simulation cycles equal to or greater than a given threshold.

Input: N_0 the node making the decision
B the broadcast channel
$filling$ the minimum desired neighbor filling
$maxTime$ the maximum time a node can remain connected to B
Output: leave or do not leave the broadcast channel
if N_0 is not connected to B **then**
do not leave
else if $|K_{N_0}| = |K_{N_0}|_{max}$ **then**
leave
else if $d(N_0)/d_{max}(N_0) > filling$ **then**
leave
else if Number of simulation cycles N_0 has been connected to $B \geq maxTime$ **then**
leave
else
do not leave
end if

ALGORITHM 3 Channel leaving decision

6.1.6 Connection Target Selection

When attempting to establish new connections, Algorithm 4 applies. For the purpose of selecting the target of a connection request, one defines two subsets of K_{N_0}:

- $K_{N_0}^{channel}$ is the set of known nodes that N_0 got to know about through the broadcast channel.
- $K_{N_0}^{P2P}$ is the set of known nodes that N_0 got to know about through the exchange of lists of neighbors, as described in 6.1.1.

From a programming point of view, the two sets described above are implemented as queues, respectively called the *channel queue* and the *P2P queue*. The discrimination between the two possible origins of the known nodes exists because the simulator gathers statistical information about how many connections between nodes have been established using the broadcast channel and how many from the peer-to-peer exchange of neighbor list.

Because of the implementation of the simulator, the two queues are however not equal: the P2P queue has a higher priority than the channel queue, meaning that when the node will look for nodes to which to connect, it will first consider the P2P queue, and only when the P2P queue is empty, it will consider the channel queue.

Input: N_0 the acting node
Output: nothing
if $K_{N_0}^{P2P} \neq \emptyset$ **then**
  while $K_{N_0}^{P2P} \neq \emptyset$ and no connection has been established **do**
    $N \leftarrow$ dequeue the first other node in the channel queue
    if $N \in C_{N_0}$ **then**
      loop again
    else
      try to establish a connection to N
    end if
  end while
else if $K_{N_0}^{channel} \neq \emptyset$ **then**
  while $K_{N_0}^{channel} \neq \emptyset$ and no connection has been established **do**
    $N \leftarrow$ dequeue the first other node of the channel queue
    if $N \in C_{N_0}$ **then**
      loop again
    else
      try to establish a connection to N
    end if
  end while
end if

ALGORITHM 4 Connection target selection

6.2 Specific Behaviors

The following behaviors differ depending on the type of broadcast channel.

6.2.1 IRC as a Broadcast Channel

Just after joining the channel, the node advertises itself to all other nodes on the channel. This is equivalent to sending a PRIVMSG (see Section 4.1.1) to the IRC channel which other nodes are listening to. This self-advertisement is made only once after joining the IRC channel. The node then waits for the unknown nodes present on the channel to request a connection to the node. The node is thus not discovering unknown nodes, but is being discovered by them.

6.2.2 Usenet as a Broadcast Channel

Just after joining the channel, the node retrieves advertisements of unknown nodes. If not enough advertisements have been found, the node posts an advertisement of itself to the channel, in order to be discovered by unknown nodes. Advertisements are posted based on Algorithm 5.

Input: N_0 the node making the decision
$limit$ minimum known nodes for leaving
Output: post or do not post an advertisement
if the previous advertisement of N_0 has not been expired yet **then**
do not post
else if $\left|K_{N_0}\right| < limit$ **then**
post
else if $d(N_0) = d_{max}(N_0)$ **then**
do not post
else if $K_{N_0} = \emptyset$ **then**
post
else
do not post
end if

ALGORITHM 5 Posting decision

7 EXPERIMENTAL RESULTS

7.1 Goals of the Simulations

The goal of this work is to find optimum or near-optimum values of the various parameters of the simulation in order to prevent the creation of disjoint connected components in the network and at the same time minimize the strain on the broadcast channel. As simulation results will show it, these two goals are opposed, therefore a trade-off between these two goals needs to be found.

7.1.1 Disjoint Networks

Disjoint connected components are unavoidable, but two different cases must be considered.

- Several disjoint connected components with one of them of a much larger size than the others: this is not a bad case since in a real system, the nodes in the smaller connected components will soon notice there are not enough other nodes to transact with and will then join the broadcast channel to find more connections or disconnect from their current neighbors and attempt to create new connections with the largest connected component; this kind of behavior is however not implemented in the simulator, since it requires the node to actually transact, meaning that a real P2P protocol has to be implemented in the node.

  Algorithm 1 (page 59) prevents the formation of small connected components by forbidding two nodes that can establish each only one connection

(i.e., the node's degrees are equal to their maximum degree minus one) to connect to each other, thus both giving up on trying to establish new connections. This behavior especially prevents the creation of connected components made of nodes having each a maximum degree of two and connected to each other as a ring, or made of two nodes having a maximum degree of one. In practice, one can however see in simulation results disjoint connected components where some nodes still could try to establish new connections, but are prevented from doing so by the value of the *minimum desired neighbor filling* being lower than their actual degree over maximum degree ratio. Optimizing the value of this parameter will help preventing the creation of more than one connected component.

- Several disjoint connected components of approximately the same size: this is the worst case scenario, since the connected component is sufficiently large for all the nodes in it to find enough neighbors without the need to join the broadcast channel for more, thus eliminating all possibility to establish a bridge between the connected components.

7.1.2 Usage of the Broadcast Channel

The broadcast channel, be it IRC or Usenet, is a public infrastructure of the Internet that has not been designed with advertisement of P2P networks in mind. It is obvious that the more nodes will use the system to join a network, the more the broadcast channel will be used. Each step of the communication with the broadcast channel (join, leave, send or receive an advertisement) will generate network traffic.

In the case of IRC being used as the broadcast channel, the goals are:

- To minimize the number of times a node joins the channel, since each joining generates traffic, not only because of the joining step, but also because of the leaving step which needs to happen at some point.

- To minimize the time a node spends on the channel, since it will automatically receive the advertisements that newcomers to the channel will send.

- To minimize the number of nodes present on the channel at one given time, because the more nodes are on the channel, the more memory is required from the IRC server to store information about the channel members, and the more network traffic is generated when a node joins, leaves or sends an advertisement.

In the case of Usenet being used as the broadcast channel, the goals are:

- To minimize the number of times a node joins the channel, for the same reasons as the ones evoked above about IRC.

- To minimize the number of advertisements the node retrieves from the Usenet server. This can be done in two ways:

1. Prevent the node from retrieving advertisements that it has already retrieved before and which are still stored on the server.
2. Set the expiration date of the messages so as to prevent advertisements from being stored there too long (older advertisements have a higher probability of being useless, since the nodes who posted them have probably already established connections to other nodes).

7.2 Parameters of the Simulation

The simulator accepts twelve parameters, but only a few of them are the real unknowns of the problem: *Minimum desired neighbor filling*, *Minimum known nodes count for leaving*, *Maximum channel staying duration* and *Channel capacity*; in the Usenet-based simulator, the *Advertisement expiration age* is an additional unknown.

The following parameters are fixed for all simulations:

Max neighbor distribution is set to *powerlaw*, since this is the actual connection distribution of the Gnutella network.

Activation probability is set to 1.0.

Incoming connection acceptance probability is set to 100% since there is no reason why a node should reject an incoming connection (other than the reasons evoked in Section 6.1.3, *Incoming Connections*).

Connection decision probability is set to 100% since the nodes are meant to be eager to establish new connections; there is thus no reason why a node should decide not to connect to another node (other than the reasons invoked in Section 6.1.2, *Connection Requests*) when it has the possibility to do so.

Channel join probability is set to 100% since there is no reason why a node should decide not to join the broadcast channel when it needs to do so (other than the reasons evoked in Section 6.1.4, *Joining the Broadcast Channel*).

Channel leave probability is set to 100% since the node should not stay longer on the channel than it is absolutely necessary, as defined in Section 6.1.5, *Leaving the Broadcast Channel*.

The *Number of nodes* is not a real unknown, but the search for the optimum values of the above mentioned unknowns may depend on it. Thus the simulations will be run several times, with different numbers of nodes. The maximum value of this number is however limited by the amount of memory available in the computer running the simulation, and by the time taken by the simulations to complete.

7.3 Processing of the Results

Unless otherwise stated, all the simulations mentioned below have been run ten times with the same parameters, but with different values of the seed of the random number generator. The values chosen for the seed are the numbers 1 to 10, since these are as good as any other.

As mentioned in 4.2.3, once all the individual simulations have been run (i.e., with different values of the same parameters), a first script called *efficiency* extracts the interesting values from the result files of each individual simulation and outputs a summary of the simulation:

- The clustering efficiency index, which evaluates how well the nodes have clustered during the simulation. The formula is

$$\eta = \frac{\sum_{\mathcal{C}_i \subset \mathcal{G}} |\mathcal{C}_i|^2}{\max_{\mathcal{C}_i \subset \mathcal{G}} |\mathcal{C}_i|^2} \tag{8}$$

  where $\mathcal{G}$ is the network and $\mathcal{C}_i \subset \mathcal{G}$ are the disjoint connected components.

  The use of squares emphasizes the difference between the sizes of the connected components, and the division by the size of the largest component restricts η to $[1, n+1]$[1], where n is the number of disjoint connected components. $\eta = 1$ means that there is only one single connected component, and when η gets close to $n+1$, there are multiple disjoint components which have more or less even sizes. The goal is to have $\eta = 1$, or at least as close to 1 as possible.

- The number of disjoint connected components
- The number of times the channel has been joined (in the IRC simulator)
- The average traffic per node (i.e., the sum of all traffic generated during the simulation divided by the number of nodes)

A second script called *average* calculates the average and the standard deviation of each of the above-mentioned values over the ten values of the seed, in order to minimize the effect of randomness on the results.

A third and final script (*combine*) combines the values of the average traffic per node and the clustering efficiency index according to the following formula:

$$E(t,c) = \begin{cases} t+c, & \text{if } c > 1+\epsilon \\ t, & \text{if } c \leq 1+\epsilon \end{cases}, \tag{9}$$

where t is the normalized traffic per node and c is the clustering efficiency index. The value of ϵ must be chosen so that, as explained in Section 7.1.1, only

[1] The actual range is assumed to be $[1, n]$, but it is not trivial to prove, whereas the previous one is. Moreover, the demonstration would be irrelevant here.

very small secondary connected components (SCC) are allowed to exist. A value of $\epsilon = 10^{-4}$ would allow e.g., one SCC of approximately 1% of the size of the giant connected component (GCC) to exist, or two small SCC of approximately 0.7% of the size of the GCC, assuming that the size of the GCC is much larger than the one of the SCC.

The formula for $E(t, c)$ reflects the fact that the clustering efficiency is binary and prevails over the traffic: if the clustering efficiency is not good enough, the set of parameters that led to this result are discarded, with no regard to the traffic, otherwise, only the traffic is considered when evaluating the fitness of the parameters. Therefore, when $E(t, c) \geq 1$ the result is considered as "bad" and when $E(t, c) < 1$ it is considered as "good".

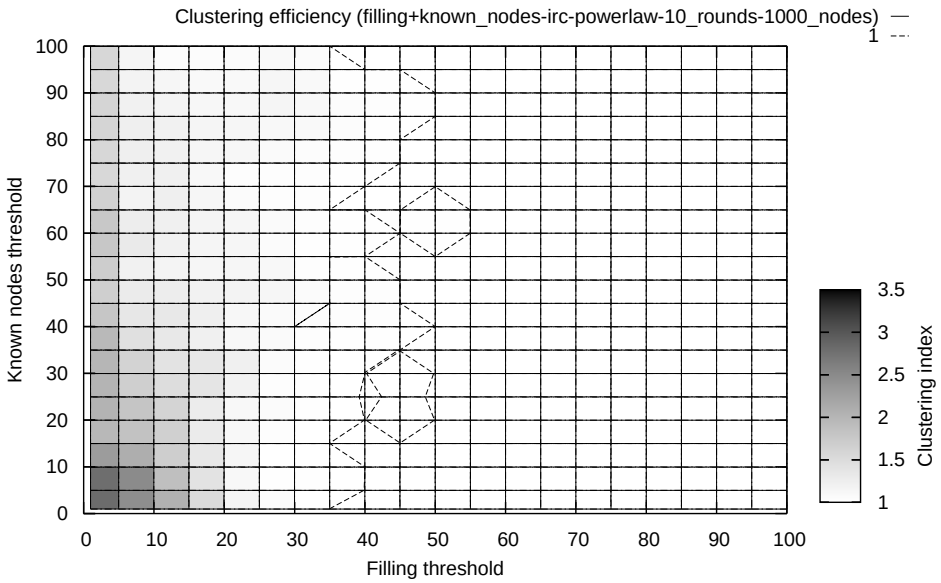
7.4 IRC-based Simulation Results

7.4.1 Simulations of 1000 Nodes

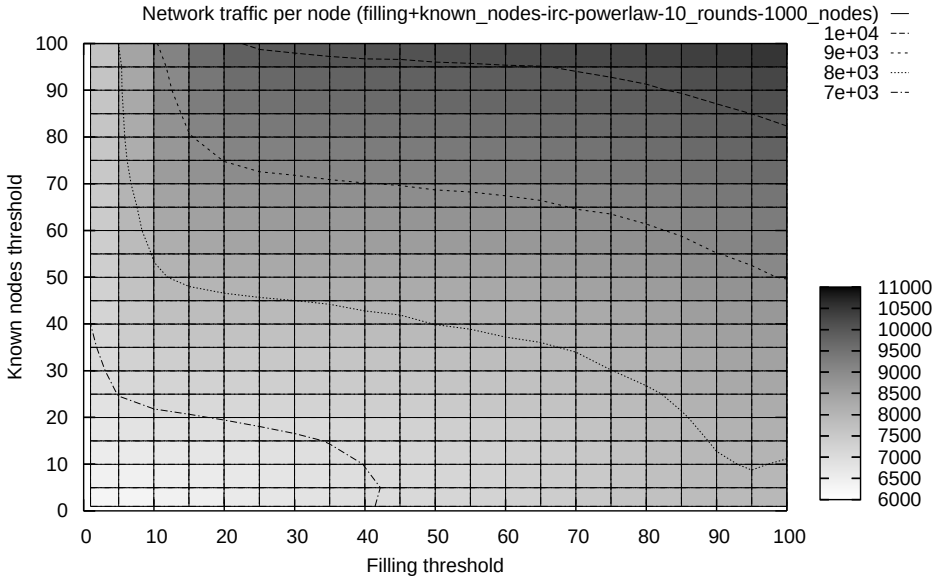
At first, simulations using IRC as a broadcast channel have been run with 1000 nodes and with the value of the *minimum desired neighbor filling* equal to 1, 5, 10, ..., 95, 100 and the value of the *minimum known nodes count for leaving* equal to 1, 5, 10, ..., 95, 100 (as mentioned in Section 4.2, these values are actually percentages of the maximum degree of the nodes). Several statistical data have been gathered from the results of the various simulations and are shown in Figures 11 and 12.

Figure 11(a) shows that small values of the parameters result in the formation of several disjoint components. Simulation results show (for illustration purposes) that values of 1 for both parameters and a value of 10 for the seed of the random number generator yields twenty connected components with sizes ranging from 5 to 215, and a median size of 35. The value of the *minimum desired neighbor filling* has more influence on the clustering process than the value of the *minimum known nodes count for leaving*: whatever the value of the *minimum known nodes count for leaving*, the clustering coefficient is above 1.0001 for values of the *minimum desired neighbor filling* lower than 30. Moreover, for values of the *minimum desired neighbor filling* above 55, the clustering coefficient is always equal to or lower than 1.0001. The band situated between the values 30 and 55 of this parameter is rather chaotic, and the value of the clustering coefficient there depends greatly on the seed of the random number generator.

The results can be explained by the fact that the closer the degree of the node is to its maximum degree (i.e., the "filling level" is close to the maximum), the more tightly the network is connected and the less possibilities there is for disconnected components to survive. Moreover, the relatively small influence of the number of known nodes on the clustering can be explained by the fact that a node gathers knowledge of other nodes mostly by exchanging lists of neighbors with its own neighbors (in order to minimize the usage of the broadcast channel). Therefore, most of the known nodes are already part of the same con-

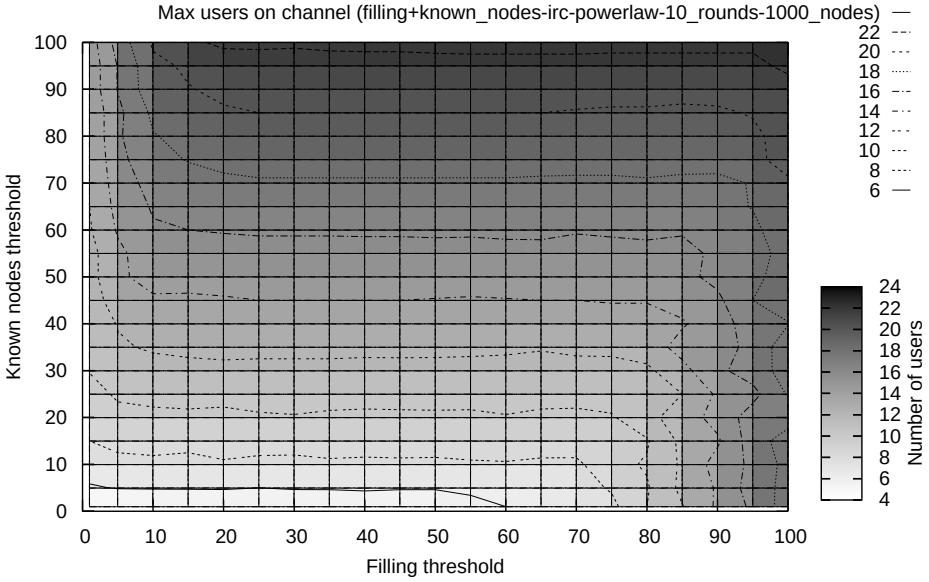


(a) Clustering efficiency as a function of the minimum desired neighbor filling and the minimum known nodes count for leaving. The area at the right of the contour line has a clustering coefficient equal to or lower than 1.0001

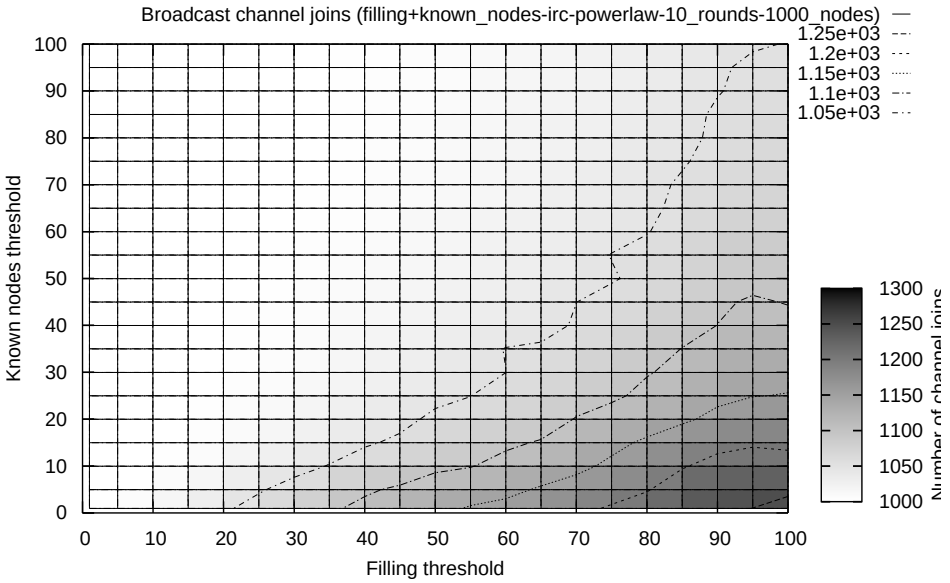


(b) Network traffic per node (in bytes) as a function of the minimum desired neighbor filling and the minimum known nodes count for leaving

FIGURE 11 Results of the IRC-based simulations



(a) Maximum number of users on the broadcast channel



(b) Number of joins to the broadcast channel

FIGURE 12 Results of the IRC-based simulations (continued)

nected component as the one the node belongs to, meaning that the connections between two disjoint connected components are not favored by the increase of the *minimum known nodes count for leaving*. Very low values of this parameter favor the creation of several disjoint connected components, but that effect diminishes quickly when the value increases.

Figure 11(b) shows that the network traffic per node (in- and out-going) is between 6000 and 11000 bytes for the duration of the simulation (since the simulation does not refer to time in any way, it is not possible to calculate the data rate in bytes/s). The traffic depends almost equally on the values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving*. This can be explained by the fact that when these two values are high, the node will join the broadcast channel more often, leading to an increase in the traffic.

Figure 12(a) shows that the maximum number of users connected to the broadcast channel during one simulation varies between 4 and 24 and depends mostly on the *minimum known nodes count for leaving*, with a minimum for lower values of the parameters. This is consistent with the fact that when the node tries to get to know more other nodes it stays longer on the channel; the nodes then start to accumulate on the channel.

Figure 12(b) shows that the number of joins to the broadcast channel during one simulation increases when the *minimum desired neighbor filling* increases and the *minimum known nodes count for leaving* decreases. This is consistent with the fact that the longer the node remains on the channel (i.e., with a high value of *minimum known nodes count for leaving*), the more nodes it gets to know and the higher the possibilities to establish new connections, and that the more neighbors it wants to be connected to (i.e., a high value of the *minimum desired neighbor filling*), the more nodes it needs to get to know, either by joining the broadcast channel more often, or by staying longer on the channel.

Figure 13 represents a combination of the normalized value of the traffic and of the clustering coefficient, according to the formula presented in Section 7.3. Values above 1 on the map mean that the simulation led to a situation with a clustering coefficient above 1.0001[2]. Conforming to what was described above, lower values of the *minimum desired neighbor filling* lead to the formation of disconnected components in the network, but also of lower traffic. The optimum is therefore in an uncertain region, where the two constraints are opposing each-other and whose location may be changing with different parameters of the simulation (like the number of nodes or the seed of the random number generator).

One can read from the map that the optimum is reached for values of *minimum desired neighbor filling* between 30 and 35 and for values of the *minimum known nodes count for leaving* between 1 and 5. The choice of the optimal values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving* is however not straightforward: the frontier on the map between the areas where disconnected components are created and the area where they can be ignored is uncertain and depends on a random value.

[2]It could also mean that the traffic is so important that its normalized value is above 0.9999; this however can happen in practice only with values of both parameters close to 100.

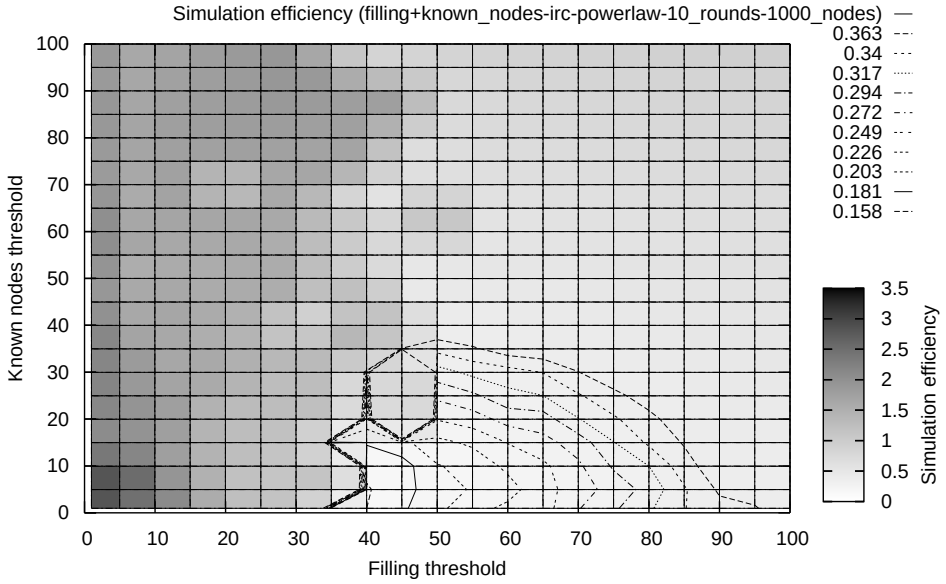


FIGURE 13 Efficiency of the IRC-based simulations

7.4.2 Influence of the Number of Nodes

Figure 14 shows the average clustering efficiency for 1000 to 10000 nodes by step of 1000 nodes. The gray level represents the standard deviation divided by the mean, which is close to zero for the values on the right side of the curve. This indicates that the clustering efficiency varies very little when the number of nodes varies. It moreover remains satisfactory (i.e., below the threshold of 1.0001) for the values of the *minimum desired neighbor filling* above 35 and low values of *minimum known nodes count for leaving*, which were found to be close-to-optimum in Section 7.4.1.

A study (see Appendix 5.1) of the variation of the traffic per node as a function of the number of nodes ($T(N)$) shows that with a value of the *minimum desired neighbor filling* of 1, the function is of the form $T(N) = a_{i,j} + b_{i,j} \log N$ (with i taking the chosen values of the *minimum desired neighbor filling* and j taking the values of *minimum known nodes count for leaving*), and the correlation coefficient between T and $\log N$ is above 0.97. When the value of the *minimum desired neighbor filling* increases, the correlation coefficient diminishes. Plotting $T(N)$ for given values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving* shows that the traffic cannot be characterized anymore as a $\log N$ function and the plot resembles more a constant function with noise (the value of the correlation coefficient is not anymore meaningful since it characterizes only the noise). In the latter case, the relative noise, measured as the ratio of the standard deviation of the traffic values over the average traffic values, is below 0.5%. The traffic per node can then be considered as independent of the number of nodes.

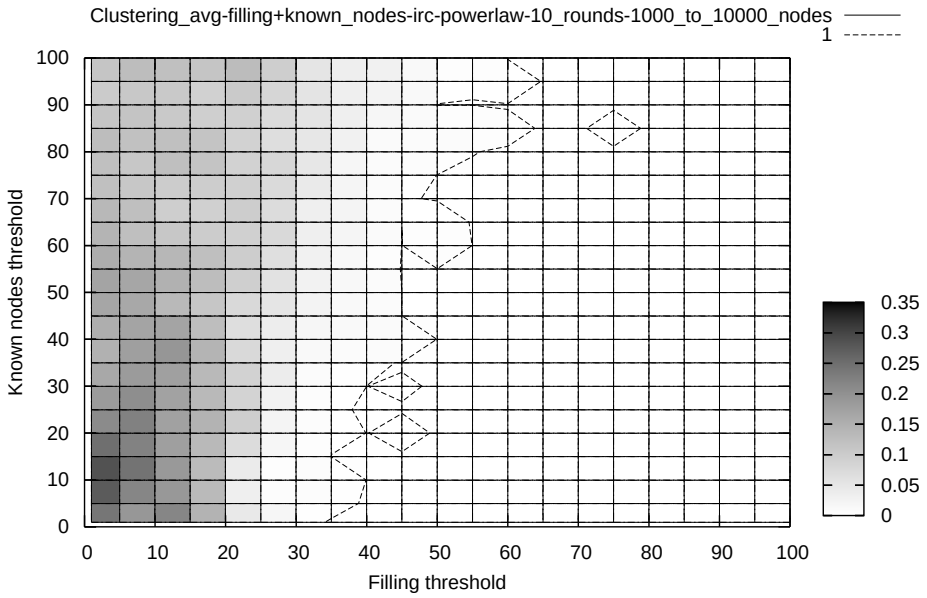


FIGURE 14 Average clustering efficiency of the IRC-based simulations for 1000 to 10000 nodes by step of 1000 nodes. The Gray level represents the standard deviation divided by the mean. The area on the right side of the contour line has an average clustering ratio lower than or equal to 1.0001

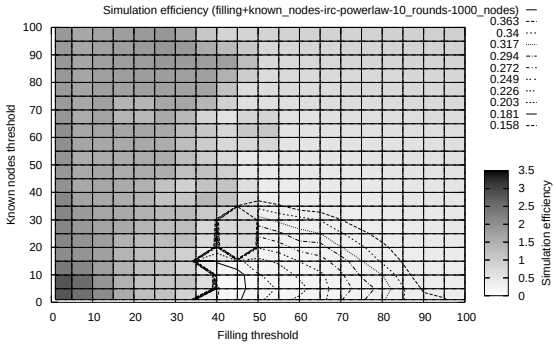
Figure 15 shows that while the number of nodes increases, the location of the optimum remains approximately on the same spot. Because of the use of random numbers in the simulations, the frontier between acceptable and non-acceptable clustering coefficients varies, but the graphs show that the variation is small.

One can however estimate with good confidence that a value of the *minimum desired neighbor filling* of 35 and a value of the *minimum known nodes count for leaving* between 1 and 5 is as close to the optimal as one can dare to go without risking to have too high a clustering coefficient.

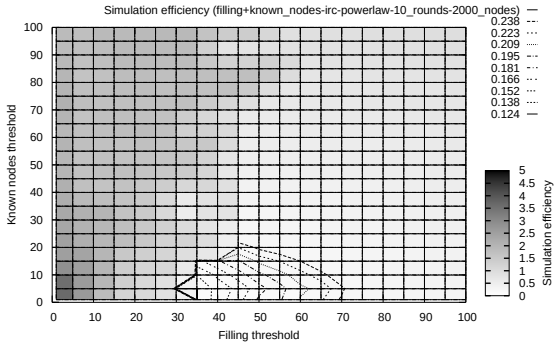
7.4.3 Network Characterization

The network can be characterized with the three values described in Section 4.2.3. The following parameters have been used for the simulations:

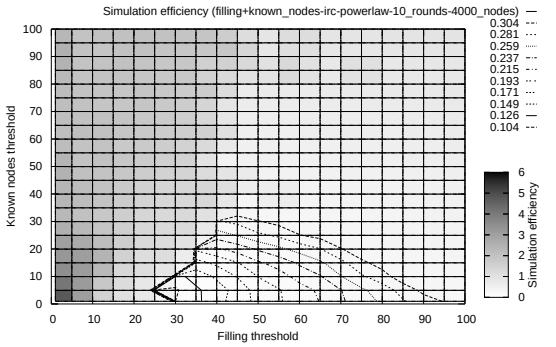
- the number of nodes is 1000,
- the *minimum desired neighbor filling* is 35,
- the *minimum known nodes count for leaving* is 5,
- the values of the seed of the random number generator have been chosen so that the resulting network is made of one single connected component. The values are 3, 5, 6, 7, 8 and 10.



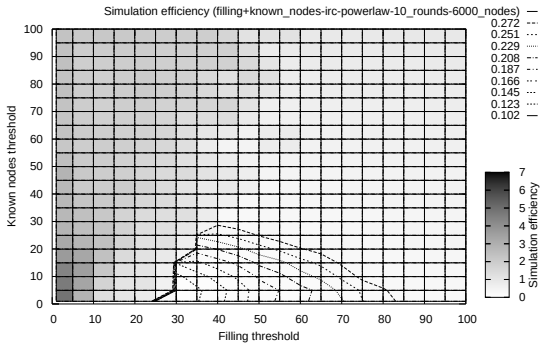
(a) 1000 nodes



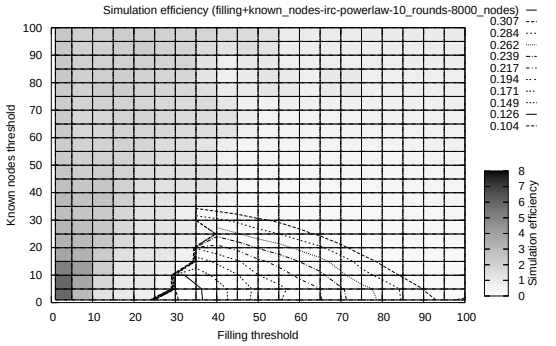
(b) 2000 nodes



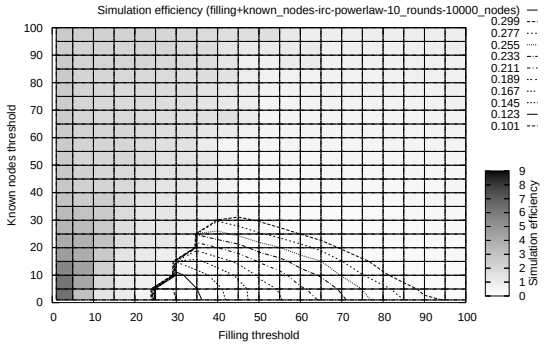
(c) 4000 nodes



(d) 6000 nodes



(e) 8000 nodes



(f) 10000 nodes

FIGURE 15 Efficiency of the IRC-based simulations for increasing numbers of nodes

The *clustering coefficient* of the network is between 0.576 and 0.590, with a mean of 0.582 and a median of 0.582. The *longest shortest path* is between 9 and 21, with a mean of 12.7 and a median of 11. The *average shortest path* is between 3.83 and 7.82, with a mean of 5.00 and a median of 4.37.

7.5 Usenet-based Simulation Results

7.5.1 Simulations of 1000 Nodes

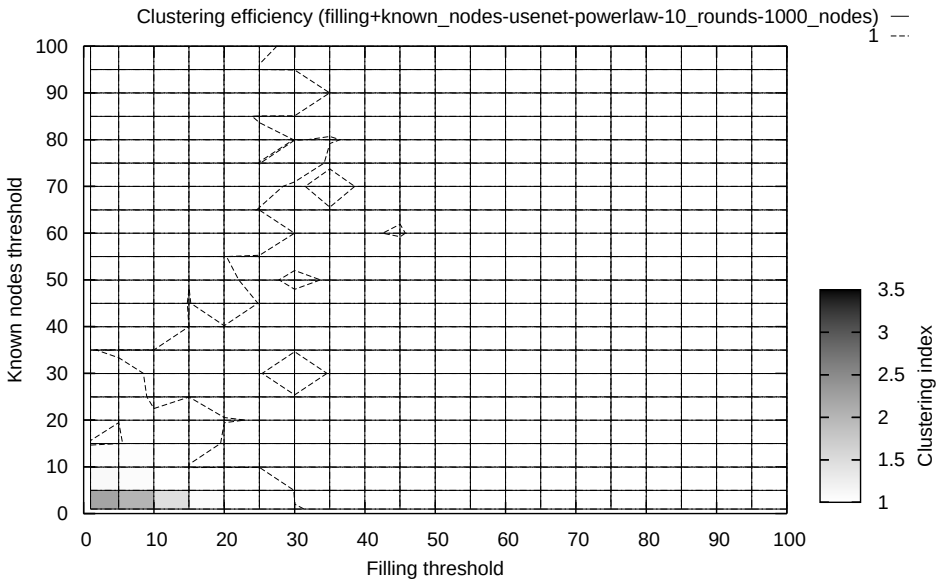
At first, simulations using Usenet as a broadcast channel have been run with 1000 nodes and with the value of the *minimum desired neighbor filling* equal to 1, 5, 10, ..., 95, 100 and the value of the *minimum known nodes count for leaving* equal to 1, 5, 10, ..., 95, 100 (as mentioned in Section 4.2, these values are actually percentages of the maximum degree of the nodes). Several statistical data have been gathered from the results of the various simulations and are shown in Figure 16.

Figure 16(a) shows that small values of the parameters result in the formation of several disjoint components. Simulation results show (for illustration purposes) that values of 1 for both parameters and a value of 10 for the seed of the random number generator yields thirty connected components with sizes ranging from 2 to 71, and a median size of 12. The value of the *minimum desired neighbor filling* has more influence on the clustering process than the value of the *minimum known nodes count for leaving*: most values of the *minimum known nodes count for leaving* (with the notable exception of 35; this can be attributed to randomness in the simulations) lead to a clustering coefficient above 1.0001 for values of the *minimum desired neighbor filling* lower than 35. Moreover, for values of the *minimum desired neighbor filling* above 50, the clustering coefficient is always equal to or lower than 1.0001.

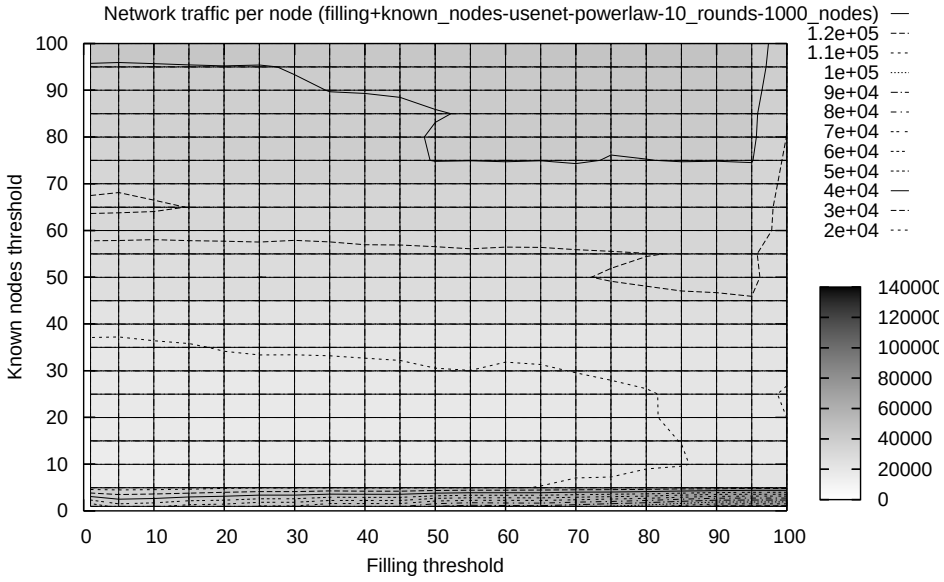
Like in the case of the IRC-based simulations, the results can be explained by the fact that the closer the degree of the node is to its maximum degree (i.e., the "filling level" is close to the maximum), the more tightly the network is connected and the less possibilities there are for disconnected components to survive.

Figure 16(b) shows that the network traffic per node (in- and out-going) is between 20000 and 140000 bytes for the duration of the simulation (since the simulation does not refer to time in any way, it is not possible to calculate the data rate in bytes/s). The traffic depends mainly on the values of the *minimum known nodes count for leaving*. With a value of 1 for this parameter, the traffic reaches its maximum, then reaches its minimum with a value of 5 of the same parameter, and increases up to about 50000 bytes when the parameter increases to 100. The influence of the *minimum desired neighbor filling* parameter is less important: when that parameter increases, the traffic increases as well.

The influence of the *minimum known nodes count for leaving* can be explained by the fact that the parameter is a percentage of the maximum degree of the node: since the upper limit of this degree is 99, the actual minimum known nodes count



(a) Clustering efficiency as a function of the minimum desired neighbor filling and the minimum known nodes count for leaving. The area at the right of the contour line has a clustering coefficient equal to or lower than 1.0001



(b) Network traffic per node (in bytes) as a function of the minimum desired neighbor filling and the minimum known nodes count for leaving

FIGURE 16 Results of the Usenet-based simulations

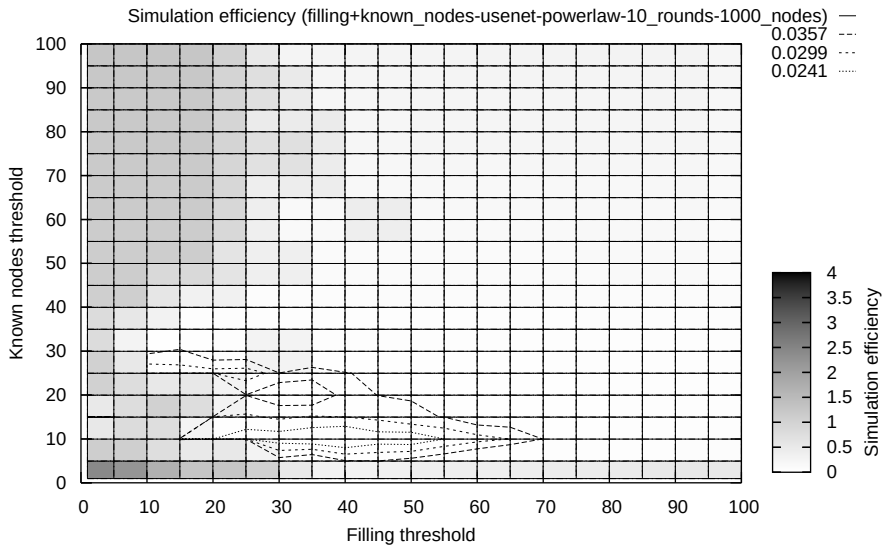


FIGURE 17 Efficiency of the Usenet-based simulations

for leaving for each node is 1 node (in this particular case, we consider the actual number of nodes, not the percentage of the maximum degree), even for the nodes which have a high value for their maximum degree. The nodes will then connect to the channel every time they want to discover more other nodes. Each connection generates much traffic, in addition to the traffic necessary for receiving the advertisements, which explains why traffic is high when the value of the *minimum known nodes count for leaving* is 1, and drops when its value is 5. When the *minimum known nodes count for leaving* increases toward 100, the increase in the traffic is explained by the fact that the node receives many advertisements on each connection.

Figure 17 represents a combination of the normalized value of the traffic and of the clustering coefficient, according to the formula presented in Section 7.3. Values above 1 on the map mean that the simulation led to a situation with a clustering coefficient above 1.0001[3]. Conforming to what was described above, lower values of the *minimum desired neighbor filling* lead to the formation of disconnected components in the network, but also of lower traffic. The optimum is therefore in an uncertain region, where the two constraints are opposing each-other and whose location may be changing with different parameters of the simulation (like the number of nodes or the seed of the random number generator).

One can read from that map that the optimum[4] reached for values of *minimum desired neighbor filling* between 30 and 50 and for a value of the *minimum*

[3]It could also mean that the traffic is so important that its normalized value is above 0.9999; this however can happen in practice only with values of both parameters close to 100.

[4]The actual optimum is surrounded by values which are far from optimal; this "optimum" is considered as a simulation artifact and is therefore disregarded.

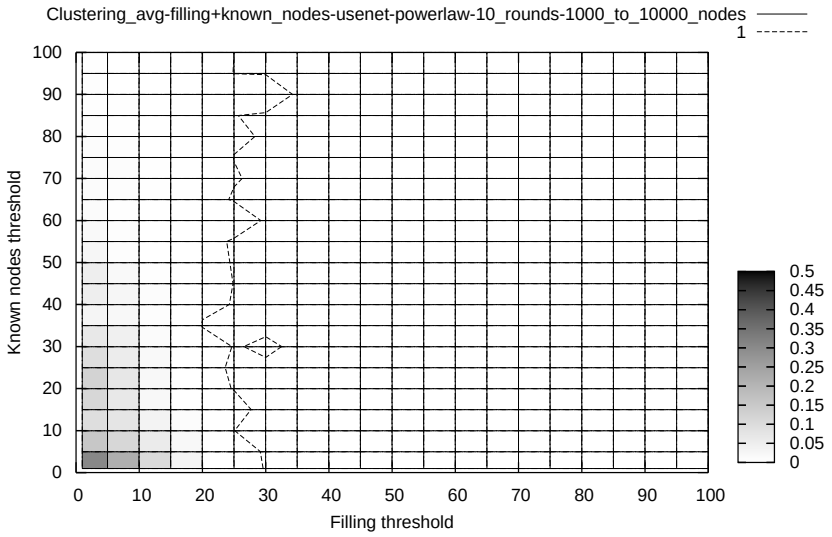


FIGURE 18 Average clustering efficiency in the Usenet-based simulations for 1000 to 10000 nodes by step of 1000 nodes. The gray level represents the standard deviation divided by the mean. The area on the right side of the contour line has an average clustering ratio lower than or equal to 1.0001

known nodes count for leaving of 10. The choice of the optimal values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving* is however not straightforward: the frontier on the map between the areas where disconnected components are created and the area where they can be ignored is uncertain and depends on a random value.

7.5.2 Influence of the Number of Nodes

Figure 18 shows the average clustering efficiency for 1000 to 10000 nodes by step of 1000 nodes. The gray level represents the standard deviation divided by the mean, which is close to zero for the values on the right side of the curve. This indicates that the clustering efficiency varies very little when the number of nodes varies. It moreover remains satisfactory (i.e., below the threshold of 1.0001) for the values of the *minimum desired neighbor filling* above 35 and low values of *minimum known nodes count for leaving*, which were found to be close-to-optimum in Section 7.5.1. One can therefore generalize these conclusions to greater numbers of nodes.

A study (see Appendix 5.2) of the variation of the traffic per node as a function of the number of nodes ($T(N)$) shows that the function is of the form $T(N) = a_{i,j} + b_{i,j}N$ (with i taking the chosen values of the *minimum desired neighbor filling* and j taking the values of *minimum known nodes count for leaving*), and the correlation coefficient between T and N is above 0.81, with a median value of 0.982. The correlation coefficient decreases when the values of both parameters increase.

Figure 19 shows that while the number of nodes increases, the location of the optimum remains approximately on the same spot. Because of the use of random numbers in the simulations, the frontier between acceptable and non-acceptable clustering coefficients varies, but the graphs show that the variation is small.

One can however estimate with good confidence that a value of the *minimum desired neighbor filling* of 35 to 40 and a value of the *minimum known nodes count for leaving* between 10 and 15 is as close to the optimum as one can dare to go without risking to have too high a clustering coefficient.

7.5.3 Influence of the Expiration Time

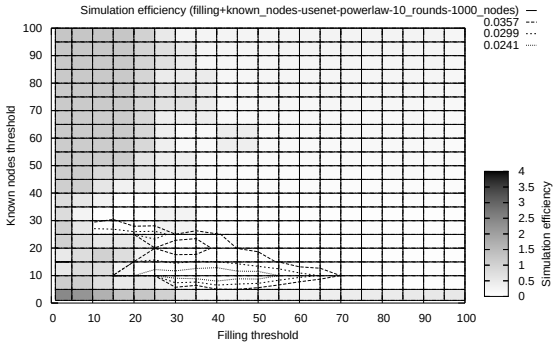
The Usenet messages may contain the optional header *Expires* [HA87] that sets the expiration date of the message. This allows to control the lifetime of the advertisement depending on the number of neighbors the node wants to gather. The simulation shows (see Figure 20) that optimal values for a and b (see Section 4.2 page 42 for the definition of the parameters) are in the intervals $[0.4, 0.5]$ and $[0, 10]$ respectively; the optimum is located at $(a, b) = (0.3, 5)$. These low values show that nodes find neighbors quite quickly, i.e., after only few iterations.

7.5.4 Network Characterization

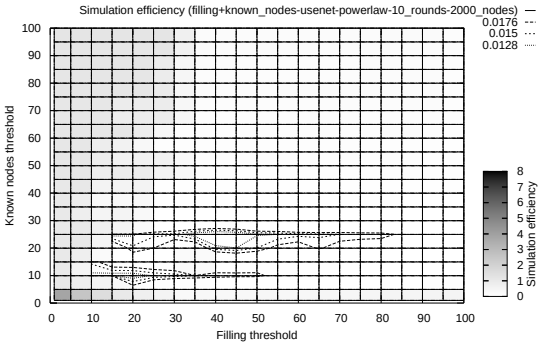
The network can be characterized with the three values described in Section 4.2.3. The following parameters have been used for the simulations:

- the number of nodes is 1000,
- the *minimum desired neighbor filling* is 35,
- the *minimum known nodes count for leaving* is 10,
- the a and b parameters of the expiration time are 0.3 and 5, respectively,
- the values of the seed of the random number generator have been chosen so that the resulting network is made of one single connected component. On the contrary to the case of the simulator using IRC, all values of the seed (from 1 to 10) give acceptable results.

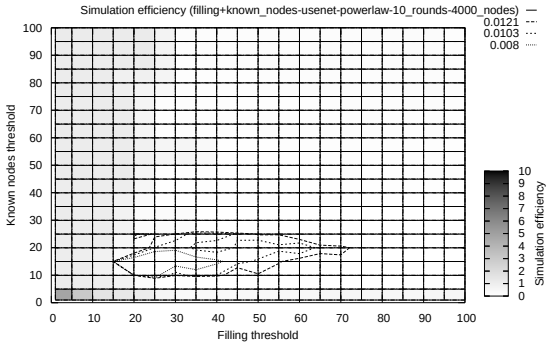
The *clustering coefficient* of the network is between 0.688 and 0.731, with a mean of 0.704 and a median of 0.695. The *longest shortest path* is between 7 and 10, with a mean of 8.6 and a median of 9. The *average shortest path* is between 3.81 and 4.39, with a mean of 4.00 and a median of 3.91.



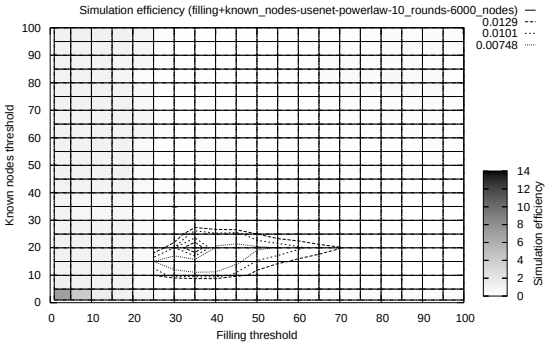
(a) 1000 nodes



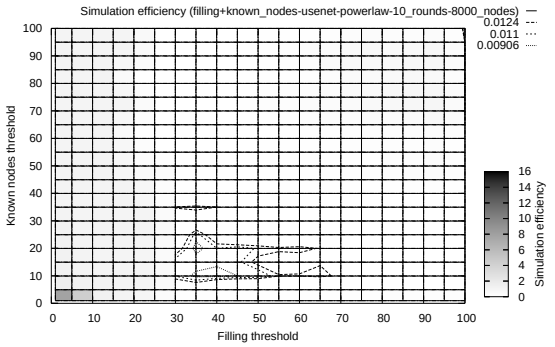
(b) 2000 nodes



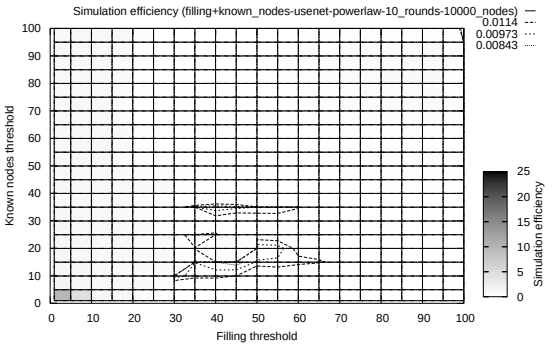
(c) 4000 nodes



(d) 6000 nodes



(e) 8000 nodes



(f) 10000 nodes

FIGURE 19 Efficiency of the Usenet-based simulations for increasing numbers of nodes

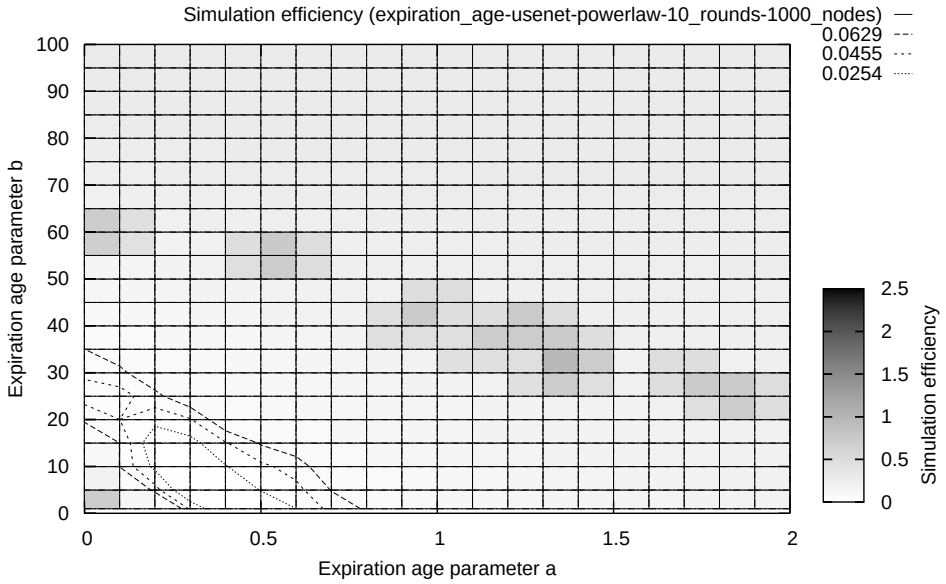


FIGURE 20 Efficiency of the Usenet-based simulations with variable expiration time (varying *a* and *b* parameters) and values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving* set to respectively 35 and 10

8 CONCLUSION

Peer-to-peer applications are more and more common nowadays on the Internet, mainly aimed at person-to-person file sharing or message exchange. Although the Internet, as a network, is by design a peer-to-peer system, most applications, including the most successful ones, have been designed with a client/server architecture. The usage of these traditional applications is constantly decreasing, while peer-to-peer applications are gaining ground.

Peer-to-peer applications connected to each other form an overlay network, having its own structure and which is often independent of the underlying network structure. Since peer-to-peer networks are in some sense autonomous, when an isolated node wants to join such a network, it needs to acquire specific technical information, called *binding information*, about a least one node, the *entry point* which is already part of the network. Connecting to the entry point will then make the node a part of the network. The problem of acquiring the binding information is usually considered as trivial and delegated to one single source of information. This single source of information is a weakness in the organization of the network, since if it is removed, it can prevent anyone from joining the network.

The goal of the work presented in this thesis was therefore to develop a fully distributed system providing the same service, which cannot therefore be removed, and which uses the existing infrastructure of the Internet. IRC and Usenet were chosen as such systems, since they are both widely deployed, well known, and fully distributed. Using World Wide Web search engines and random network scanning were also briefly considered.

A formal protocol for publishing binding information was described, allowing the user to transparently access the information through any of the media

described above. A formal notation was also defined, in order to formalize and shorten the descriptions of the behaviors of the nodes using the protocol.

A software simulator was built in order to simulate the behavior of the system, prove its validity and study its impact on the infrastructure it relies on (IRC or Usenet) in terms of bandwidth usage. The simulator considered and implemented only the behavior of the nodes on a functional level, not the detailed exchange of data that would have been required in the real system. These behaviors were also described in detail in the thesis. The simulator also assumed that the nodes were willing to discover as many other nodes as necessary in order to establish connections and thus to acquire as many neighbors as possible, up to a given limit.

In the system based on IRC, the nodes trying to join the network published their own binding information on a broadcast channel (an IRC channel with a predefined name in this case). The nodes which were already part of the network and were willing to establish more connections could then connect to the ones advertising themselves.

In the system based on Usenet, the nodes trying to join the network read Usenet articles from a given newsgroup, looking for binding information. If they succeeded, they would use the information in order to attempt to establish a connection, otherwise they would publish their own binding information in the hope that another node would attempt to connect to them.

The simulations were run with several parameters in order to determine an optimum set of values for the parameters. These parameters concerned (1) the number of neighbors a node needs to have in order to give up active attempts to acquire more neighbors, thus relying only on being found by others (*minimum desired neighbor filling*) and (2) the number of other nodes a node needs to discover (but not connect to) in order to give up using the broadcast channel (*minimum known nodes count for leaving*). In the case of the Usenet-based simulator, the expiration time of the articles was also considered (*expiration time*).

The simulations produced values that represented mainly the average traffic generated by the system and an index representing how efficiently the nodes had clustered, the goal being to prevent the creation of a partitioned network. These two results were then combined into a single efficiency index of the system. The same simulations were run for an increasing number of nodes (from 1000 to 10,000), trying to predict the scalability of the system.

The results of the simulations showed that with carefully chosen values of the parameters, it was possible to advertise peer-to-peer networks in a fully decentralized way, using for that purpose already existing Internet infrastructures such as the IRC networks or the Usenet network. The method of advertisement that was described in this work, used with proper values of the parameters, allowed nodes to aggregate into a network made of one single connected component, which was the first requirement for the system to be considered as "working".

Both types of simulations, i.e., using IRC or Usenet as a broadcast channel for the advertisement, behaved similarly and yielded optimal results for approx-

imately the same values of the parameters:

- Low values of the *minimum desired neighbor filling* produced several disjoint connected components, whereas higher values guaranteed a single connected component. Lower values however also reduced the network traffic per node which occurred between the node and the broadcast channel.
- Low values of the *minimum known nodes count for leaving* led to a lower traffic per node, but had very little influence on the clustering efficiency (in the case of the IRC-based system).
- The optimum was located in an area where the minimization of the traffic conflicted with the creation of more than one connected components. The actual optimum had therefore to be chosen so that the probability of getting several disjoint connected components was low, but without getting too high network traffic values.

Moreover, in the case of the Usenet-based simulations, the expiration times of the advertisements had to be taken into consideration in order to minimize the traffic.

The optimal values for the IRC-based simulations were a *minimum desired neighbor filling* of 35 and a *minimum known nodes count for leaving* of 5. These parameters led to a traffic of 6900 bytes per node. The optimal values for the Usenet-based simulations were a *minimum desired neighbor filling* of 35, a *minimum known nodes count for leaving* of 10, and values of the a and b parameters of the expiration time set to respectively 0.3 and 5. These parameters led to a traffic of 9400 bytes per node.

The IRC-based system generated only 3/4 of the traffic produced by Usenet-based system, but Usenet servers are in practice usually dimensioned for much higher traffic than IRC servers. However, the traffic per node in the Usenet-based simulations depended on the number of nodes in the network, whereas it was constant in the case of the IRC-based system, meaning that large networks using Usenet will generate much more traffic than smaller ones.

The *minimum desired neighbor filling* influenced strongly the clustering efficiency of both IRC- and Usenet-based simulations whereas the *minimum known nodes count for leaving* had little influence on the clustering efficiency. This can be explained by the fact that a node gathered knowledge of other nodes mostly by exchanging lists of neighbors with its own neighbors (in order to minimize the usage of the broadcast channel). Therefore, most of the known nodes were already part of the same connected component as the one the node belonged to, meaning that the connections between two disjoint connected components were not favored by the increase of the *minimum known nodes count for leaving*.

The *minimum desired neighbor filling* influenced strongly the network traffic of the IRC-based simulations; the *minimum known nodes count for leaving* had much influence on the traffic of the Usenet-based simulations. This can be explained by the difference in ways IRC and Usenet work: gathering more neighbors required staying connected longer to the IRC channel or retrieving more advertisements

from the Usenet server, but getting additional known nodes when the *minimum known nodes count for leaving* was low required to reconnect often to the broadcast channel, and doing so in Usenet generated much more overhead traffic than in IRC. The traffic generated by the overhead was more important than the one generated by a prolonged usage of the broadcast channel, which explained the difference in the results of the simulation.

Finally, the characterization of the network created during the simulation showed that the networks were similar, but the simulations using Usenet produced networks that were more tightly connected than the simulations based on IRC: the clustering coefficient was higher in the first case, and the so called "diameter" of the network (characterized by the longest and the average shortest paths) was shorter. These results showed that using Usenet as a broadcast channel was producing slightly "better quality" networks, but at the expense of more network traffic.

Further work on the subject may study the efficiency of a WWW-based advertisement system, which was described but not implemented, as well as an optimal scheme for random IP address "knocking" in case none of the abovementioned broadcast channels (IRC, Usenet, Web server) is available. The scheme could e.g., be based on the statistical distribution of the IP address classes to various organizations as well as the distribution of IP addresses to the hosts within these classes, in order to maximize the probability of discovering a member of the peer-to-peer network one tries to join.

Moreover, the current advertisement system relies on the fact that each and every node in the network is eager to take part in the advertisement process. In reality, however, the nodes often act selfishly, and are not willing to share a part of their resources for the common good: the influence of such nodes on the efficiency of system is still to be researched.

APPENDIX 1 SIMULATOR'S FLOWCHARTS

The following flowcharts represent the flow of operations in the variant of the simulator that uses IRC as a broadcast channel, and in the one using Usenet; they are called "IRC simulator" and "Usenet simulator", respectively.

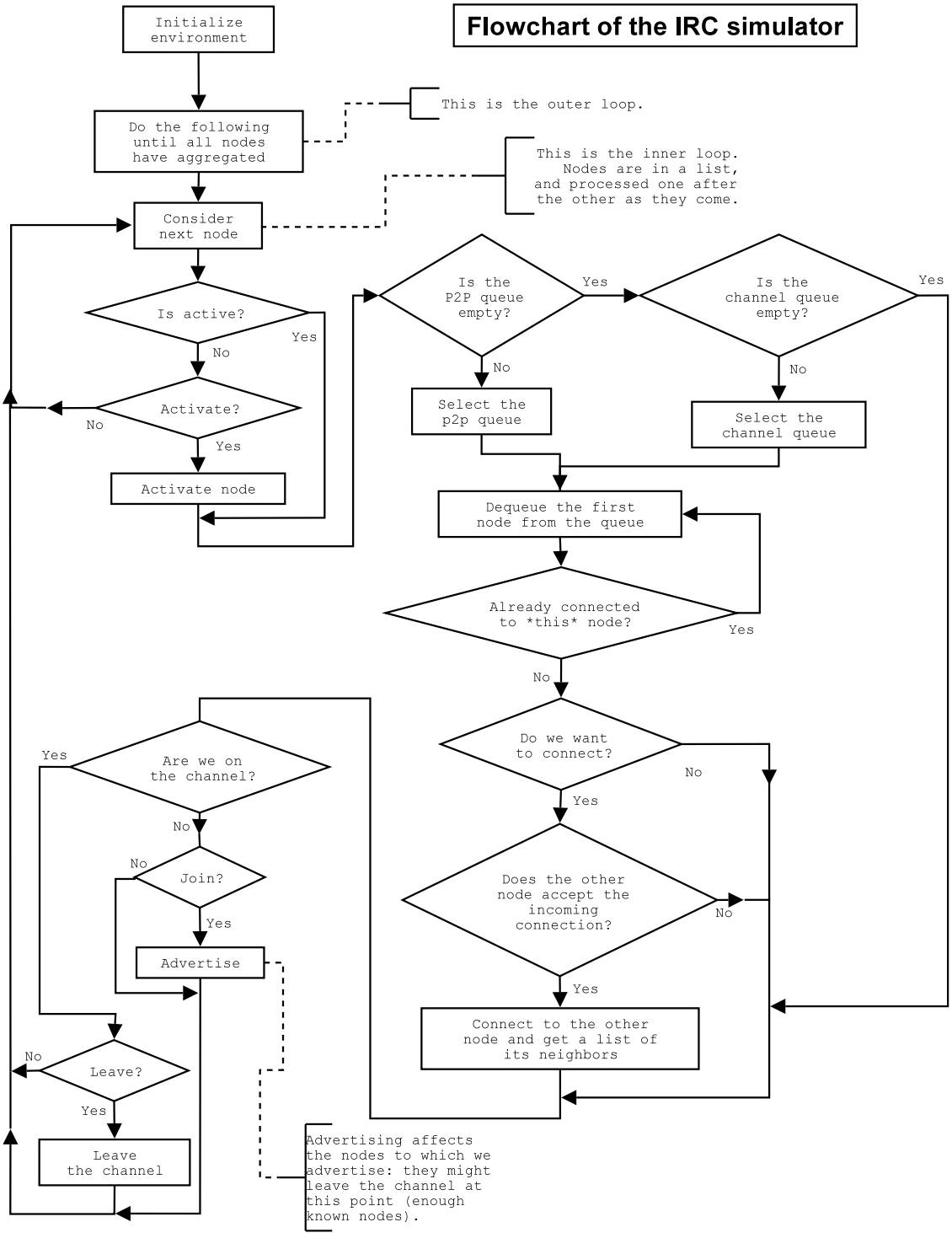


FIGURE 21 Flowchart of the IRC-based simulator

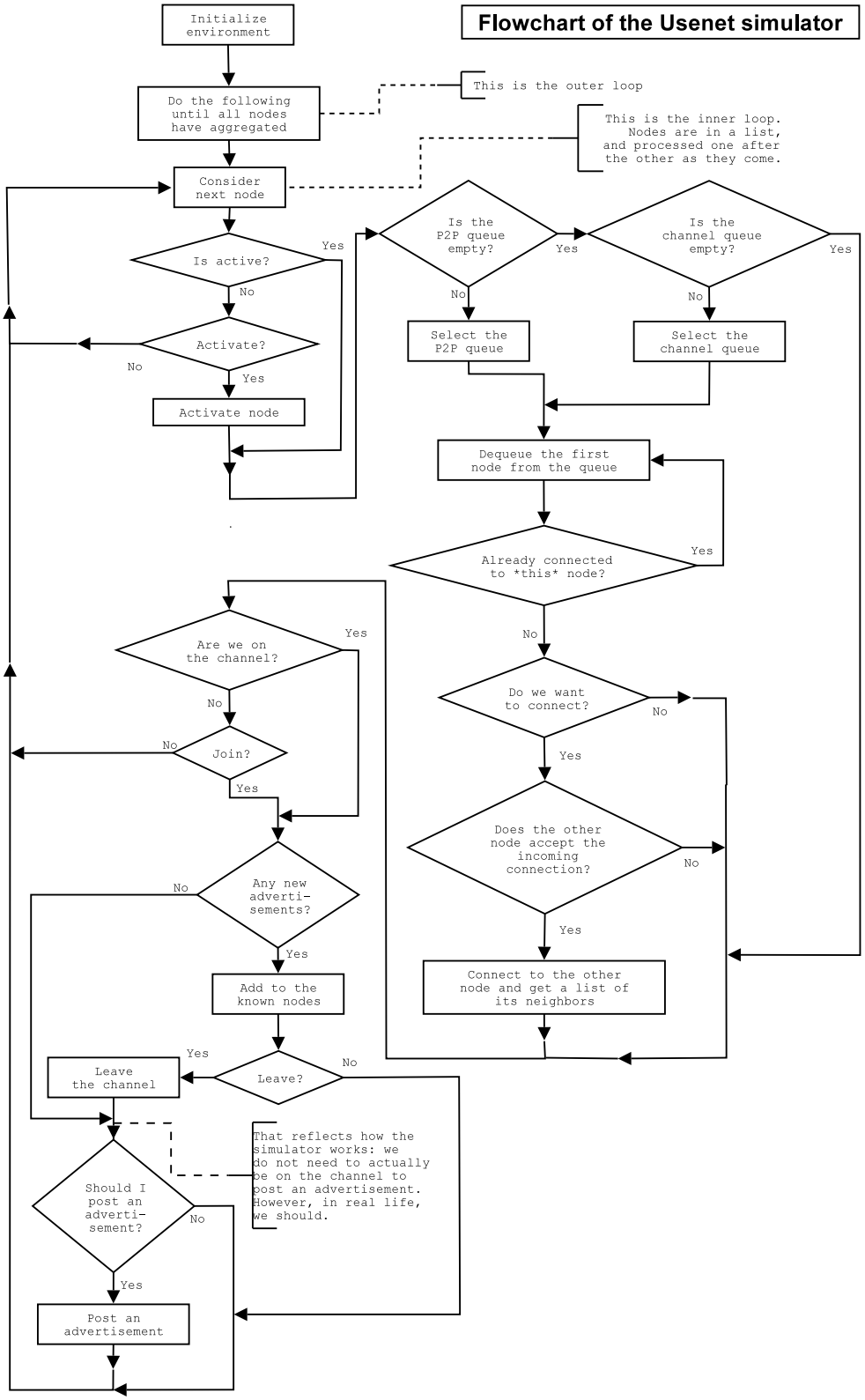


FIGURE 22 Flowchart of the Usenet-based simulator

APPENDIX 2 GNUTELLA DISTRIBUTION RANDOM DEVIATE

Appendix 2.1 Algorithm

In [RIF02], the connectivity distribution of Gnutella nodes is described as follows:

- For node's degrees between 1 and 10, the distribution is mostly uniform.
- For node's degrees between 10 and 100, the distribution is power-law.

In addition, [SGG01] proposes a value of $\lambda = 2.3$ as parameter of the power-law distribution.

Let us name $\mathcal{G}$ as the connectivity distribution. For the sake of simplicity, we consider that $\mathcal{G}$ is uniform for $1 \leq degree \leq 10$ and follows a power-law distribution for $10 \leq degree \leq 100$. The uniform part of $\mathcal{G}$ follows $c(x) = C$ for $x \in [1, 10]$, and the power-law part of $\mathcal{G}$ is of the form $p(x) = Kx^{-\lambda}$ for $x \in [10, 100]$. We also consider that the curve of the entire distribution is continuous, i.e., that $c(10) = p(10)$.

The strategy adopted for creating a random deviate of $\mathcal{G}$ is the following. Let us name A_0 the area under the curve of c and A_1 the area under the curve of p. The properties of statistical distributions imply that the area under the curve of $\mathcal{G}$ is equal to 1. We then have:

$$A_0 + A_1 = 1 \tag{10}$$

This means that a random deviate of $\mathcal{G}$ has a probability of A_0 of being uniformly distributed between 1 and 10, and a probability of $1 - A_0$ of being power-law-distributed between 10 and 100. Therefore, the first step in creating a random deviate of $\mathcal{G}$ will consist in deciding to which part of $\mathcal{G}$ it belongs. This decision can be taken using a uniform random deviate. The second step then consists in generating a random number following c or p, according to the decision taken above.

Equation 10 is equivalent to

$$\int_1^{10} c(x)\,dx + \int_{10}^{100} p(x)\,dx = 1 \tag{11}$$

The function $P(x) = -\frac{K}{1-\lambda}x^{1-\lambda} + Q$ being a primitive of $p(x)$, we can write

$$\begin{aligned} A_0 &= 9C && (12)\\ A_1 &= P(100) - P(10) && (13)\\ &= \frac{K}{\lambda - 1}\left(10^{1-\lambda} - 100^{1-\lambda}\right) && (14) \end{aligned}$$

Since $p(10) = c(10)$, we have $10^{-\lambda}K = C$, therefore

$$K = 10^{\lambda}C \tag{15}$$

We can then write

$$C = \frac{1}{9 + \frac{10^{\lambda}}{\lambda - 1}\left(10^{1-\lambda} - 100^{1-\lambda}\right)} \tag{16}$$

The numerical calculation with $\lambda = 2.3$ then gives:

$$C = 0.06132418896883533025$$

From 15 and 12, we draw

$$A_0 = 0.55191770071951797225$$

The transformation method described in [PFTV92, Chapter 7] allows to find a method for computing random deviate of the p distribution, using the inverse function of P. We have the following constraints:

$$\int_{10}^{100} p(x)\,dx = P(100) - P(10) = 1 \tag{17}$$

and

$$P(10) = 0 \tag{18}$$

From 14 and 17, since in this particular case we have $A_1 = 1$, we can draw

$$K = \frac{1 - \lambda}{10^{1-\lambda} - 100^{1-\lambda}} \tag{19}$$

and from 18 and 19 we can draw

$$Q = \frac{1}{1 - 10^{1-\lambda}} \tag{20}$$

The numerical calculations with $\lambda = 2.3$ gives

$$\begin{aligned} K &= -27.307002182863772813 90 \\ Q &= 1.052763144821798459 12 \end{aligned}$$

We can then use P^{-1}, the inverse function of P, to calculate a random number following the distribution of p.

$$P^{-1}(y) = \left(\frac{1 - \lambda}{K}(Q - y)\right)^{\frac{1}{1-\lambda}} \tag{21}$$

If y is a random number generated by a uniform deviate in $[0, 1]$, then $P^{-1}(y)$ is a power-law 2.3 deviate in $[10, 100]$.

Finally, generating a random number following the distribution of c can be done with the formula:

$$x = 9y + 1 \tag{22}$$

If y is a uniform deviate in $[0, 1]$, then x is a uniform deviate in $[1, 10]$.

The algorithm of a deviate following $\mathcal{G}$ is then described in Algorithm 6. It relies on the function *my_rand()* which is a uniform random deviate in $[0, 1]$.

if *my_rand()* < 0.5519 **then**
  return *my_rand()* $\times 9 + 1$
else
  return $(0.04761 \times (1.053 - \textit{my\_rand()})^{-0.7692}$
end if

ALGORITHM 6 Gnutella deviate

Appendix 2.2 Goodness of Fit

The random deviate for p has been tested for goodness-of-fit using the χ^2 method, as described in [Knu98]. The main idea behind the test is to evaluate how a set of experimental values (obtained from the random deviate) fits a set of expected values (obtained from the theoretical distribution of the random deviate). Once the χ^2 has been calculated, one can deduce, based on the χ^2 statistical distribution, the probability that the experimental distribution is identical to the expected distribution. Once this probability is known, the rest is a matter of interpretation. [Knu98] states that probabilities situated in the $[0.10, 0.90]$ interval are acceptable values, meaning that the random deviate produces a distribution similar to the expected distribution. Below 0.10, the experimental values are too close to the theoretical ones to be considered as truly random values, and above 0.90, they are too random to be considered close enough to the distribution of the expected values.

Three different uniform deviate have been use as implementation of the *my_rand()* function:

1. The GNU C Library *rand()* function.
2. The *ran1()* function described in [PFTV92, Section 7.1].
3. The algorithm described in [MN98] and implemented by Takuji Nishimura.

The results of the Gnutella deviate depend on the quality of the uniform deviate and on the value of the seed. The χ^2 value has been computed from a set of 100,000 random values, and 15 similar χ^2 tests have been run, each with a different seed. The seed for each run was taken from the computer's system clock using the Unix *time()* function, which returns the number of seconds elapsed since January 1st, 1970. The results were as follows:

- With the GNU C Library *rand()* function, 8 runs out of 15 gave χ^2 values within the $[0.10, 0.90]$ probability interval.
- With the *ran1()* function described in [PFTV92, Section 7.1], 11 runs out of 15 gave χ^2 values within the $[0.10, 0.90]$ probability interval.
- With the algorithm from [MN98], 13 runs out of 15 gave χ^2 values within the $[0.10, 0.90]$ probability interval.

APPENDIX 3 SIMULATION RESULTS FILE EXAMPLE

```
IRC simulator, build 290
Command line: /home/mweber/research/external-adv-simulator/bin/simulator-irc
                1000 2000 powerlaw 1 100 100 100 100 1000 2000 45 25 0:50 9
Parameters:
  Number of nodes: 1000
  Number of iterations: 2000
  Max neighbor distribution: powerlaw
  Activation probability: 1.000000
  Incoming connection acceptance probability: 100.000000
  Connection decision probability: 100.000000
  Channel join probability: 100.000000
  Channel leave probability: 100.000000
  Maximum channel stay duration: 1000
  Channel capacity: 2000
  Minimum desired neighbor filling: 45.000000
  Minimum known nodes count for leaving: 25.000000
Random generator seed: 9
Running time: 2 sec
Memory usage: 4252 kB
Nodes forming a network with node 0: 997
  Biggest max neighbors: 98
  Connectability distribution: 48 76 59 60 68 58 65 60 61 52 35 35 37 22 35 24 14 11
                               17 9 7 4 7 12 4 6 4 6 5 5 4 3 5 5 8 3 5 2 2 2 3 0 1 2
                               5 2 0 0 1 3 0 1 4 2 0 1 0 1 2 3 1 1 1 3 1 0 0 0 1 1 0
                               0 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 2 0 0 0 0 2 0 0 1 1 1
  Connectivity distribution:   49 75 59 60 68 58 65 60 61 52 35 35 37 22 35 24 14 11
                               17 9 7 4 7 12 4 6 4 6 5 5 4 3 5 5 8 3 5 2 2 2 3 0 1 2
                               5 2 0 0 1 4 0 2 5 3 0 1 0 1 2 2 2 1 2 2 1 1 0 1 3 1 1
                               0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  Neighborhood filling level: 0 0 0 0 0 1 0 3 8 5 980
Nodes forming a network with node 886: 3
  Biggest max neighbors: 98
  Connectability distribution: 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  Connectivity distribution:   2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  Neighborhood filling level: 0 0 0 0 0 0 0 1 0 0 2
Number of disjoint networks: 2
Total channel joins/leaves: 1039/1036
Max users on channel: 11
Global traffic on channel: 7485584
Channel/P2P connections: 1381/4896
Iterations without action before the end: 1228
Real iterations used: 772
```

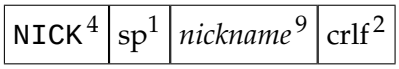
FIGURE 23 Example of the results of one simulation

APPENDIX 4 IRC AND NNTP PROTOCOL MESSAGES

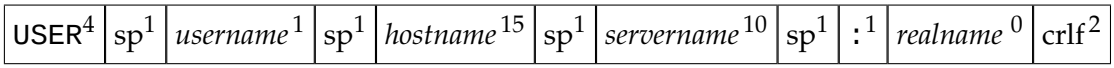
This appendix contains the descriptions of the IRC [OR93] and NNTP [KL86] protocol messages described in Chapter 5. Each figure represents the structure of the messages, the length of each part and the total length of the message. The typographical convention for the parts of the message is as follows:

- Text strings are set in `fixed width font`.
- Single ASCII characters are represented in roman font; white space characters are represented by a symbolic name ("sp" is "space", "cr" is "carriage return" and "lf" is "line feed"); unspecified digits are represented by the string "digit".
- Arguments are represented by their names and set in *italic font*.
- A framed ellipsis $(\cdots)$ represent protocol parts of unspecified length, repeating previous parts.
- If a protocol message is too long to fit on one single line, it is split across several lines. The lines after the first one are indented and preceded by an ellipsis $(\cdots)$ (which is not framed, on the contrary to the one above.)

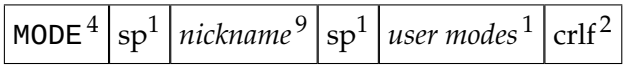
The length of each part is shown in exponent, and each part is placed inside a frame in order to clearly show the separation between the different parts of a message.



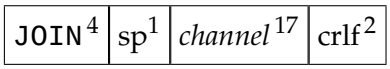
(a) NICK request (16 bytes)



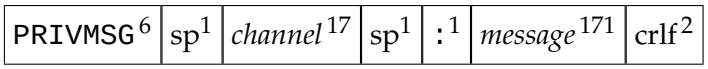
(b) USER request (37 bytes)



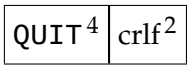
(c) MODE request (18 bytes)



(d) JOIN request (24 bytes)

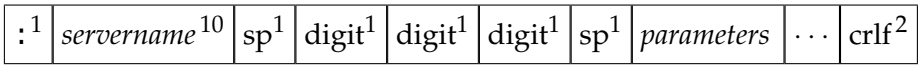


(e) PRIVMSG request (199 bytes)

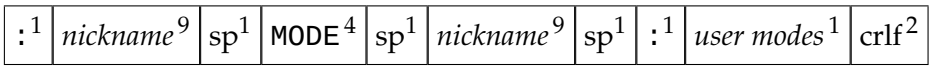


(f) QUIT request (6 bytes)

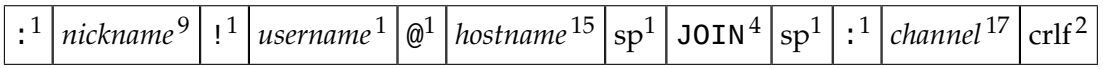
FIGURE 24 IRC request messages sent by the node and their sizes



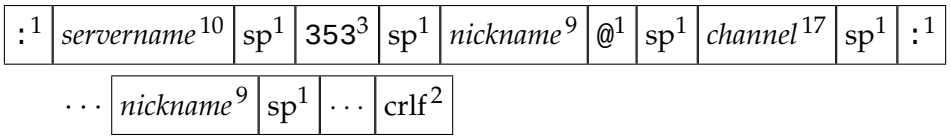
(a) Numerical replies (18 + n bytes)



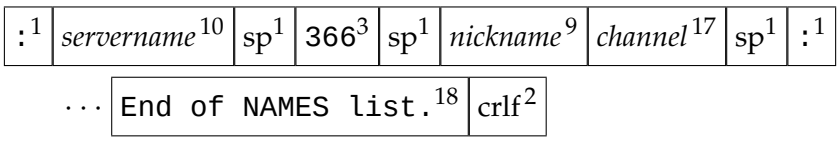
(b) MODE confirmation (30 bytes)



(c) JOIN indication and confirmation (54 bytes)

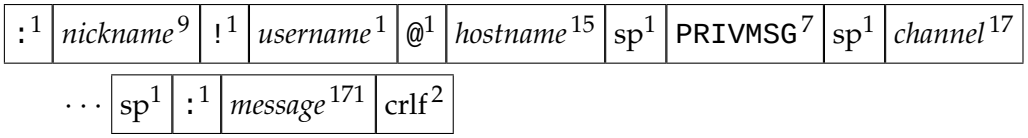


(d) Name reply (48 + 10 × *members*, *members* ≤ 46 bytes)



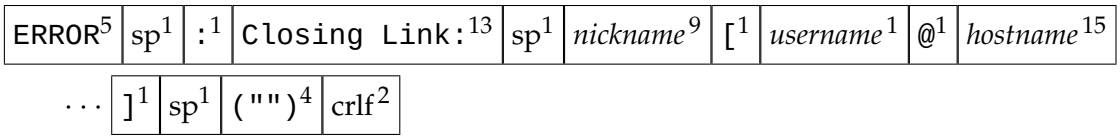
(e) End of name reply (64 bytes)

FIGURE 25 IRC indication messages received by the node during the connection step and their sizes



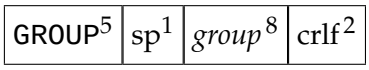
(a) PRIVMSG indication (228 bytes)

(b) QUIT indication (39 bytes)

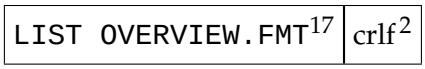


(c) QUIT confirmation (56 bytes)

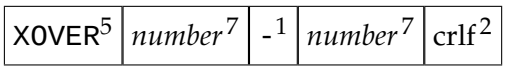
FIGURE 26 IRC indication and confirmation messages received by the node after the connection step and their sizes



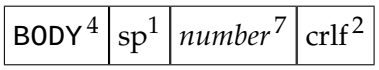
(a) GROUP request (16 bytes)



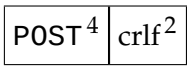
(b) LIST OVERVIEW.FMT request (19 bytes)



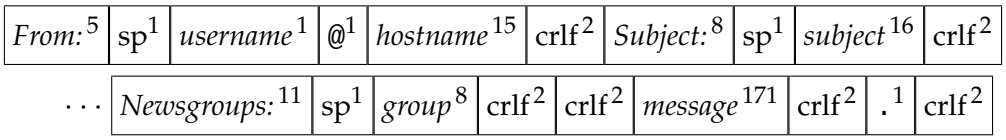
(c) XOVER request (22 bytes)



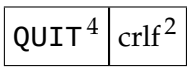
(d) BODY request (14 bytes)



(e) POST request (6 bytes)



(f) POST request continuation (252 bytes)



(g) QUIT request (6 bytes)

FIGURE 27 NNTP request messages sent by the node and their sizes

`200`[3]	sp[1]	*greeting*[49]	sp[1]	`(posting ok).`[14]	crlf[2]

(a) Server greeting (70 bytes)

`211`[3]	sp[1]	*number*[7]	sp[1]	*number*[7]	sp[1]	*number*[7]	*group*[8]	crlf[2]

(b) GROUP reply (37 bytes)

`215`[3]	sp[1]	`Order of fields in overview database.`[38]	crlf[2]

···	*list of fields*[78]	`.`[1]	crlf[2]

(c) LIST OVERVIEW.FMT reply (125 bytes)

`224`[3]	sp[1]	*number*[7]	`-`[1]	*number*[7]	sp[1]	`fields follow`[13]	crlf[2]	*overview record*[212]

···	crlf[2]	···	`.`[1]	crlf[2]

(d) XOVER reply ($38 + 214 \times$ *records* bytes)

`222`[3]	sp[1]	*number*[7]	sp[1]	*message-ID*[35]	sp[1]	`body`[4]	crlf[2]	*message*[171]	crlf[2]	`.`[1]	crlf[2]

(e) BODY reply (230 bytes)

`340`[3]	sp[1]	`Ok, recommended ID`[19]	sp[1]	*message-ID*[35]	crlf[2]

(f) POST reply (61 bytes)

`240`[3]	sp[1]	`Article posted`[15]	sp[1]	*message-ID*[35]	crlf[2]

(g) POST confirmation (57 bytes)

`205`[3]	sp[1]	`.`[1]	crlf[2]

(h) QUIT reply (7 bytes)

FIGURE 28 NNTP indication messages received by the node and their sizes

APPENDIX 5 TRAFFIC PER NODE FOR INCREASING NUMBER OF NODES

The following tables represent the results of the linear regression (with the least-squares method) of the clustering efficiency for increasing numbers of nodes; the first table is for the IRC based system, and the second is for the Usenet-based one.

Two regression models, linear (of the form $ax + b$) and logarithmic[1] (of the form $a\log x + b$), have been considered for each pair of *minimum desired neighbor filling* and *minimum known nodes count for leaving* and the one yielding the best correlation coefficient has been chosen.

The first two columns are the values of the *minimum desired neighbor filling* and the *minimum known nodes count for leaving*. The third column shows therefore which model has been chosen. The fourth and fifth columns show the a and b values resulting from the linear regression, respectively. The sixth column is the correlation coefficient of the data.

Appendix 5.1 IRC-Based System

001	001	log	10.2096624861302	6199.67777941911	0.970774016395325
001	005	log	15.615035966996	6231.54489444667	0.981493745577975
001	010	log	29.1538351143142	6300.1442006426	0.977871022971157
001	015	log	49.1689357507987	6309.27170481303	0.972935785994571
001	020	log	80.9738492626845	6238.60426940236	0.974588734638538
001	025	log	117.122649599588	6112.59255239095	0.976807022312565
001	030	log	154.318641917281	5941.60035415086	0.978428217651762
001	035	log	196.895085480216	5742.77269062085	0.97936062998198
001	040	log	247.691629540707	5483.20538223842	0.981202300690396
001	045	log	295.441930796687	5205.15999762297	0.98274649028911
001	050	log	363.814346972795	4834.49461934367	0.985617961635142
001	055	log	409.214428958451	4528.31533172771	0.986986300173046
001	060	log	471.723093777034	4154.52462417985	0.988543806459406
001	065	log	527.231405531174	3785.17103622968	0.989908831640928
001	070	log	591.428000233981	3387.17668278803	0.990729171034307
001	075	log	657.55422652365	2968.4549595315	0.991824396682392
001	080	log	787.972781568821	1958.63071530838	0.993824734641786
001	085	log	842.851536905828	1585.33955449625	0.994888539371551
001	090	log	908.082206754916	1153.20872014773	0.995638035484306
001	095	log	964.019648343999	758.329162169151	0.996431047025823
001	100	log	1071.56997978809	88.715543327311	0.996966901881438
005	001	log	10.5359273359059	6219.06371757906	0.934471470747877
005	005	log	10.2095441165086	6286.71641878578	0.996565987378989
005	010	log	15.1902908802056	6433.1827462365	0.983626167089593
005	015	log	20.9836381631275	6576.02563278083	0.983305243269473
005	020	log	34.142902116392	6681.64244020891	0.981883485890179
005	025	log	48.0332430173792	6767.12104642503	0.978629477103462
005	030	log	63.1508903023265	6807.49574865058	0.97641231895364
005	035	log	80.627834571642	6850.80548393754	0.971375127138366
005	040	log	104.160213896063	6854.89841662282	0.97334691601997
005	045	log	126.493338015578	6825.38076128752	0.970340034193438
005	050	log	162.565534643892	6773.50037357497	0.970376383310814
005	055	log	184.410168470675	6703.02621541301	0.967903310593675
005	060	log	222.648541914689	6576.71642092424	0.967627693963661
005	065	log	256.98063990497	6428.24521619493	0.965845713425723
005	070	log	303.995338484148	6217.69510143306	0.964049155250952
005	075	log	355.101446989876	5971.0249679625	0.962430634269511
005	080	log	405.346824981377	5702.75284505485	0.961703976043779
005	085	log	455.178123201645	5414.20251993507	0.961386074982267
005	090	log	515.637359339739	5068.01678428175	0.961881703984075
005	095	log	572.925702216614	4706.30840324895	0.962043583342371
005	100	log	679.227873692728	4114.5232632589	0.961477147674987
010	001	log	2.8149972035319	6350.13565314568	0.674381869770872
010	005	log	5.8246568225902	6353.62532121524	0.929193095899557
010	010	log	8.72023379125099	6502.71405824188	0.953213658619659
010	015	log	10.9035861305221	6675.26437578918	0.974484622323375

[1] The data has been linearized before applying the linear regression method

010	020	log	17.528492559952	6840.74816985182	0.983698678694823
010	025	log	23.8531691714907	6997.66844832774	0.984182716252314
010	030	log	30.3063343901149	7119.41432356047	0.980429786160978
010	035	log	37.7673492260735	7257.42985078092	0.974749209812963
010	040	log	49.0933784301636	7377.08705640543	0.976420182673833
010	045	log	60.1843998854514	7454.11417786079	0.972377643349189
010	050	log	80.1062102742529	7555.81226245889	0.973779639694332
010	055	log	89.0268037362471	7607.92804509101	0.970936029308028
010	060	log	109.62939978881	7648.63955066367	0.970192169279004
010	065	log	125.072261238666	7677.66121225397	0.968970890801149
010	070	log	146.200439081481	7708.72840757336	0.969013387709396
010	075	log	170.692482287504	7710.41334225775	0.968823561732335
010	080	log	193.954696175408	7694.4246022473	0.967965927888367
010	085	log	215.046747179909	7674.3216429852	0.967275961780908
010	090	log	245.035160615118	7612.82069105898	0.966931270888687
010	095	log	271.745098658929	7537.51745719261	0.965738736588625
010	100	log	335.398842374276	7349.64230369578	0.963030483339501
015	001	log	3.60895293495841	6431.66334852235	0.731964851722573
015	005	lin	-2.28245793640614e-05	6474.60772849029	0.028811104879834
015	010	log	3.96251591987528	6576.61902604165	0.86858590368304
015	015	lin	0.000695346928570668	6783.06775517461	0.678117990231781
015	020	log	8.79040913448552	6930.71437151684	0.960471341966798
015	025	log	10.6946786018984	7125.95696769934	0.98161097559962
015	030	log	14.7717929758178	7269.78610060414	0.976040685790575
015	035	log	19.5633605596562	7432.57742066304	0.962245614621715
015	040	log	25.381987611231	7604.46880907388	0.967438463602382
015	045	log	32.3480367939272	7720.93648087475	0.964406694419473
015	050	log	43.248128331388	7907.39313199172	0.965659312172225
015	055	log	47.1808570467837	8006.25532508297	0.962933907392029
015	060	log	60.2099990280744	8119.06042855378	0.962169151148979
015	065	log	67.7929963438294	8222.80784857312	0.959862496563069
015	070	log	81.8469539942375	8320.80726182312	0.962330426462401
015	075	log	96.229414766199	8418.10844775148	0.963449117886691
015	080	log	109.809130857471	8494.17633485136	0.962023043046061
015	085	log	122.832348051702	8550.24041553403	0.961813518183117
015	090	log	139.779073936027	8611.93519962247	0.961995824588687
015	095	log	152.008885680346	8673.11155533748	0.961084717930405
015	100	log	193.205092423661	8695.79780059546	0.960800317542783
020	001	log	-5.05776313633738	6600.40040623871	0.737389221699394
020	005	lin	0.000416000546211292	6559.62712060176	0.647914107246884
020	010	log	-2.33556900831149	6688.39050838841	0.598745423618042
020	015	lin	0.000137445369167039	6825.02568787037	0.122770696737542
020	020	log	-1.28333098392161	7043.67004614315	0.728643691882666
020	025	lin	0.000596153226190805	7234.62542891005	0.728871528544306
020	030	log	3.48887751854305	7384.01990385367	0.740820316287251
020	035	log	7.53978039362892	7551.68276835132	0.889295921120702
020	040	log	12.7834234010345	7728.30947753009	0.916823788283341
020	045	log	17.4689108772884	7864.89328015638	0.942510359061552
020	050	log	22.7128840032201	8104.91543972696	0.939834868747762
020	055	log	24.3068701633004	8225.27757537774	0.937107533060235
020	060	log	33.3578083015776	8375.7474664409	0.942628052576494
020	065	log	39.0026757297489	8498.07940558633	0.939704419931619
020	070	log	48.1413061459697	8642.87547786195	0.945263135396057
020	075	log	58.1649900204806	8781.66574167125	0.949940968785117
020	080	log	67.3668102187665	8898.78555461945	0.949221810597679
020	085	log	74.9436480612542	9006.90566423961	0.947539490094793
020	090	log	89.0193452806689	9096.08946071638	0.949864569708996
020	095	log	95.1827217871883	9214.97231516977	0.947796221934083
020	100	log	119.661865223818	9394.74437634953	0.950194079260865
025	001	log	-6.5336074210872	6709.75917512755	0.510431511648017
025	005	log	-3.58184925206907	6692.24847351093	0.392367602096497
025	010	log	-1.95287782692479	6761.54110279516	0.276823625941435
025	015	log	-1.661783813617	6897.98481491294	0.864413100239696
025	020	lin	-0.000265479436748678	7073.29164514815	0.321428596111052
025	025	lin	-0.000305379174223968	7269.01880358087	0.698622026101842
025	030	log	2.44286956540896	7415.32835685041	0.682748243287605
025	035	log	-1.27376404857784	7647.53897820686	0.262890768414084
025	040	log	1.76674180672782	7840.19427235167	0.585561165240916
025	045	log	4.06496363020343	7996.45928111418	0.718479308979662
025	050	log	8.84158640124105	8239.8616463178	0.83955622679737
025	055	log	12.8708011514164	8337.72223704367	0.870921309642982
025	060	log	18.1717623722898	8522.71335951317	0.900509568061382
025	065	log	22.5804919769182	8656.85954935305	0.901988504471891
025	070	log	29.2292516602951	8824.79446612209	0.923299513724902
025	075	log	36.6441536690956	8988.22110747743	0.92632288094757
025	080	log	43.0724440583964	9131.98085329319	0.929769678970468
025	085	log	50.0016400061371	9245.30140809419	0.934278117450078
025	090	log	59.281602669776	9380.9740079094	0.932761468024981
025	095	log	65.0425000466885	9503.18692618884	0.934037810912362
025	100	log	81.230428390728	9761.44093378518	0.937101307308168
030	001	log	2.28576815682122	6725.80965891277	0.27599280078186
030	005	log	-2.1198546029885	6769.63299848187	0.140605130320324
030	010	lin	5.57215079367161e-05	6814.39710075838	0.0378252104298932
030	015	log	-0.191383681380206	6948.22099352006	0.0725726815434866
030	020	log	-1.63160855557343	7134.50765205857	0.2731982464071
030	025	lin	-0.000407019177057913	7307.64636898619	0.334501443209956
030	030	log	5.07455437263741	7420.40023734594	0.511858035136833
030	035	lin	-0.000621689289683104	7665.50087383775	0.43686746860259
030	040	lin	-0.000314287908727924	7876.97308068429	0.576702649191927

030	045	lin	-0.000172532154761751	8049.20146198942	0.252730569415802
030	050	log	5.76483568843821	8279.80729526011	0.74348918267923
030	055	log	6.37572433477974	8406.13056713224	0.670571265296145
030	060	log	10.5712558098472	8600.02253413146	0.824792348740167
030	065	log	14.6898876116445	8736.33262602485	0.844303903082013
030	070	log	23.2622855930546	8885.88511028865	0.885160523472614
030	075	log	27.0534125335652	9081.70270688775	0.907759674466265
030	080	log	34.2484782082752	9218.50822433404	0.912675866243541
030	085	log	41.9955857368518	9324.73664807924	0.91593876492262
030	090	log	48.2949318322644	9487.87147815455	0.915907942064297
030	095	log	54.1656508797177	9608.83092136084	0.918291503605121
030	100	log	66.3822032247886	9903.62055980509	0.92135830176705
035	001	lin	-0.000423148404761155	6831.91807939682	0.208599981239964
035	005	lin	0.00100102432300931	6827.75863684062	0.434648275213678
035	010	log	-4.10266348169077	6933.52104549129	0.462447495308251
035	015	lin	0.000599074363155799	7009.83701218518	0.399357570718261
035	020	lin	-6.9742052624623e-05	7173.73899769226	0.0849298559408138
035	025	lin	-0.000219010824949414	7350.41773064728	0.216757541661639
035	030	log	1.80187688607117	7486.76706043844	0.486763907876539
035	035	log	2.44835012471628	7673.33549814303	0.286907682653732
035	040	lin	-0.000458143988094727	7904.18609243386	0.4440425063339
035	045	log	-3.29981569898961	8098.94886617002	0.597847169914762
035	050	log	6.19634490092489	8294.19887372492	0.749855524945571
035	055	log	5.77021664893331	8427.51998693256	0.66515954815397
035	060	log	5.38566395419809	8658.98744973316	0.634835572266953
035	065	log	11.7935839542149	8774.03728236406	0.780102263887186
035	070	log	15.7427903523603	8964.38894934884	0.835555355260966
035	075	log	19.460821034256	9159.78401035818	0.868024288111227
035	080	log	27.3447190504734	9289.73412043776	0.884446734111359
035	085	log	32.1697546053894	9422.26410114521	0.898083585248311
035	090	log	38.5411766732404	9584.18260991245	0.896804255769585
035	095	log	43.9411132571351	9708.77768902446	0.900622359766575
035	100	log	54.0398331966698	10024.1867268159	0.902090885492067
040	001	log	-9.08731718901677	6991.86348015694	0.644555686496515
040	005	lin	0.000930597106755527	6910.25959186209	0.300459078999694
040	010	lin	0.000572060559432907	6972.23686451397	0.396507275719785
040	015	lin	0.00104376934415499	7077.73216550001	0.469155955484522
040	020	log	4.65226750492852	7190.25129268135	0.587591547050754
040	025	log	-5.84890022416496	7449.19357971531	0.630759560232494
040	030	lin	-9.41700232337558e-05	7543.78038639344	0.0828513240205908
040	035	lin	-0.00042121989754619	7734.37088655555	0.313721438305966
040	040	lin	-0.000645155228174868	7934.07411512897	0.402495333664088
040	045	log	-6.98474622823186	8158.13138865108	0.74592562067474
040	050	log	2.27898443399898	8349.95384281728	0.57696930527902
040	055	log	2.62939949002499	8475.24734304504	0.306807402780744
040	060	log	1.32907333566579	8712.07062232645	0.244007176771047
040	065	log	6.19619592477517	8839.262335805	0.630838041217193
040	070	log	10.5685884748667	9023.84903906679	0.78114831090891
040	075	log	14.89875319858	9213.18983987514	0.816072926972559
040	080	log	20.7336003301895	9360.75870708414	0.864581949641763
040	085	log	25.8265439779917	9489.78863905009	0.863351997049491
040	090	log	33.1106578546508	9643.04615636182	0.868542475504075
040	095	log	39.0165374215907	9763.00965523603	0.878970040541712
040	100	log	46.058363638873	10105.6659668855	0.878329290742871
045	001	log	-2.54682806421502	7017.28140402302	0.208207399841516
045	005	lin	0.00355256003571351	6965.30466382011	0.758782282404377
045	010	log	-2.90056190464269	7081.45281559291	0.408314934796411
045	015	lin	0.000624362481252578	7147.08137168256	0.52278135513597
045	020	lin	0.00067211065103502	7288.05437635185	0.523778843008695
045	025	log	-5.38884540442031	7499.42574035301	0.622575460328842
045	030	lin	0.000558476048580805	7583.8906712963	0.21044903893436
045	035	log	0.606747462350354	7762.33436111546	0.124492044362467
045	040	lin	-3.14281343758514e-05	7960.12451257012	0.0349494701432457
045	045	log	-4.96605631736839	8167.98982737105	0.476287453452017
045	050	log	1.31432480894324	8382.49365306972	0.227310062594114
045	055	log	5.3279875811241	8475.00557754391	0.593917543437955
045	060	lin	-0.000458652611112595	8747.27114634569	0.472496946416018
045	065	log	4.39756407648789	8872.89387143889	0.462900837830191
045	070	log	4.85301370664631	9089.90530702585	0.496241046456412
045	075	log	15.5733621523513	9221.39675421703	0.869803524903268
045	080	log	20.324085030288	9376.9946096379	0.873940620134424
045	085	log	25.1100609560611	9507.59559031935	0.863686259729847
045	090	log	30.9074486479741	9673.32120788567	0.867857938845134
045	095	log	37.8234558350536	9783.14947362038	0.878175021764183
045	100	log	45.7401923111243	10117.6056986543	0.875611256713778
050	001	lin	0.0003675572777777791	7083.30129269136	0.0842162942665508
050	005	log	13.1827851571746	6971.16279687782	0.700450680925255
050	010	log	-2.7354099515505	7176.90214103582	0.192883389986663
050	015	log	4.16151846848687	7203.44025110232	0.678694553644156
050	020	log	-3.0912273510666	7397.57307057011	0.238044071464356
050	025	log	0.655235082195775	7512.77533378076	0.079286551627664
050	030	lin	-0.00053926803968241 7	7650.47235627778	0.354375662269829
050	035	log	-4.87293658248083	7861.17246416575	0.580104160444393
050	040	lin	-0.00102642461471847	8014.20154866667	0.590319115509189
050	045	lin	-0.000576560242064794	8168.5671598695	0.429817993472243
050	050	lin	-0.000754705252575445	8429.24007465896	0.733997854333163
050	055	log	1.53342620980541	8535.25944136272	0.174495708582368
050	060	log	-3.82096642348488	8801.9388051002	0.488817150496304
050	065	log	2.42792216095271	8912.4876503803	0.207809431078043

050	070	lin	-0.000457219273430961	9158.06609276738	0.395312840864453
050	075	log	1.68394728512054	9364.8528299309	0.158640695439375
050	080	log	15.67253785563	9437.02872225017	0.800826314622109
050	085	log	20.8380071165398	9562.1542396108	0.861075830742437
050	090	log	29.2848607598562	9704.49525194704	0.874674205906783
050	095	log	32.3335606572713	9846.04423850717	0.882761912298298
050	100	log	36.6291323583146	10214.4695520546	0.865762171314422
055	001	log	-9.07362277787508	7216.89882292568	0.502331045071887
055	005	lin	0.000818223619047801	7132.5243561164	0.398114517908982
055	010	lin	0.00111032188311562	7200.67260500001	0.382381909327301
055	015	lin	0.00149688613131191	7287.97915344445	0.612127080121758
055	020	lin	0.00091048676118345	7419.76839994444	0.28659905132621
055	025	log	-0.742698520133174	7575.32777891922	0.11431704784535
055	030	lin	0.00147039280952315	7677.96340853969	0.823454475999102
055	035	log	-6.71107162518865	7920.78393774329	0.67189516706844
055	040	lin	-0.000280894997114124	8040.72393588889	0.187480772717254
055	045	log	-4.79909261351914	8238.07406341224	0.706876633127785
055	050	lin	-0.000916749507936835	8457.60782242681	0.590946167252119
055	055	log	7.80504309896072	8503.34924601636	0.810848292820825
055	060	log	-6.07790379570925	8843.95245731677	0.765459160360758
055	065	log	3.21675510198533	8925.38625682808	0.255251688641482
055	070	lin	-0.000601578519936119	9177.24201418415	0.449999015510308
055	075	log	9.1434294460098	9315.73732389312	0.6603854227479
055	080	log	15.3740040171253	9454.3565486221	0.786444932659937
055	085	log	21.4850269755657	9571.62670813119	0.859545126159106
055	090	log	26.2111052429737	9744.39768078243	0.851987355585284
055	095	log	29.4936371986993	9884.28034815616	0.869574461810396
055	100	log	34.7141157567417	10242.7872037197	0.861540518323994
060	001	log	-10.3741650878386	7291.98718892663	0.558562039059713
060	005	lin	0.00111919366269906	7201.88803765784	0.358698023336097
060	010	lin	0.000234832658079071	7280.83791244731	0.235577962112023
060	015	lin	0.000901784006974914	7364.57940831481	0.242467902751756
060	020	log	9.84490898511848	7400.45622753368	0.564190922181853
060	025	log	-7.07628034511	7694.1591078934	0.40987676991676
060	030	lin	-0.00117444635135606	7753.61444320883	0.599412072409415
060	035	lin	-0.000869779111111164	7917.75249960494	0.485141542677795
060	040	lin	-0.00058039072222213	8090.78687291358	0.316759905060879
060	045	log	-5.55275236714411	8282.50555606111	0.551558893301385
060	050	lin	-0.000261583726189534	8483.82891946031	0.181406477836163
060	055	log	1.60782476036351	8586.28184649407	0.44960169344592
060	060	log	-6.61909560238213	8872.59894239166	0.722062825914419
060	065	log	1.48439826172295	8963.23508624453	0.113440262358376
060	070	lin	-0.000186657654475598	9195.3930020904	0.215323231098713
060	075	log	10.0481433416226	9325.8368017361	0.678573627096322
060	080	log	15.1825742185588	9473.0409918058	0.756945393896195
060	085	log	15.6509778122823	9638.76497601578	0.8047485456419
060	090	log	21.6430831564155	9799.56219424493	0.835151917546583
060	095	log	26.2319940231094	9926.44269958573	0.847547903675341
060	100	log	34.1506992092389	10260.2476726269	0.849889663339178
065	001	log	-9.63329907567891	7334.3695994431	0.470590710892354
065	005	log	12.0362005064886	7150.05892634007	0.651545994790725
065	010	lin	0.000496486569821835	7339.82300968207	0.261797337865327
065	015	lin	-0.000249325773922866	7433.94425232498	0.348554198186113
065	020	lin	0.00253774416907195	7533.00357990741	0.92161144289063
065	025	lin	0.00136834786171263	7674.45445224074	0.517880392541942
065	030	lin	0.000700991581287529	7788.94634620371	0.448347491961576
065	035	log	-4.4520258893106	7998.16525436531	0.503076092700277
065	040	lin	0.000798919413186942	8117.33220266665	0.643199019395439
065	045	lin	-0.000567653292905184	8277.1249900952	0.541124069690103
065	050	log	1.91546700792665	8498.69095420897	0.178948898429803
065	055	log	2.40151176160295	8608.34810657185	0.24343961360536
065	060	lin	-0.00124918403958707	8847.96717182775	0.835188040463195
065	065	log	1.9016559869073	8982.06583972735	0.143294122977051
065	070	log	-6.26757250514044	9271.47582152609	0.628408709052679
065	075	log	5.7852800195131	9381.93168750318	0.5446023823935
065	080	log	9.7123252950525	9537.77787632108	0.6793309286774
065	085	log	10.4152452626295	9699.16050884216	0.666112582770809
065	090	log	18.9434337819579	9839.04952803441	0.805803417217511
065	095	log	20.5694133275134	9989.34064967072	0.808818452632372
065	100	log	34.3764165994348	10269.1568387056	0.861397504690839
070	001	log	-13.9679862479589	7450.19954231528	0.480662429075246
070	005	log	10.7206827163827	7252.65664553169	0.686673023740122
070	010	log	11.0462278230769	7342.17052141196	0.672249763600156
070	015	lin	0.000562489712602442	7513.08841564815	0.15793195077893
070	020	log	10.0537995998824	7547.27029118582	0.659288439008268
070	025	lin	0.000775465888409181	7764.28935696296	0.238420995490438
070	030	log	-2.37389012394645	7898.35599723005	0.137545815458869
070	035	lin	-0.000772758090565185	8029.65785910387	0.410580776419576
070	040	log	1.23758822434686	8177.98159079486	0.133396868895562
070	045	log	-0.834974822435948	8335.3757585351	0.126246810147048
070	050	lin	0.000367838021007531	8560.26664238656	0.29317537972708
070	055	lin	0.000420371273872064	8669.116787635	0.29108701822838
070	060	lin	-0.000419637033256746	8878.18598408918	0.271088840100895
070	065	lin	-0.00100214501587351	9041.27901640917	0.628539954093186
070	070	log	-3.47872023085362	9274.5247103251	0.312081261298047
070	075	log	6.6205660079946	9397.66623176314	0.600476441167693
070	080	log	10.2128589676488	9554.62323460109	0.799381645090382
070	085	log	10.9096144970094	9716.8906451811	0.732980362776066
070	090	log	16.0234077799956	9883.24354572266	0.881210850571832

070	095	log	18.0506875409959	10030.6313533365	0.777983133828272
070	100	log	9.65919718563004	10506.3514564913	0.513268163508939
075	001	log	-15.1519062144605	7540.85389998015	0.581926643475933
075	005	log	-9.95303504077414	7520.1928814902	0.757681067571623
075	010	lin	-0.000638950706348817	7533.84386613403	0.355273963404095
075	015	lin	-0.00225764037698507	7623.19285379189	0.551734666624417
075	020	log	-7.12785754255058	7795.82918164912	0.409653734616279
075	025	lin	0.000698836100859196	7852.8545093879	0.662425481229277
075	030	lin	0.0025140594404767	7937.70389397883	0.824962482326785
075	035	lin	0.000145827833331625	8099.01904466668	0.0736648445251391
075	040	log	5.11862441213899	8213.85619341745	0.625209593769872
075	045	lin	0.00118302451346788	8384.24795931482	0.604387024089761
075	050	log	-2.57082649303354	8634.38982793366	0.31870949572648
075	055	log	-9.62150510028724	8804.75085144055	0.702963483835526
075	060	log	1.27086505824903	8906.87423375189	0.109440082926527
075	065	log	9.4110063556265	8986.02951625914	0.670954107601235
075	070	log	5.23637198652574	9229.17088626146	0.526466654497208
075	075	log	3.6221723380345	9449.62141067691	0.373193438483142
075	080	log	0.825179340252055	9662.36785485129	0.115356301327869
075	085	lin	0.000347221801587939	9833.11273015167	0.184354703718613
075	090	log	7.22079756139674	9979.35904295473	0.734672942632118
075	095	log	10.9558570748121	10109.8751561399	0.746096211065318
075	100	log	20.084605452087	10429.8444807665	0.783637453402629
080	001	log	-32.9140492732532	7757.93626913136	0.694522471530906
080	005	log	-7.50016417886973	7564.67956842711	0.594687145620669
080	010	log	-3.15248140307387	7632.15079315571	0.210164453778767
080	015	log	-10.4393771451562	7780.6248602385	0.696666396955945
080	020	log	-7.93634609721397	7882.8997420833	0.415375910956028
080	025	lin	7.4356174604098e-05	7934.37961164726	0.0311089972748325
080	030	lin	-0.000392322475525612	8039.73036448769	0.160362422664613
080	035	log	-5.96073900741089	8230.10991802147	0.338529838202146
080	040	log	-1.90943492101159	8339.40383131929	0.131197537256977
080	045	log	-2.33001185092104	8473.5046625223	0.141453143121704
080	050	log	-4.82293821297622	8711.7151095489	0.282212450035393
080	055	log	-3.8064330795305	8811.33383155937	0.247756563320783
080	060	lin	0.000162032330964169	8958.10926946986	0.143550493338512
080	065	lin	-0.000319991461759625	9114.30895477777	0.132472569808012
080	070	log	6.80578049418519	9252.34444219729	0.691379924507711
080	075	lin	-0.000234007224343858	9518.40068663422	0.159766058966081
080	080	log	5.13943179084528	9652.99385762202	0.424174557800312
080	085	lin	0.00113899936904808	9854.54811374603	0.634086091977238
080	090	log	8.26733597548987	9995.77863703039	0.662680891868777
080	095	log	3.55644653868997	10197.6565367993	0.481644733621823
080	100	log	10.0483869266308	10538.1900772363	0.672350361589012
085	001	log	3.28020581867751	7518.88808806193	0.111387266628501
085	005	log	-11.6021198208802	7681.71783325541	0.75898130475618
085	010	log	-24.4995167576594	7912.15132753581	0.628528696905083
085	015	log	-34.980168500225	8087.41146763809	0.887573201644067
085	020	log	-12.535132227515	8016.06605196518	0.885945249102469
085	025	log	-10.1100804249133	8120.3325812542	0.773050776915354
085	030	lin	0.000153616678939806	8135.19410407953	0.191986867817221
085	035	lin	-0.000484271734128396	8273.4427582575	0.191789411243387
085	040	log	9.93713287164483	8330.54664530197	0.634815015779665
085	045	log	-3.87062621312823	8573.08111246241	0.415424609734125
085	050	log	-3.97936926201353	8773.82404398061	0.229454479136204
085	055	log	-2.30783920937466	8865.10797513034	0.203758918440249
085	060	log	-5.05556353714431	9068.34953410817	0.369975995562763
085	065	log	4.67689854510359	9126.43706176385	0.368657943554267
085	070	lin	0.000386061834295591	9358.36023079629	0.138518198498046
085	075	log	5.54065501981797	9513.48782891688	0.436998692746458
085	080	log	-1.4163421633077	9750.07016855524	0.131911513697877
085	085	log	7.97237076060375	9828.16866663074	0.602211489639103
085	090	log	13.000949478212	9987.43336338626	0.869548140597995
085	095	log	6.66382472602344	10205.8684976658	0.41118466982009
085	100	log	11.6173306620969	10550.7744994314	0.650794028547713
090	001	lin	-0.000578399539870875	7621.95226586829	0.0898035068464158
090	005	log	-30.9124424712959	7939.60238537519	0.972767608109004
090	010	log	7.72975687473599	7727.28127616768	0.246929177840566
090	015	log	-22.8225818566287	8079.99232198047	0.826124858213079
090	020	lin	-0.00175304914304614	8023.46688412226	0.441074915149349
090	025	log	-10.220222229254	8234.04315903052	0.431600263290877
090	030	log	-5.91732208332797	8296.71873600084	0.376728596029771
090	035	log	-17.2174155819623	8533.94326792228	0.857713870198285
090	040	lin	0.000225943194323598	8523.72859218519	0.0428479161825458
090	045	lin	0.000956061011903485	8638.3246957672	0.348568705464902
090	050	lin	0.00105707848412693	8826.87551966843	0.308310835549718
090	055	log	5.31436933623246	8897.13808135895	0.33225758473447
090	060	log	-2.12104563620072	9123.66758292533	0.13436992534028
090	065	lin	0.000780445756371816	9243.0238487963	0.246585640820971
090	070	lin	0.000300747420394059	9428.42391714815	0.163127742307562
090	075	log	-6.01255756025233	9675.53018119225	0.511992287779274
090	080	lin	9.7165531400969e-05	9794.40947838164	0.196902370124713
090	085	log	6.83509761192885	9888.79590561282	0.512764723515936
090	090	log	12.5629959898316	10038.4405009201	0.74755016384527
090	095	lin	0.000533202750001351	10300.6470045185	0.195916150944476
090	100	log	11.4407428490535	10589.6398203081	0.627626588588392
095	001	lin	-0.0036653550595258	7683.08632351192	0.569183946556647
095	005	log	-40.236908641064	8072.01792470332	0.8516908529109

095	010	lin	-0.00235632368953171	7853.98749426965	0.391473108915195
095	015	log	-22.8696687589919	8133.72317708909	0.56578347410512
095	020	lin	-0.00421277171768461	8104.82411062924	0.816516545588685
095	025	log	-24.5080041276547	8412.55850177045	0.721201697328905
095	030	log	5.0263019526744	8262.54127776055	0.44875877530777
095	035	log	-11.509008436814	8549.30711377539	0.678535537141023
095	040	log	-16.4673497989779	8734.78525667058	0.615671084291452
095	045	log	-5.38193374590655	8764.19503970952	0.529457158624621
095	050	lin	0.00273347063095172	8884.18355143916	0.548125668120633
095	055	lin	0.000377593539682031	8998.0099551993	0.0760835829873701
095	060	lin	0.00108191486111141	9162.07062986111	0.313030828722357
095	065	log	-6.34600355843781	9363.85177085349	0.419283486887843
095	070	log	-4.57317340003826	9530.36079065143	0.673386609401247
095	075	log	-8.99177752832117	9753.23306075808	0.417991728375832
095	080	log	6.90061432353715	9789.49231687158	0.357060398747723
095	085	log	-1.2141821605451	10010.2012551895	0.359397435244265
095	090	log	8.98200390689335	10118.0515452943	0.431373658341627
095	095	log	-2.23877929961073	10368.6806032699	0.11492834966581
095	100	log	8.89320563625236	10653.0260075154	0.450928542191188
100	001	lin	-0.00093459434098573	7651.82761826105	0.135707089320615
100	005	log	-32.7680257849336	8000.5556034004	0.905344514577028
100	010	lin	-0.00018993432653163 2	7857.46107689797	0.138778701412203
100	015	log	-32.6609019755858	8230.69069322415	0.910073980582548
100	020	log	-31.5106899149185	8356.97023460246	0.813806642576181
100	025	log	1.64699648130601	8191.17342473732	0.0861218457961218
100	030	log	2.56071445850406	8293.10784913006	0.0879708767028745
100	035	log	-11.3713192709102	8562.52183748079	0.633468204504339
100	040	log	-20.5430145017489	8782.74794512278	0.870942921895185
100	045	lin	-0.000670170685046206	8737.19134416876	0.647376840078664
100	050	log	12.540701906143	8805.25106491244	0.419511899359629
100	055	log	7.50393499782559	8954.84955466905	0.294784130119446
100	060	log	-22.6890930421763	9393.57563363777	0.732930304818632
100	065	lin	-0.00147549169398698	9345.23657034607	0.501285146674655
100	070	log	-7.84704012089018	9587.57166908485	0.442197827983684
100	075	log	-11.1188244079306	9800.37570292439	0.375142370672108
100	080	log	-2.58623232146157	9903.07046498813	0.138709336683192
100	085	log	-21.7448776578873	10227.1525773655	0.802213036574159
100	090	lin	-0.000322001736046614	10235.5399068195	0.108022264550961
100	095	lin	-0.00208702515392303	10400.5954542593	0.370897137161457
100	100	lin	-0.00172149015584714	10782.380102	0.41313466531188

Appendix 5.2 Usenet-Based System

001	001	lin	87.0413087282309	-22960.1274048889	0.999833243887559
001	005	lin	22.5890586775194	-28458.0304589998	0.974872316648905
001	010	lin	24.6407152705935	-34866.8769305185	0.969492742929493
001	015	lin	26.9493570097234	-42405.7614935184	0.96118215943756
001	020	lin	23.3334266494088	-27105.3866604074	0.976560072881476
001	025	lin	22.0042616358754	-26055.0310620371	0.974123341140279
001	030	lin	24.5154216690055	-27241.0532721295	0.967042623594582
001	035	lin	36.9522963560087	-69927.1375193334	0.930244460887441
001	040	lin	23.012835740803	-18315.6674001664	0.973349050796604
001	045	lin	22.4518421530274	-11924.9563875556	0.980350455146813
001	050	lin	22.2172841874928	6962.48432027778	0.996242276436612
001	055	lin	20.4542153235159	-725.769104611085	0.998729058902352
001	060	lin	19.9361014500489	13840.0850642036	0.996896776189267
001	065	lin	21.3635758860674	2980.21705207389	0.998255633201801
001	070	lin	20.7210171896647	8229.71394507409	0.999295868155839
001	075	lin	22.9454795402578	12572.2163527406	0.999476841492287
001	080	lin	22.7448494897624	14084.0505534815	0.998883349507683
001	085	lin	21.5755386095279	11493.572838759	0.99976575281058
001	090	lin	21.9063183562855	13957.5293950927	0.999305422329458
001	095	lin	21.9789147672641	15616.6791073335	0.999190466889469
001	100	lin	34.4736091161017	9254.63796183339	0.998387920421606
005	001	lin	74.7815962427915	-20429.8779577224	0.999080531839791
005	005	lin	22.9114057840645	-29206.3149248149	0.976337562701961
005	010	lin	24.0161438924055	-32304.4793541112	0.975677559030971
005	015	lin	26.6773071876849	-41034.8225647037	0.963220824222764
005	020	lin	22.8730605304704	-25085.1227951111	0.978685532215457
005	025	lin	21.0904305616849	-22805.6536827037	0.981004346061459
005	030	lin	24.7600958947172	-27746.5723321111	0.966216292219486
005	035	lin	36.6838244549332	-69261.8541572595	0.933667100892443
005	040	lin	22.5195933722922	-17476.4525715	0.966285707008876
005	045	lin	22.016824900709	-10022.5941049629	0.988972668065007
005	050	lin	22.6734967092087	5299.82496462969	0.996285991562923
005	055	lin	20.4619598672852	-620.007212814846	0.998872462898766
005	060	lin	20.0578793816479	13751.6202575558	0.997196703152038
005	065	lin	21.4395800418535	2968.70564338889	0.997656551384386
005	070	lin	20.9120447231337	7544.21654351855	0.999493562443265
005	075	lin	23.0745425831148	12137.2962218332	0.999512720052707
005	080	lin	22.7952095358456	14128.8215054444	0.998830894768666
005	085	lin	21.6751312909199	11243.2402108333	0.999834666480006
005	090	lin	22.1159698250642	13471.2585172778	0.999419041207388
005	095	lin	21.9298109750606	16301.2132836668	0.998965360915201
005	100	lin	34.5917611447369	8823.8821922965	0.998418976949777

010	001	lin	70.3308598820757	-11762.3939581665	0.999236020499073
010	005	lin	21.5712731519502	-23883.0879618333	0.983540564543229
010	010	lin	23.4982195301202	-30394.7198762962	0.977931890193447
010	015	lin	26.2307603060382	-39744.1182877222	0.960765646551069
010	020	lin	23.0077409070269	-25373.5983673702	0.976156253564751
010	025	lin	23.2276966310572	-29719.704321037	0.965281583059707
010	030	lin	25.2573991519769	-29427.644011111	0.958199554111294
010	035	lin	35.5025865677076	-65890.2856194074	0.925758143282611
010	040	lin	24.7901281369252	-23969.1664912037	0.95948793023157
010	045	lin	21.7300992555423	-9261.06382879634	0.991768025902517
010	050	lin	22.5657224006318	6138.05281375934	0.997921458572531
010	055	lin	20.238804982973	994.662757370475	0.998823229993569
010	060	lin	20.4590067531926	12506.5972058334	0.997771252333982
010	065	lin	21.824678105981	3142.5376563148	0.997872565789107
010	070	lin	21.4497397400693	6465.84094133342	0.998880264923364
010	075	lin	23.2215439077463	13067.4705366852	0.999308634366783
010	080	lin	22.7861446191775	15396.1242966667	0.999140418414582
010	085	lin	21.9266681117309	11142.1909506111	0.999748326681937
010	090	lin	22.2001292381008	14241.3982442037	0.999537707827235
010	095	lin	22.1994756514582	16034.1723616109	0.999782126135316
010	100	lin	35.162075124936	7197.8673636298	0.998048357947532
015	001	lin	71.9178914108401	-5894.676230426	0.999426989248021
015	005	lin	20.3792427414793	-19240.8664885372	0.985347718168066
015	010	lin	21.1270996836313	-24173.8369490555	0.978478719067787
015	015	lin	23.1597150415647	-29703.4609164075	0.970698997611656
015	020	lin	20.6065659886917	-17678.0904462962	0.981997561978108
015	025	lin	21.6096502691087	-23984.7560928518	0.976967532370655
015	030	lin	23.7805956103468	-24652.0766415187	0.974555936961848
015	035	lin	24.4075128183002	-29393.8804381238	0.932838761359776
015	040	lin	23.2463670794925	-18441.5928746296	0.973371329263019
015	045	lin	20.7537251879815	-5269.52327587039	0.996143366590246
015	050	lin	21.9574210876654	6599.43173179649	0.998137736206312
015	055	lin	20.3767023718057	2157.16888281479	0.99936222378483
015	060	lin	21.0957522918968	12338.7335567222	0.997570733343862
015	065	lin	23.0395977464055	962.338387888682	0.995771201786111
015	070	lin	21.6124684206515	8389.99216816676	0.998965058051519
015	075	lin	23.6172188387571	13363.4543705185	0.999320590038003
015	080	lin	22.5286852419887	17402.8247733521	0.998720288628632
015	085	lin	22.5295754099877	11343.4428137223	0.999373059546968
015	090	lin	22.7625729245861	15134.0216333517	0.999438435871637
015	095	lin	22.9020689481681	16770.8018282777	0.998930119295475
015	100	lin	35.7228977020303	7081.88474033322	0.99792767402645
020	001	lin	75.1247773286362	-8613.14754590759	0.999821812382851
020	005	lin	20.6509784595957	-18884.9746153518	0.977058154954611
020	010	lin	19.8395260350098	-20284.2533377962	0.977343182219039
020	015	lin	19.9831638866154	-20711.1146099444	0.968635776099017
020	020	lin	20.1926412894029	-18419.4590130927	0.984911972184628
020	025	lin	19.8316088732424	-18573.2686563333	0.978603559762158
020	030	lin	22.7389864002287	-20962.7216870554	0.980965739534406
020	035	lin	23.6339396892376	-29727.6520264814	0.932784449814042
020	040	lin	24.1562367119798	-21059.8440402221	0.962854347355322
020	045	lin	21.2332866585486	-7956.01208620374	0.985805716735921
020	050	lin	22.5163708464057	5315.11615879636	0.996997965511531
020	055	lin	22.8008928390431	-4909.73688057414	0.995457135071107
020	060	lin	21.8400770368603	11207.6045637963	0.997537510504173
020	065	lin	25.2747497472287	-4619.97753605555	0.991181925348102
020	070	lin	22.6668215028903	7231.50474522232	0.998485164054842
020	075	lin	24.293609159215	13615.5005982223	0.998302347150575
020	080	lin	22.1254071913432	20116.6057878704	0.997577367496421
020	085	lin	23.6354670959295	10427.5541521296	0.998930730940091
020	090	lin	23.8907048747061	13375.9846661479	0.998844297772537
020	095	lin	23.3188844627569	18028.155131611	0.998328247468084
020	100	lin	36.5186382742713	3287.9025055556	0.998190109933325
025	001	lin	77.6525602760449	-6587.69653231424	0.999659777785668
025	005	lin	19.6359995299882	-14422.945264463	0.981644424464748
025	010	lin	17.5015324559433	-12395.5264861482	0.990007783993057
025	015	lin	16.8502112263052	-10389.4335804999	0.984790052997682
025	020	lin	18.5484715038624	-13778.575047037	0.981796759521984
025	025	lin	19.0668868826299	-16409.0182074999	0.979406850638952
025	030	lin	21.8141191587307	-18698.6524692964	0.976557809826618
025	035	lin	19.3170942968206	-15123.2679159258	0.95961485531628
025	040	lin	21.5719213146101	-12018.1358659074	0.98353599915785
025	045	lin	22.4640887199943	-11298.7485617778	0.98523664126636
025	050	lin	22.8134916314853	2671.40373414815	0.995447248984152
025	055	lin	25.9138007381631	-13317.4471707779	0.988569240972601
025	060	lin	23.1691480609006	7561.9392482409	0.999092126746638
025	065	lin	27.0324606511933	-8965.94076744423	0.990295016241533
025	070	lin	23.5477520134142	4920.29399155545	0.998572785130777
025	075	lin	24.5972618435389	12188.0542825002	0.998996102808289
025	080	lin	22.8692241707626	19679.4885573889	0.995994182778382
025	085	lin	24.9426044941813	7820.7680861851	0.998890938970484
025	090	lin	24.9558663461143	12541.0240990184	0.999026139131145
025	095	lin	24.5148427789464	16118.1888746479	0.998568425940546
025	100	lin	37.0966980954027	1250.54717399963	0.998864434159166
030	001	lin	80.0199468710466	-9446.11220096244	0.999609661900025
030	005	lin	19.0808160980407	-11088.5209826482	0.986216266369886
030	010	lin	16.2164062951349	-8334.97854594437	0.99247397059664
030	015	lin	15.4953687336535	-5632.33811557409	0.993061997215528

030	020	lin	16.7651382585649	-9708.09844449995	0.980073761905553
030	025	lin	18.498235483779	-13757.1687519074	0.9884662114552
030	030	lin	17.6845281382196	-6837.07658753697	0.990910209640594
030	035	lin	16.7050550109632	-6483.27974583319	0.974806551637394
030	040	lin	21.336544851968	-10089.2661146851	0.993929938029434
030	045	lin	21.4959854004851	-9408.86823975925	0.983130332934079
030	050	lin	22.9098537539231	1918.13863162938	0.997066090878847
030	055	lin	25.5002018551731	-12408.6707516666	0.984414196854998
030	060	lin	24.4826125145918	3552.13374512989	0.998062838127362
030	065	lin	27.0234017936556	-7476.7711974073	0.993747059002828
030	070	lin	23.900171275252	5555.19856596299	0.998375466143531
030	075	lin	24.8549713089355	12709.7364338148	0.999020193697105
030	080	lin	23.2251739115621	19594.3913936111	0.996184345815251
030	085	lin	27.1682998296241	1676.00934172206	0.993682606843445
030	090	lin	26.0673002485036	9960.32942411123	0.9986587125286
030	095	lin	25.1950803627083	15282.1900683148	0.999174359038769
030	100	lin	36.313002250399	4805.90986538905	0.998552729743006
035	001	lin	84.9265716447444	-10126.1912295746	0.999604633759857
035	005	lin	18.8528471424072	-9399.99494775927	0.990860640829129
035	010	lin	16.4853582983165	-9660.26631185188	0.986260147311834
035	015	lin	15.9462464998062	-7207.3876513704	0.98790027022074
035	020	lin	16.3634233750431	-7987.66549257422	0.982082125983518
035	025	lin	16.7727487211515	-9507.9382153334	0.990020957499961
035	030	lin	17.9804147405604	-7859.82308574073	0.992141598362432
035	035	lin	15.5244787113716	-1779.90872505562	0.993553445055255
035	040	lin	23.065212484133	-17607.1317562037	0.961149741139474
035	045	lin	23.6918156288249	-15749.1857235926	0.973119037311198
035	050	lin	23.5239459071783	-555.150706611326	0.995534991526701
035	055	lin	28.5796580225103	-22326.1116484814	0.97649192383138
035	060	lin	25.8997775111989	1701.73596342583	0.992842957119881
035	065	lin	31.298814940957	-18890.6473794259	0.975921182684014
035	070	lin	27.8855773479117	-6738.31803901831	0.982497456849883
035	075	lin	25.9035493618884	10940.4279049627	0.999107406287602
035	080	lin	23.8268064037184	18650.5477394258	0.998075390739627
035	085	lin	27.358836703013	4207.89376799995	0.992000519662508
035	090	lin	26.5911108274268	10828.2975316485	0.998619201861247
035	095	lin	26.9920277964844	10737.7726955183	0.998486005569119
035	100	lin	37.2262527820481	444.886987481557	0.997471106286904
040	001	lin	86.4976104824727	-5722.45022094488	0.99923774387215
040	005	lin	19.8165316047609	-12153.6013169629	0.987450756574088
040	010	lin	16.4687248451164	-9617.57875581486	0.986952124398726
040	015	lin	15.6645443893985	-6213.59328942596	0.990794948888914
040	020	lin	15.7993602744536	-7212.6147466297	0.989788092238001
040	025	lin	17.0628284183617	-10509.1376412591	0.986993283737692
040	030	lin	17.823103570354	-7590.93417429616	0.990325244789871
040	035	lin	15.7007628676169	-1912.3595715	0.995042059083523
040	040	lin	21.6446737969983	-12415.7030153521	0.98946972982118
040	045	lin	23.0373361700584	-12742.9201794999	0.981488346465496
040	050	lin	24.371284282159	-2760.87516420381	0.996596459777749
040	055	lin	34.8913499038359	-40318.1312468516	0.97433846114075
040	060	lin	30.4361326180351	-13807.9815935184	0.98047182926387
040	065	lin	38.5263624454856	-44304.1940729444	0.952490865021779
040	070	lin	32.8130979721525	-21251.6763287039	0.976220342377318
040	075	lin	26.7371467782232	9701.37915607422	0.999260106391315
040	080	lin	24.6631266791854	18331.1184966112	0.998202675206365
040	085	lin	29.418563775493	-1471.27288481477	0.993018576476274
040	090	lin	27.6474960297924	8802.27135690753	0.998579411978262
040	095	lin	28.7471575590647	6387.22130109239	0.99698328231138
040	100	lin	36.7352734498369	2067.71397477784	0.996994129130136
045	001	lin	86.8644920948899	-1814.74710059306	0.999159094575842
045	005	lin	20.1845948175334	-12149.3101063704	0.992483076720705
045	010	lin	16.731599247341	-9782.15716229629	0.986521393149603
045	015	lin	15.4539952463379	-5251.3394570926	0.993003783915645
045	020	lin	16.0527546229057	-8080.59021670373	0.986712256825596
045	025	lin	17.9297310921214	-13028.7583957592	0.980300292271873
045	030	lin	17.873153689025	-7314.81598862962	0.988964565002381
045	035	lin	15.9057710192258	-2495.9919509444	0.993844864334188
045	040	lin	22.1551095238468	-14433.9459760185	0.985803486488935
045	045	lin	23.3898350255609	-13311.911818926	0.969960939485876
045	050	lin	24.7593457772806	-3292.10187005549	0.99725792890157
045	055	lin	35.8543585990799	-45833.0033317406	0.947835686593563
045	060	lin	31.7036291097626	-17509.7195956109	0.985304923810582
045	065	lin	37.7080864994972	-41899.344007389	0.944464249949699
045	070	lin	30.6173668933139	-12480.2986968334	0.990439694011278
045	075	lin	27.8243351009514	6382.22346970378	0.998779777957296
045	080	lin	25.4652671509702	17132.4383704817	0.998205124613364
045	085	lin	31.3539002878658	-7178.76154233326	0.988466372903642
045	090	lin	28.9407288614721	7595.1880642407	0.995230720084597
045	095	lin	33.0154144872078	-5867.16284500004	0.982923464180764
045	100	lin	37.1750639376922	-389.537507481436	0.996195316781246
050	001	lin	94.005235792571	-7904.87954981514	0.999431008025618
050	005	lin	20.9552262971171	-12774.3443670926	0.992538119673079
050	010	lin	18.5012899923663	-15748.1226920186	0.975567415109614
050	015	lin	16.4168717097741	-7804.56995205539	0.989472565257986
050	020	lin	16.2622282874384	-8504.11130479636	0.989272466361465
050	025	lin	18.0698481118771	-13018.6524976852	0.985566057556294
050	030	lin	17.66592726131	-6679.14834735176	0.990406914656108
050	035	lin	19.7740398115325	-11396.7055879999	0.953099067340413
050	040	lin	71.54428893564	-165716.074340389	0.904355301927566

050	045	lin	68.9968689396414	-160856.856222945	0.88428844373171
050	050	lin	35.3264283267874	-37312.1118311482	0.974732472214071
050	055	lin	57.0116560403971	-111104.25983187	0.923210792830234
050	060	lin	55.1350661404639	-89586.7398596112	0.897181198858492
050	065	lin	27.9086070768282	-4089.37542147102	0.993972292650308
050	070	lin	107.147958070359	-253554.465834334	0.881869390207111
050	075	lin	30.2659401939591	1009.31502774087	0.998329085543607
050	080	lin	29.9882112752992	4362.54478381487	0.998060871524094
050	085	lin	38.1064249478869	-18994.4762658036	0.927189630365904
050	090	lin	45.7458296794173	-46036.6127086481	0.919210850654291
050	095	lin	40.283808192347	-26175.764684611	0.969808487379998
050	100	lin	35.7820938110282	2805.48739916639	0.996774623189598
055	001	lin	94.9687298089182	-2794.36441151821	0.999594309235359
055	005	lin	21.3260117381753	-13573.5424144999	0.987396627991032
055	010	lin	17.7270779873684	-11702.3055988517	0.985595543693174
055	015	lin	17.0209700155991	-9137.79599464807	0.990125702586537
055	020	lin	16.1520172254064	-7478.75174148145	0.991435802597174
055	025	lin	19.0463234343283	-15293.3544893888	0.987065365476165
055	030	lin	18.1434444750878	-7114.38428379636	0.993645800298765
055	035	lin	17.2885532910503	-3011.28735735195	0.966965116302804
055	040	lin	44.5842227543287	-74769.0095507485	0.918932491986999
055	045	lin	74.2098615693416	-178475.454974482	0.879409540922642
055	050	lin	39.9454650001472	-52716.5411646667	0.957326883517613
055	055	lin	62.3688506665887	-120848.447460667	0.928227740180671
055	060	lin	52.4995784586741	-83528.986512537	0.942051904209191
055	065	lin	45.1904762596197	-56032.9799020358	0.936623265500741
055	070	lin	43.0471692879805	-43565.3281541114	0.935555312340794
055	075	lin	47.1906428588178	-46368.1285527771	0.912367649298508
055	080	lin	51.5438227410866	-58263.3699003333	0.940636915829306
055	085	lin	72.9903380473313	-127090.731670876	0.898061421744867
055	090	lin	53.5378260092361	-55716.3656251619	0.892784780907221
055	095	lin	43.993115127834	-31490.7196807265	0.963133401310182
055	100	lin	35.6207480738841	3068.48348462972	0.997321199095616
060	001	lin	97.8410761024035	-5721.3513313703	0.999624881792919
060	005	lin	21.238605491652	-11595.7428470185	0.991051233227153
060	010	lin	18.4588575239199	-13629.3114801665	0.982699438693393
060	015	lin	17.1127709229456	-9069.90296607408	0.990016985008916
060	020	lin	17.7492858324594	-13000.9748708519	0.979805768575689
060	025	lin	19.4761124638305	-16735.0518377222	0.983520216333082
060	030	lin	19.962297351209	-13449.540578463	0.979053381531181
060	035	lin	16.7089225196859	-3916.90522507398	0.994032671408553
060	040	lin	41.9133526388095	-70312.2428657936	0.877409498865859
060	045	lin	82.6235648264375	-197914.744181426	0.885222191313287
060	050	lin	46.9878622765551	-70964.5000977037	0.945958236542229
060	055	lin	77.0754083815966	-166690.098425722	0.92380228100721
060	060	lin	62.8590372606846	-113099.278046611	0.942296862267524
060	065	lin	72.2640451519153	-149011.429963407	0.902804558604348
060	070	lin	106.152133903055	-235736.897970463	0.81099103579589
060	075	lin	79.4393834558102	-142752.763197944	0.896765438423747
060	080	lin	64.8728742558031	-99556.2951251669	0.900004801306005
060	085	lin	69.463319295034	-97773.9591190817	0.911460443419041
060	090	lin	58.2403932899107	-66336.0509827677	0.842373700464872
060	095	lin	89.7683006595735	-178401.167692854	0.84447982767563
060	100	lin	35.1570398692518	5051.01799705581	0.998052275425754
065	001	lin	100.033497993787	-5300.88401287061	0.999476807307554
065	005	lin	22.3521748082126	-13914.0855738519	0.986762554207044
065	010	lin	18.71631570824	-15072.5105904075	0.976093479125931
065	015	lin	17.0347638558008	-8585.40154333327	0.989483668511116
065	020	lin	17.8215374172739	-12834.271055463	0.981854725036252
065	025	lin	19.7403907892184	-17318.4916590741	0.978662770456155
065	030	lin	19.0073196623004	-10281.9402506481	0.980377507719637
065	035	lin	16.3093993445172	-1344.75181971991	0.99459464822573
065	040	lin	45.9075550574372	-77664.2532482134	0.930901808345169
065	045	lin	75.930716707086	-172350.375464148	0.896510045472147
065	050	lin	63.6861822617246	-123854.867582981	0.926363491153765
065	055	lin	84.3393325801525	-192155.385344704	0.909914444286797
065	060	lin	72.9069133378927	-146553.981390704	0.918310338317985
065	065	lin	84.5326984403485	-179967.141935167	0.925490780264684
065	070	lin	137.828327884061	-331653.980050165	0.841556534083594
065	075	lin	106.655814316379	-235724.015501926	0.894179931700677
065	080	lin	99.3363691977819	-200021.829979019	0.919202029543476
065	085	lin	61.105023885102	-68428.0530155781	0.851297855937261
065	090	lin	109.844099616956	-216994.321324148	0.876458635394806
065	095	lin	120.225884889891	-267345.944518199	0.848028081804448
065	100	lin	34.7996909145438	5986.58656364784	0.997268911245972
070	001	lin	102.920684934232	-4047.09499053657	0.999701370171398
070	005	lin	23.5151594202975	-15825.665167537	0.988959647947538
070	010	lin	19.2061999024024	-15845.0486659076	0.977488122222534
070	015	lin	17.7135563770406	-10055.7128356481	0.989695205827209
070	020	lin	17.2575345117018	-10194.8346491851	0.987603845457665
070	025	lin	24.9305344960863	-34469.8329962406	0.948082057722037
070	030	lin	22.4447929172027	-21211.6225240555	0.961258511395226
070	035	lin	16.512627456918	-1440.2782604807	0.991981679583106
070	040	lin	54.3399673110979	-102287.340652509	0.892387362129329
070	045	lin	90.9428352274652	-219826.059814074	0.90542110277976
070	050	lin	49.8947633249319	-67369.9497635033	0.924121792081378
070	055	lin	110.966372579675	-263085.115615908	0.854396461027665
070	060	lin	88.8376931081895	-185329.306920963	0.937002255134303
070	065	lin	110.78550931731	-264358.050159352	0.901699307694902

070	070	lin	159.949356783998	-380330.686560876	0.88353442055104
070	075	lin	129.010626047454	-279255.101404985	0.898504321522767
070	080	lin	226.903327099646	-566354.552746611	0.923023545662513
070	085	lin	133.250626342764	-251812.923622882	0.919466982194781
070	090	lin	231.91390065203	-586494.52280239	0.904325170291447
070	095	lin	137.588461341531	-313949.722178586	0.842227062755223
070	100	lin	34.1134775101008	9710.94658020372	0.99761773453477
075	001	lin	104.31084873925	-3100.23575111083	0.999041531388576
075	005	lin	25.0540997643204	-21290.0775593334	0.975052675683318
075	010	lin	20.686766779651	-20164.7932666481	0.964451276632779
075	015	lin	18.7189441407869	-13149.877470963	0.98278805988137
075	020	lin	18.9485223203299	-14363.8178550369	0.978248266749315
075	025	lin	23.9031685871022	-28078.3264333519	0.973706019669468
075	030	lin	22.1743291986967	-20104.3074672407	0.95729789619348
075	035	lin	20.1897603550094	-14657.0951796111	0.948614717911856
075	040	lin	49.2434518666105	-84070.8035885328	0.936026527044604
075	045	lin	107.850160432352	-272084.409164648	0.886342165533512
075	050	lin	82.8676742551422	-163022.687860345	0.904207592408866
075	055	lin	103.702497189548	-237247.386446833	0.932433607779714
075	060	lin	112.505369767595	-253787.537853167	0.927209909061548
075	065	lin	106.052714876128	-232428.161989259	0.939341412030037
075	070	lin	198.34985022917	-474154.53753141	0.893946877557232
075	075	lin	296.041580895395	-816201.778175036	0.8875386793406
075	080	lin	227.807267841329	-574356.499706196	0.841516601579808
075	085	lin	369.173190113996	-1049354.67550343	0.876924087000013
075	090	lin	293.780964094903	-776068.7462625	0.90633002796868
075	095	lin	360.680571383248	-1014922.15258637	0.878213624353302
075	100	lin	33.9567881728684	11456.2118426482	0.998340523523255
080	001	lin	107.311229406664	-4652.06643573998	0.999365028560524
080	005	lin	25.9021891243771	-22658.8780051853	0.979907613999859
080	010	lin	21.2543770230067	-21115.6407355926	0.962891043260714
080	015	lin	18.846068918627	-12760.7485303891	0.985480957582931
080	020	lin	19.125157108634	-15085.8260040742	0.979879593339117
080	025	lin	24.6379509901167	-32126.8407599074	0.963762644022064
080	030	lin	22.0173966704382	-18925.3481167035	0.970661061393968
080	035	lin	16.6180404201158	-1143.72562073308	0.994611665596106
080	040	lin	44.3496587236839	-62258.5441966645	0.913088898977279
080	045	lin	102.668203156358	-255231.87409087	0.878413887334354
080	050	lin	92.3047337762059	-201953.336929708	0.912203806224974
080	055	lin	111.860487589115	-264159.703040167	0.91650374959632
080	060	lin	117.839342578279	-269680.709191592	0.932460835960981
080	065	lin	112.119435319758	-258452.512331037	0.914426839385428
080	070	lin	242.334953463707	-644839.3716857	0.823022191073024
080	075	lin	303.146684728713	-849252.324284628	0.857775575299348
080	080	lin	287.187298171625	-783566.360606832	0.880584288352621
080	085	lin	264.919252344553	-681257.102348838	0.836125125621018
080	090	lin	285.080166021629	-763993.481155037	0.900318151314784
080	095	lin	332.905741042557	-914781.10283426	0.883105040231175
080	100	lin	35.672634153157	8705.36099153719	0.998387400203465
085	001	lin	109.403386347289	-4971.01940738759	0.99943081756858
085	005	lin	27.3842662962932	-26532.4184158705	0.974982070107368
085	010	lin	20.5315874640974	-18172.2818305	0.980671028879501
085	015	lin	19.6802580264399	-15174.0254523518	0.981327563297593
085	020	lin	21.7262628393817	-23591.6561429443	0.965325609456149
085	025	lin	23.6591014571779	-27743.8064675184	0.967605025791681
085	030	lin	21.9427565153571	-15623.3633150396	0.965193334516358
085	035	lin	17.0292545416998	-1826.3841663207	0.990457784255625
085	040	lin	76.6777031970507	-180845.158877537	0.884106248636181
085	045	lin	105.389899043585	-259211.940343278	0.888336303782485
085	050	lin	119.333390885933	-280550.017659796	0.859068243334928
085	055	lin	138.484774166233	-329048.763481814	0.899064363560242
085	060	lin	131.804254944991	-311387.871738574	0.926114412249798
085	065	lin	144.490267963805	-356063.072664815	0.907171874513552
085	070	lin	279.236182515516	-745934.836202711	0.835737287261005
085	075	lin	345.606053203841	-964195.648579612	0.8622887251655
085	080	lin	345.909609729909	-955610.885482093	0.893687773148817
085	085	lin	442.791869446442	-1297660.534616	0.858724651126497
085	090	lin	305.973297919251	-825825.285825667	0.886333100011047
085	095	lin	225.514505441848	-529512.744671706	0.882604872244045
085	100	lin	36.6509031015353	7664.11181288923	0.998728336285506
090	001	lin	111.184077109222	-270.777775462368	0.998542241614729
090	005	lin	29.06051781986	-30378.8878881112	0.970656576621045
090	010	lin	22.1543345725166	-23033.698854889	0.967552021206778
090	015	lin	19.7472164355087	-15211.0665358333	0.977233004330863
090	020	lin	20.2800349441224	-17842.4901361295	0.975864303119825
090	025	lin	28.7442631702006	-46141.7627532222	0.906822120416085
090	030	lin	19.7441706103605	-8955.3862989496	0.981156627480677
090	035	lin	20.4368492938194	-11116.042136412	0.965447516353284
090	040	lin	74.266524287102	-171645.349724259	0.899761031559719
090	045	lin	109.516052699071	-274817.741906223	0.877465644659625
090	050	lin	220.805032235438	-608494.540500796	0.878281067795711
090	055	lin	127.632286891328	-306980.138199297	0.931220473713977
090	060	lin	145.231890708363	-332953.737745815	0.93641110123389
090	065	lin	133.997823066254	-311127.14607287	0.926470855400808
090	070	lin	426.844991028496	-1257551.20515402	0.856932255511786
090	075	lin	399.668014914395	-1135086.53421713	0.880801968202254
090	080	lin	390.172028181116	-1077103.25072963	0.901092225029527
090	085	lin	145.848012002015	-295908.441828087	0.897531549380589
090	090	lin	319.582556950112	-875410.718245332	0.881481805557734

090	095	lin	273.631925289428	-618587.046046378	0.908396458689074
090	100	lin	37.8154167890387	7416.75973309245	0.998303280930727
095	001	lin	111.931597755167	5206.77148474194	0.999319705731589
095	005	lin	29.028752396777	-29078.6232101666	0.968457020590427
095	010	lin	22.3269951880705	-22952.2200801297	0.969269704931833
095	015	lin	19.5088925450435	-13094.0086607594	0.985444501452746
095	020	lin	24.0958719723978	-30777.8315231483	0.948976863875906
095	025	lin	27.4312457033018	-38668.1558722036	0.957328244900603
095	030	lin	23.1566922625611	-20180.8309670185	0.96996369734528
095	035	lin	18.5115647925094	-7096.81386211119	0.987636522974898
095	040	lin	72.4763072930761	-166962.05594426	0.891248949650769
095	045	lin	117.67293183387	-298069.620508	0.89057943960087
095	050	lin	272.248513707721	-782828.761617685	0.860465125211597
095	055	lin	132.950246466811	-327856.056540407	0.906304815339096
095	060	lin	144.141772036535	-325732.976183019	0.949923682359556
095	065	lin	154.33452753513	-362122.562684093	0.898903548052594
095	070	lin	457.009354899086	-1381001.26928996	0.841069703401008
095	075	lin	423.619991137197	-1230493.57604583	0.852315649874535
095	080	lin	310.437216502611	-807840.315569185	0.913172986008287
095	085	lin	156.833906415599	-327378.519180063	0.884450450073134
095	090	lin	408.879912414385	-1124176.32446796	0.902660810831159
095	095	lin	375.216426210746	-1026414.66137522	0.875993810496277
095	100	lin	38.6924941006828	6790.49778612959	0.998165290784243
100	001	lin	113.443943411124	1229.89345849887	0.99932875615284
100	005	lin	26.1550476306756	-18990.0131310926	0.990125820401348
100	010	lin	20.0588162107316	-15961.5875696667	0.985368310422639
100	015	lin	19.2387283816272	-13675.1234204814	0.985448179187138
100	020	lin	19.5950457651137	-16787.2924757963	0.97953750200527
100	025	lin	20.4591552636744	-20102.9736166296	0.962045773156969
100	030	lin	21.8954250823973	-22754.5395549632	0.952908810401462
100	035	lin	22.4523626015822	-24897.6124401666	0.938084639849719
100	040	lin	18.6618963311433	-10075.7337761851	0.987265597491322
100	045	lin	18.9268540398795	-8756.29364461117	0.99015276607936
100	050	lin	19.4867104122046	-15423.1757417962	0.959837560960758
100	055	lin	17.1896024616018	-5028.17813666673	0.987717424003725
100	060	lin	15.8924285003829	-484.611975407344	0.992824053112389
100	065	lin	15.2161443021142	2461.91646701851	0.993546404989386
100	070	lin	14.1797579876184	5487.70693394433	0.994148511768918
100	075	lin	13.6728632519337	6371.29272555537	0.992876838645478
100	080	lin	13.2145193084197	8745.63345742594	0.993670037568143
100	085	lin	12.7547511423232	10692.1036855556	0.994386983716336
100	090	lin	12.464209589347	12734.2355963889	0.994351156909822
100	095	lin	12.1453433914961	15306.9449849815	0.996387124402094
100	100	lin	10.1065969433091	16728.7777150184	0.9954501144683

GLOSSARY

ADSL Asymmetrical Digital Subscriber Line, a technology that uses classical phone lines (copper pair) for high-speed Internet access.

AOL America On-Line, currently one of the largest Internet Service Providers in the USA. Before providing Internet access, AOL was providing proprietary on-line services to its customers.

BBS Bulletin Board System, "a computer system running software that allows users to dial into the system over a phone line and, using a terminal program, perform functions such as downloading software and data, uploading data, reading news, and exchanging messages with other users" [Col04]

BitTorrent A P2P file transfer protocol where a file is broken into small segments that can be downloaded in no particular order from any of the other downloaders of the file. This allows the file to be massively distributed, using much more bandwidth (and higher speed) than if all the downloaders were connecting to one unique source.

DCC Direct Client-to-Client, a way for IRC users to directly connect to another user's IRC client, bypassing the server.

eDonkey A peer-to-peer file sharing application developed by MetaMachine, which allows to download files concurrently from multiple sources.

eMule an open-source, eDonkey-compatible client.

File sharing The "activity of making files available to other users for download over the Internet, but also over smaller networks. Usually file sharing follows the peer-to-peer (P2P) model, where the files are stored on and served by personal computers of the users. Most people who engage in file sharing are also downloading files that other users share. Sometimes these two activities are linked together." [Col04]

Freenet A "decentralized censorship-resistant peer-to-peer distributed data store. Freenet works by pooling the contributed bandwidth and storage space of member computers to allow users to anonymously publish or retrieve various kinds of information. The network routing method Freenet uses is both a key based routing as well as a type of distributed hash table." [Col04]

FTP The File Transfer Protocol. FTP servers were widely used before the advent of peer-to-peer file sharing systems.

Gnutella A decentralized peer-to-peer file sharing system, without a central index. The original software, published in 2000 by two employees of Nullsoft, a division of AOL, was quickly removed, but its protocol has been reverse-engineered and numerous applications nowadays support the Gnutella protocol.

ICQ The first peer-to-peer instant messaging application and one of the most popular ones today.

IP Internet Protocol, a protocol designed for use in interconnected systems of packet-switched computer communication networks.

Instant messaging A computer system that allows instant text communication between two or more people through a network (in particular the Internet).

IRC Internet Relay Chat is an instant messaging application, older than ICQ, but which uses an open protocol and no central server, unlike ICQ.

Jabber An open, XML-based protocol for instant messaging and presence.

JXTA An infrastructure by Sun Microsystems designed as a middleware for developing Java-based peer-to-peer applications.

Kazaa A peer-to-peer file sharing system that implements the FastTrack protocol and uses a distributed index instead of a centralized one, as did Napster.

MLdonkey A free software file-sharing software that implements the eDonkey, Overnet and Gnutella protocols, among others.

MSN Microsoft Network. MSN Messenger is Microsoft's instant messaging application.

Napster The first popular peer-to-peer file sharing system, relying on one central server to index the files owned by the users.

NNTP Network News Transfer Protocol is the protocol used for the distribution, inquiry, retrieval, and posting of news articles in Usenet.

Overnet An offspring of the eDonkey file sharing system, using the same protocol but not relying on servers to locate files.

Peer-to-peer A networking model, opposed to the traditional client-server model, where all the nodes of the network are peers and can act both as client and server at the same time.

Search engine A software that indexes WWW documents and allows users to query them using a full-text search.

SMB System Message Block, a protocol for accessing resources (files, printers, software modules, . . .) designed in 1985 by IBM and then further developed by Microsoft.

TCP Internet's Transport Control Protocol, a reliable, connection oriented transport protocol, running on the top of IP.

UDP Internet's User Datagram Protocol, an unreliable, connectionless transport protocol running on the top of IP.

Usenet An asynchronous communication medium where users post and read text messages to newsgroups. The medium is sustained among a large number of servers which propagate the messages to each other in a peer-to-peer manner.

WWW The World Wide Web, sometimes also abbreviated “Web”

REFERENCES

[aim04] AOL instant messenger. http://www.fact-index.com/a/ao/aol_instant_messenger.html, 2004.

[Ano02a] Anonymous. Gnucleus. http://www.gnucleus.com/, 2002.

[Ano02b] Anonymous. Internet : arrêt du site d'échanges de fichiers Napster. *Le Monde*, Mar 28th, 2002.

[Ano03a] Anonymous. Newzbot! public usenet resources for the masses. http://www.newzbot.com/, 2003.

[Ano03b] Anonymous. Usenet top1000 servers. http://www.top1000.org/, 2003.

[AW03] O. Arturo Ribeiro and M. Weber. P2p applications' architectures. Technical report, University of Jyväskylä, Agora Center, 2003.

[Bar00] S. Barber. Common nntp extensions. RFC 2980, IETF, 2000.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *SIGOPS European Workshop*, September 2002.

[Col04] Collective. Wikipedia, the free encyclopedia. http://www.wikipedia.org/, June 2004.

[DM03] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks — From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.

[Däm03] Hauke Dämpfling. *Gnutella Web Caching System*. http://www.gnucleus.com/gwebcache/newgwc.html, June 2003.

[ea00] D. Box et al. Simple object access protocol (soap) 1.1. Note, WWW Consortium, 2000.

[emu04] eMule-project.net – official emule site. http://www.emule-project.net/, 2004.

[fre] *Freenet*. http://freenetproject.org/cgi-bin/twiki/view/Main/ICSI. Version 0.5.

[FV03] Kevin Fall and Kannan Varadhan. *The ns Manual (formerly ns Notes and Documentation)*. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf, 2003.

[Gon01] Li Gong. Project jxta: A technology overview. http://www.jxta.org/, April 2001.

[Goo03a] Google Inc. Google. http://www.google.com/, 2003.

[Goo03b] Google Inc. Google web apis. http://www.google.com/apis/, 2003.

[HA87] M. Horton and R. Adams. Standard for interchange of usenet messages. RFC 1036, IETF, 1987.

[IEE99] IEEE 802.11 Working Group. IEEE standard for information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. Standard 8802-11:1999(E), ISO/IEC, 1999.

[IEE02] IEEE 802.15 Working Group. IEEE standard for information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 15.1: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans). Standard IEEE Std 802.15.1^{TM}-2002, IEEE Computer Society, 2002.

[Jab05] Jabber Software Foundation, http://www.jabber.org/. *Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP*, 2005.

[KL86] B. Kantor and P. Lapsley. Network news transfer protocol. RFC 977, IETF, 1986.

[Knu98] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.

[KVW+04] Jani Kurhinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen, and Jarkko Vuori. Short range wireless p2p for co-operative learning. In *3rd International Conference on Emerging Technologies and Applications (ICETA 2004)*, Kosice, Slovakia, 2004.

[LRW03] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *Proceedings of The Third Workshop on Internet Applications (WIAPP'03)*. IEEE, 2003.

[man] *UNIX Reference Manual — TALK*.

[mld04] MLdonkey World. http://mldonkey.berlios.de/, 2004.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, Jan 1998.

[msn04] Msn messenger. http://messenger.msn.com/, 2004.

[OR93] J. Oikarinen and D. Reed. Internet relay chat protocol. RFC 1459, IETF, 1993.

[Ora01] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., 1st edition, March 2001.

[ove02] *Overnet*. http://www.overnet.com/documentation/index.html, 2002. Version 0.45.

[PFTV92] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[Rei01] Lisa Rein. O'Reilly Network: ICQ. http://www.oreillynet.com/pub/d/572, 2001.

[RIF02] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, January/February 2002.

[Sa03] S. Shepler and al. Network file system (nfs) version 4 protocol. RFC 3530, IETF, 2003.

[Sal99] Chip Salzenberg. What is usenet. http://www.faqs.org/faqs/usenet/what-is/part1/, 1999.

[sam05] samba. www.samba.org, 2005.

[Sch01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *IEEE International Conference on Peer-to-Peer Computing*, München, Germany, August 2001.

[SGG01] Stefan Saroiu, P. Krishna Gummadi, and Steven D. 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, WA, USA, 2001.

[Sha02] Richard Sharpe. *Just what is SMB?* http://samba.anu.edu.au/cifs/docs/what-is-smb.html, October 2002.

[Sha03] Sharman Networks, http://www.kazaa.com/us/help/glossary.htm. *KaZaA*, 2002-2003. Version 2.1.

[Ste94] W. Richard Stevens. *The Protocols*, volume 1 of *TCP/IP Illustrated*. Addison-Wesley Publishing Company, 1994.

[SW02] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the Second ACM SIGCOMM Internet Measurement Workshop (IMW 02)*. ACM, 2002.

[Wri99] David W. Wright. Guidelines on usenet newsgroup name. http://www.faqs.org/faqs/usenet/creating-newsgroups/naming/part1/, 1999.

[WVV03] Matthieu Weber, Jarkko Vuori, and Mikko Vapa. Advertising peer-to-peer networks over the internet. In *Radiotekhnika*, volume 133, pages 162–170, 2003.

[yah04] Yahoo! messenger. http://messenger.yahoo.com/, 2004.

[ZRM94] Klaus Zeuge, Troy Rollo, and Ben Mesander. *The Client-To-Client Protocol (CTCP)*. http://www.irchelp.org/irchelp/rfc/ctcpspec.html, 1994.

YHTEENVETO (FINNISH SUMMARY)

Internetissä on nykyisin yhä enemmän vertaisverkko-ohjelmia, jotka ovat enimmäkseen tarkoitettu ihmisten väliseen tiedostojen jakamiseen tai viestien vaihtoon. Vaikka Internetin verkkoratkaisu onkin vertaisverkkojärjestelmä, useimmat sovellukset, myös suosituimmat, on suunniteltu niin, että niiden pohjana on palvelintekniikka. Näiden perinteisten sovellusten käyttö vähenee jatkuvasti, kun taas vertaisverkko-ohjelmien käyttö valtaa alaa.

Toisiinsa yhteydessä olevat vertaisverkko-ohjelmat muodostavat päällysverkon, jolla on oma rakenne ja joka on usein riippumaton alla olevan verkon rakenteesta. Koska vertaisverkot ovat tavallaan autonomisia, halutessaan liittyä tällaiseen verkkoon yksittäisen solmun täytyy hankkia *liittymistieto* ainakin yhdestä verkkoon jo kuuluvasta solmusta, joka toimii *sisääntulosolmuna*. Solmusta tulee osa verkkoa, kun se luo yhteyden sisääntulosolmuun. Liittymistiedon hankkimista pidetään yleensä vähäpätöisenä ongelmana ja se jätetään yhden ainoan tietolähteen tehtäväksi. Vain yhden tietolähteen käyttäminen on heikkous verkon rakenteessa, sillä sen poistaminen voi johtaa siihen, että kukaan ei voi liittyä verkkoon.

Tässä tutkimuksessa esitetyn työn tavoite olikin kehittää täysin hajautettu järjestelmä, joka tarjoaa saman palvelun, jota ei voi poistaa, ja joka käyttää olemassa olevaa Internet-infrastruktuuria. IRC ja Usenet valittiin näiksi järjestelmiksi, koska molemmat ovat laajalle levinneitä, tunnettuja ja täysin hajautettuja. Myös Web-hakukoneiden ja verkon sattumanvaraisen selailun käyttöä kuvattiin lyhyesti.

Työssä kuvattiin muodollinen protokolla liittymistietojen julkaisemista varten, jonka avulla käyttäjä pääsee läpinäkyvästi käsiksi tietoon minkä tahansa yläkuvatun median kautta. Työssä määriteltiin myös muodollinen merkintätapa, jotta protokollaa käyttävien solmujen käyttäytymisen kuvaukset olisivat lyhyempiä ja yhdenmukaisia.

Kehitettiin simulaatio-ohjelma, jotta voitaisiin simuloida järjestelmän käyttäytymistä, todistaa sen pätevyys ja tutkia sen vaikutusta infrastruktuuriin, jossa se toimii (IRC tai Usenet) kaistankäytön suhteen. Simulaattori toteutti ja otti huomioon solmujen käyttäytymisen vain toiminnallisella tasolla eikä sellaista yksityiskohtaista tiedonvaihtoa, jota olisi vaadittu todellisessa järjestelmässä. Tämä käyttäytyminen myös kuvattiin tarkasti tutkielmassa. Simulaattori myös oletti, että solmut halusivat löytää mahdollisimman monta muuta solmua, joiden kanssa muodostaa yhteyksiä ja siten saada mahdollisimman monta naapuria, tiettyyn ylärajaan asti.

IRCiin perustuvassa järjestelmässä verkkoon liittymistä yrittävät solmut julkaisivat omat liittymistiedot lähetyskanavalla (tässä tapauksessa IRC- kanavalla, jolla on edeltämääritelty nimi). Tällöin verkkoon jo kuuluvat solmut, jotka halusivat muodostaa lisää yhteyksiä, pystyivät luomaan yhteyden solmuihin, jotka mainostivat itseään.

Usenetiin perustuvassa järjestelmässä verkkoon liittymistä yrittävät solmut lukivat Usenet-artikkeleja tietystä uutisryhmästä ja etsivät liittymistietoja. Jos ne

onnistuivat siinä, niiden oli mahdollista käyttää tietoa ja yrittää luoda yhteys, tai ne julkaisivat omat liittymistiedot siinä toivossa, että jokin toinen solmu yrittäisi luoda yhteyden niihin.

Simulaatiot ajettiin useilla parametreilla, jotta voitaisiin määrittää parhaat mahdolliset arvot parametreille. Nämä parametrit koskivat seuraavia asioita: (1) solmun tarvitsemien naapureiden määrä, jotta se lopettaisi aktiivisen uusien naapurien etsimisen ja odottaisi, että muut löytävät sen (*haluttu naapurien minimimäärä*) ja (2) kuinka monta muuta solmua solmun tarvitsee löytää (mutta ei luoda yhteyttä niiden kanssa), jotta se lähtisi pois lähetyskanavalta (*tuttujen solmujen minimimäärä ennen lähtöä*). Usenet-pohjaisen simulaattorin tapauksessa myös artikkeleiden vanhenemisaika otettiin huomioon (*vanhenemisaika*).

Simulaatiot tuottivat arvoja, jotka kuvasivat enimmäkseen järjestelmän tuottamaa keskimääräistä liikennettä sekä indeksin, joka kuvasi kuinka tehokkaasti solmut olivat klusteroituneet, kun tavoitteena oli estää verkon jakautuminen osiin. Nämä tulokset yhdistettiin sitten järjestelmän tehokkuusindeksiksi. Samat simulaatiot ajettiin kasvavalle solmumäärälle (1000–10 000), jotta voitaisiin yrittää ennustaa järjestelmän skaalautuvuus.

Simulaatioiden tulokset osoittivat, että, kun parametrien arvot valitaan huolella, on mahdollista mainostaa vertaisverkkoja täysin hajautetusti käyttäen siihen jo olemassaolevia Internet-infrastruktuureja kuten IRC-verkkoja tai Usenet-verkkoa. Tässä työssä kuvattu mainostamismenetelmä, jota käytettiin sopivilla parametrien arvoilla, salli solmujen yhdistyä verkoksi, joka muodostui yhdestä ainoasta yhtenäisestä komponentista, mikä oli ensimmäinen edellytys sille, että järjestelmää voitiin pitää “toimivana”.

Molemmat simulaatiot, eli IRCin tai Usenetin käyttäminen lähetyskanavana mainoksille, käyttäytyivät samantapaisesti ja tuottivat optimaalisia tuloksia melkeinpä samoilla parametrien arvoilla:

- Parametrin *haluttu naapurien minimimäärä* pienet arvot tuottivat useita erillisiä komponentteja, kun taas suuremmat arvot takasivat yhden yhtenäisen komponentin. Kuitenkin pienemmät arvot vähensivät solmun ja lähetyskanavan välillä tapahtuvan liikenteen määrää.

- Parametrin *tuttujen solmujen minimimäärä ennen lähtöä* pienet arvot tuottivat vähemmän liikennettä solmua kohti, mutta vaikuttivat hyvin vähän klusteroitumistehokkuuteen (IRC-pohjaisen järjestelmän tapauksessa).

- Ihannearvo sijoittui kohtaan, jossa liikennemäärien minimointi oli ristiriidassa useamman kuin yhden yhtenäisen komponentin syntymisen kanssa. Todellinen ihannearvo piti siksi valita niin, että useiden erillisten komponenttien syntyminen oli epätodennäköistä ilman, että verkkoon syntyi liikaa liikennettä.

Lisäksi Usenet-pohjaisten simulaatioiden tapauksessa mainosten vanhenemisajat täytyi ottaa huomioon liikennemäärän pitämiseksi mahdollisimman pienenä.

IRC-pohjaisessa simulaatiossa parhaat mahdolliset arvot olivat: *haluttu naapurien minimimäärä* $= 35$ ja *tuttujen solmujen minimimäärä ennen lähtöä* $= 5$. Näillä arvoilla liikennettä tuli 6900 tavua solmua kohti. Usenet-pohjaisessa simulaatiossa parhaat mahdolliset arvot olivat *haluttu naapurien minimimäärä* $= 35$, *tuttujen solmujen minimimäärä ennen lähtöä* $= 10$, ja vanhenemisajan parametrin a arvona 0,3 ja parametrin b arvona 5. Näillä arvoilla liikennettä tuli 9400 tavua solmua kohti.

IRC-järjestelmä tuotti vain noin kolme neljäsosaa liikennettä verrattuna Usenet-järjestelmään, mutta toisaalta käytännössä Usenet-palvelimet on mitoitettu korkeampia liikennemääriä varten kuin IRC-palvelimet. Liikenne solmua kohden Usenet-pohjaisissa simulaatioissa riippui kuitenkin verkon solmujen määrästä, kun taas IRC-pohjaisessa järjestelmässä se oli vakio, mikä tarkoittaa sitä, että laajat Usenet-verkot tuottavat paljon enemmän liikennettä kuin pienet.

Parametri *haluttu naapurien minimimäärä* vaikutti voimakkaasti sekä IRC- että Usenet-pohjaisen simulaation klusteroitumistehokkuuteen, kun taas *tuttujen solmujen minimimäärä ennen lähtöä* vaikutti siihen hyvin vähän. Tämä selittyy sillä, että solmu keräsi tietoa muista solmuista enimmäkseen vaihtamalla naapuriluetteloja omien naapureidensa kanssa (jotta lähetyskanava kuormittuisi mahdollisimman vähän). Siksi useimmat tunnetuista solmuista kuuluivat jo ennestään samaan yhtenäiseen komponenttiin kuin solmu itsekin, mikä tarkoittaa sitä, että kahden erillisen komponentin välisiin yhteyksiin ei vaikuttanut lisäävästi parametrin *tuttujen solmujen minimimäärä ennen lähtöä* isompi arvo.

Parametri *haluttu naapurien minimimäärä* vaikutti voimakkaasti IRC-pohjaisten simulaatioiden verkkoliikenteen määrään, ja parametri *tuttujen solmujen minimimäärä ennen lähtöä* vaikutti paljon Usenet-pohjaisten simulaatioiden liikennemääriin. Tämä selittyy IRCin ja Usenetin toimintatapojen erilaisuudella: useiden naapureiden hankkiminen edellytti pidempää pysymistä IRC-kanavalla tai useampien mainosten hakemista Usenet-palvelimelta, mutta uusien tuttujen solmujen hankkiminen silloin, kun parametri *tuttujen solmujen minimimäärä ennen lähtöä* oli pieni, edellytti useita liittymisiä lähetyskanavalle, ja tämä aiheuttaa paljon enemmän otsikkotietoliikennettä Usenetissä kuin IRCissä. Otsikkotietojen aiheuttama liikenne oli runsaampaa kuin lähetyskanavan pitkittyneen käytön aiheuttama liikenne, mikä selittää eroavaisuuden simulaatioiden tuloksissa.

Lopuksi, simulaation aikana luodun verkon luonnehtiminen osoitti, että verkot olivat samankaltaisia, mutta Usenetiä käyttävät simulaatiot synnyttivät verkkoja, jotka olivat tiukemmin yhteydessä kuin IRCiin perustuvissa simulaatioissa: klusteroitumiskerroin oli korkeampi Usenetin tapauksessa, ja niinsanottu verkon "halkaisija" (määritellään pisimmän ja keskimääräisen lyhyimmän polun avulla) oli pienempi. Nämä tulokset osoittivat, että Usenetin käyttö lähetyskanavana tuotti hieman "laadukkaampia" verkkoja, mutta samalla enemmän liikennettä verkossa.

Jatkotutkimusaihe voisi olla WWW-pohjaisen mainostusjärjestelmän tehokkuuden tutkiminen, jollainen tässä työssä kuvattiin mutta ei toteutettu, samoin kuin optimaalisen suunnitelman kehittäminen sattumanvaraiseen IP-osoitteiden

"skannaamiseen" siinä tapauksessa, että mikään edellämainituista lähetyskanavista (IRC, Usenet, Web-palvelin) ei ole käytettävissä. Kuvio voisi perustua esim. IP-osoiteluokkien tilastolliseen jakaumaan eri organisaatiossa, sekä IP-osoitteiden jakaumaan isännille näissä luokissa, jotta halutun vertaisverkon jäsenen löytyminen olisi mahdollisimman todennäköistä.

Lisäksi työssä käytetty mainostusjärjestelmä perustuu oletukselle, että verkon kaikki solmut ovat halukkaita osallistumaan mainostukseen. Kuitenkin todellisuudessa solmut usein toimivat itsekkäästi, eivätkä halua jakaa resurssejaan yhteiseksi hyväksi: tällaisten solmujen vaikutus järjestelmän tehokkuuteen on vielä tutkimatta.