Marcelo Werneck Barbosa
Orientador: Virgílio A. F. Almeida

O uso de localidade de referência para otimizar consultas em arquiteturas par-a-par

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
10 de Julho de 2003

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

O Uso de Localidade de Referência para Otimizar Consultas em Arquiteturas Par-a-Par

MARCELO WERNECK BARBOSA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. VIRGÍLIO AUGUSTO FERNANDES ALMEIDA - Orientador
Departamento de Ciência da Computação - ICEx - UFMG

Profa JUSSARA MARQUES DE ALMEIDA
Departamento de Ciência da Computação - ICEx - UFMG

Prof. NÍVIO ZIVIANI
Departamento de Ciência da Computação - ICEx - UFMG

Prof. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - ICEx - UFMG

Belo Horizonte, 10 de julho de 2003.

Abstract

Services on the Internet are evolving from centralized client-server architectures to fully distributed architectures. Systems based on such architectures are called peer-to-peer systems (P2P), and end-hosts participating in such systems are called peers. A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. The surging increase in the popularity of peer-to-peer applications has led to a dramatic need for a scalable and high performance content location protocol.

Current search algorithms in peer-to-peer systems do not take each node´s interest into account. It is known that users tend to work and relate to each other in groups. A group of users, although not always located in geographical proximity, tends to use the same set of resources (files). We show that groups in peer-to-peer file sharing systems can be identified and search mechanisms in these systems should take the existence and characteristics of these groups into account.

This work describes a distributed algorithm for peers to self-organize into clusters based on interests (peer communities). Each peer maintains a community of peers which share similar interests. Content is located by querying peers in one's community. Peers that have content in common share the same interests. The concept of the algorithm is that there is greater chance to find a file one node is looking for in its own community than in the remainder of the network. Based on this principle, every time a node issues a query for a file, it will first check if it can be found within the community; otherwise, it will be searched in the other hosts. Summarizing, this work aims at implementing an algorithm that exploits locality in interests to efficiently allow content location and retrieval.

Resumo

Os serviços na Internet estão evoluindo de arquiteturas cliente-servidor distribuídas para arquiteturas completamente distribuídas. Sistemas baseados em tais arquiteturas são chamados par-a-par (*peer-to-peer*, P2P) e computadores que participam destes sistemas são chamados de nodos (*peers*). Um problema fundamental que as aplicações par-a-par enfrentam é localizar de forma eficiente o nodo que armazena um certo item (arquivo). O grande aumento na popularidade de aplicações par-a-par demanda a criação de protocolos de localização de conteúdo escaláveis de alto desempenho.

Algoritmos de busca atuais de sistemas par-a-par não levam o interesse de cada nodo em conta. É sabido que usuários tendem a trabalhar e se relacionar em grupos. Um grupo de usuários, apesar de nem sempre estar localizado geograficamente próximo, tende a usar o mesmo conjunto de recursos (arquivos). Mostramos que grupos em sistemas de compartilhamento de arquivos par-a-par podem ser identificados e mecanismos de busca destes sistemas devem considerar a existência destes grupos bem como suas características.

Este trabalho descreve um algoritmo distribuído para nodos se organizarem em grupos baseado em seus interesses (comunidades de nodos). Cada nodo mantém uma comunidade de nodos que compartilham interesse comum. Conteúdo é localizado pesquisando os nodos de uma comunidade. Nodos que possuem conteúdo em comum compartilham os mesmos interesses. O algoritmo supõe que há uma probabilidade maior de encontrar um arquivo que um nodo está pesquisando em sua própria comunidade do que no restante da rede. Baseado neste princípio, sempre que um nodo inicia uma pesquisa por um arquivo, ele primeiro checará se o mesmo pode ser encontrado na comunidade; se não, ele será buscado nos demais nodos. Concluindo, este trabalho tem como objetivo implementar um algoritmo que explora a localidade de interesses para permitir a localização e a busca de conteúdo de forma eficiente.

Agradecimentos

Acredito que um trabalho de pesquisa deve contribuir não só para o progresso da ciência mas também para a formação de cada indivíduo. Desta forma, espero que este trabalho possa contribuir para a continuação dos estudos na área e para a inspiração de outros pesquisadores. De qualquer forma, posso garantir que foi uma grande lição pessoal.

Aprendi também que um trabalho de pesquisa não se faz sozinho e que são muitas as pessoas que colaboram para sua execução. São muitos aos quais agradecer e espero não me esquecer de ninguém.

Primeiramente, gostaria de agradecer a Deus por ter me dado uma família e amigos maravilhosos. Sem este suporte, nada disso seria possível. Meus pais são meu grande exemplo; sempre guiaram meu caminho e me apoiaram em todas as minhas decisões. A meus irmãos também por todas as preocupações.

Ao Professor Virgílio, meu orientador, cuja paciência e tranquilidade foram fundamentais neste processo.

A Melissa que sempre foi um exemplo pessoal e profissional para mim. Ela compartilhou todas as aflições do mestrado e teve paciência para discutir todas as minhas dúvidas e questionamentos. Espero um dia poder retribuir toda a ajuda dispendida.

Aos professores Robson, Clarindo e Wilson que me deram a oportunidade de trabalhar no Synergia. O Synergia possui um papel muito importante na formação de todos os alunos que trabalham aqui e, com certeza, contribuiu enormemente para a minha formação também. Algumas pessoas com as quais trabalhei aqui serão meu exemplo e fonte de inspiração por toda minha vida profissional: Márcia, Fernanda e Raphael. Além disso, o Synergia possui um ambiente agradável de trabalho e foi possível fazer aqui muitos amigos, entre os quais gostaria de agradecer especialmente a Valdo, Daniela e Benício.

Em especial a minhas amigas de infância Fabiana e Carol e a todos meus amigos, que puderam compreender minhas atividades e minha indisponibilidade principalmente nos últimos meses.

A todos do laboratório e-speed, em especial aos que sempre se mostraram extremamente dispostos a ajudar como Diego, Flávia, Barra, Marisa e Tassni.

Finalmente, aos professores que me incentivaram a ingressar no mestrado; Mariza, Rodolfo e Frederico.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Services on the Internet are evolving from centralized client-server architectures to fully distributed architectures. Systems based on such services architectures are called peer-to-peer systems (P2P), and end-hosts participating in such systems, peers [SMZ01], meaning that participants interact as equals. The participants in a typical P2P system might include computers at homes, schools, and businesses, and can grow to several million concurrent participants [BKK+03].

Peer-to-peer is much more than a system for sharing pirated music [Let03]. These systems work as follows: peers publish content to the system and download content from the system. Downloading content involves locating peers that have copies of the content, selecting a peer, and retrieving a copy from that peer [SMZ01].

Peers play a variety of roles. When accessing information, they are clients. When serving information to other clients, they are servers. When forwarding information for others, they are routers [Kub03]. Unlike traditional distributed systems, P2P networks aim to aggregate large numbers of computers that join and leave the network frequently [RF02].

Ensuring the availability of content on the Internet is expensive. Publishers who want high availability have few options. They can use hosting services, build and manage their own content distribution infrastructures, or contract Content Delivery Networks. All of these options are prohibitively expensive for an average Internet user who wants to share a hundred megabytes of digital photographs with his friends. A low cost solution is to publish the content from one's own desktop into a peer-to-peer content distribution system [SMZ03].

The popularity of P2P file sharing applications has created a flurry of recent research activity into P2P architectures [SGG01]. These networks have become, in a short period of time, one of the fastest growing and most popular Internet applications [CFK03]. These systems are rapidly growing in importance as the medium of choice for storage and exchange of information [CSWW00, PRU01, Rip01]. Sharing such large volumes of data is made possible by distributing the main costs - disk space for storing the files and bandwidth for transferring them - across the peers in the network [YGM01]. The peer-to-peer architecture

has been used to support many different applications such as compressed music file sharing, disk-space sharing, file sharing, or even computing cycle sharing. This trend has become possible in part because of the fact that personal computers have become increasingly faster, they have much more disk space than in the past, and are connected to the Internet at higher speed than before [MA01].

Distributed file service is a typical P2P application. The idea is straightforward: replace the local hard disk of a computer with pools of storage spread throughout the Internet; the computer interacts with a vast web of peers to read or write information [Kub03]. These systems allow users worldwide access and provision of information while enjoying a level of privacy not possible in the present client-server architecture of the web [AH00].

Summarizing, P2P systems are attractive for several reasons:

- The barriers to starting and growing such systems are low, since they usually don't require any special administrative or financial arrangements, unlike centralized facilities;
- P2P systems offer a way to aggregate and make use of the tremendous computation and storage resources on computers across the Internet;
- The decentralized and distributed nature of P2P systems gives them the potential to be robust to faults or intentional attacks, making them ideal for long-term storage as well as for lengthy computations.

The P2P overlay network can be viewed as an undirected graph, where the vertices correspond to nodes in the network, and the edges correspond to open connections maintained between the nodes. Two nodes maintaining an open connection between them are known as neighbors. Messages may be transferred in either direction along the edges. For a message to travel from node A to node B, it must travel along a path in the graph. The length of this traveled path is known as the number of hops taken by the message. Similarly, two nodes are said to be "n hops apart" if the shortest path between them has length n [YGM01].

The first well-known P2P systems were Napster and Gnutella. The Gnutella protocol is an open, decentralized group membership and search protocol, mainly used for file sharing [Rip01]. Napster is used to share music files. Both of them have similar goals: to facilitate the location and exchange of files between a large group of independent users connected through the Internet. In both systems, the files are stored on the computers of the individual users (peers), and they are exchanged through a direct connection between the downloading and uploading peers. All peers in the system are symmetric, in that they all have the ability to function both as a client and a server. This symmetry is one attribute that distinguishes P2P systems from many conventional distributed system architectures. Though the process of exchanging files in both systems is similar, Napster and Gnutella differ substantially in how peers locate files [SGG01]. More details on each architecture can be found in chapter 2.

From the time Napster and Gnutella were launched, peer-to-peer systems have been quite improved. More recent P2P systems have been incorporating new functionalities. The

architecture of peer-to-peer systems such as Kazaa [Thed] and Morpheus [Thee] incorporate improved design that allows for more efficient downloads (simultaneous from several peers and ability to resume after failure); and improved search (by designating some peers as search-hubs super nodes that cache the index of others [CFK03]).

Peer-to-peer networking applications such as "file sharing communities" have seen explosive growth. Applications like Kazaa boast of over two hundred million downloads of their free client, with over two and a half million downloads occurring each week [Thed, CF02]. Recent work has noted that the amount of network traffic generated by this increasingly popular class of application now rivals that of the world wide web [CF02, Mar02]. As the percentage of Internet users utilizing these applications and the volume of traffic they generate has exploded, the demands on such systems have intensified. Users of these services demand the high performance of a single dedicated server serving them as quickly as possible, i.e., not limited by the requests of other users. Thus, in order to be viable, such systems must be fast, reliable, network friendly and well distributed [CF02].

The main challenge in P2P computing is to design and implement a robust and scalable distributed system composed of inexpensive, individually unreliable computers in unrelated administrative domains [BKK+03]. The next section describes one of the most important open problems in peer-to-peer networks: the efficient search for files.

1.1 Motivation

The key to the usability of a data-sharing P2P system and one of the most challenging design aspects, is the employment of efficient techniques for search and retrieval of data. The best search technique for a given system depends on the needs of the application. For storage of archival systems focusing on availability, search techniques such as Chord [DBK+01], Pastry[RD01], Tapestry[ZKJ00] and CAN[RFH+01] are well suited, because they guarantee location of content if it exists, within a bounded number of hops. To achieve these properties, these techniques tightly control the data placement and topology within the network, and currently only support search by object identifiers. In systems where persistence and availability are neither guaranteed nor necessary, such as Gnutella, Freenet, Napster and Morpheus, search techniques can afford to have looser guarantees. Current search techniques in "loose" P2P systems tend to be very inefficient, generating too much load on the system [YGM01].

Locating files in systems with hundreds of thousands of users distributed geographically is challenging because of scale and dynamism: the number of nodes, logical files, requests, and concurrent users may all be large. The system has multiple sources of variation over time: files are created and removed frequently; nodes join and leave the system without a predictable pattern. In such a system with a large number of components, even a low variation rate at the individual level may aggregate into frequent group level changes [IRF02].

P2P computing raises many interesting research problems in distributed systems. One

of the main challenges in P2P research is the lookup problem. How do you find any given data item in a large P2P system in a scalable manner, without any centralized servers or hierarchy? This problem is at the heart of any P2P system. It is not well addressed by most popular systems currently in use, and it provides a good example of how the challenges of designing P2P systems can be addressed [BKK+03]. The lookup problem is a major issue in the P2P area mainly because the effectiveness of a peer-to-peer network is largely a function of the versatileness and scalability of its search mechanism [CFK03].

Current search algorithms in peer-to-peer systems do not take each node's interest into account. Users usually relate to each other in groups as they tend to use the same set of resources. We propose a distributed algorithm for peers to self-organize into clusters based on interests. Previous research in different areas has proved that the identification of these interest relations is possible. We show that P2P systems can also benefit from the identification of such relations and structures.

1.1.1 Communities

Whenever autonomous individuals act in a shared environment, interaction emerges which may result in manifold relations between the individuals and/or groups of individuals. Those relations and the associated behavior of individuals and groups may induce structures to the groups. Such structures are commonly called *communities*. The behavior of biological individuals, such as ants or bees, but also humans', has been widely studied in the social sciences. Key findings include that despite the largely varying capacities of individuals and groups, a set of common characteristics for acting in a shared environment still may be observed. However, this usually depends on the specific knowledge of individuals and their time already spent within a community. Among the characteristics identified, we find that:

- each individual can identify a few members of a community and may exchange information with them;
- there is no single individual who knows or controls the whole community;
- some individuals may be more "intelligent" than others and have more and/or better information;
- communities are often hierarchically structured with one or more outstanding individuals.

Starting with just a few individuals, communities continuously evolve. The resulting set of inter-related community members is generally called the *social network* of a community. While the number of individuals in a community can grow very fast; the single individual needs only little information about other individuals to still be able to potentially interact with a large number or all of the community members. The *six degrees of separation*

property [Mil67] illustrates this in the case of human communities. Moreover, communities are often characterized by a highly self-organizing behavior.

Computer networks or distributed systems in general may be regarded as communities similar to the above examples. Most obviously, the Internet or Web forms entities that can be characterized as communities. Many approaches to define communities on the Web [GKR98, FLG00] are based on the use of existing link patterns and they therefore lack the characteristic community properties to adapt to the current context and to dynamically evolve. Implicit information other than link patterns is necessary to achieve this [VBKJ02].

1.2 Goals

Few papers [SMZ03, SMZ01] have analyzed the importance of taking each node's interest into account. We developed and analyzed a new algorithm that is based on interests. We define that two nodes have interests in common if they store some identical files. We show a node stands greater chance of finding a file in another node if they share interests. Our algorithm differs from the one defined in [SMZ03] in some aspects which are described in detail in chapter 4. We also show it is possible to identify communities of nodes with the same interests. The properties of other kinds of networks such as the networks of people, scientists or web pages have been studied for quite long and have shown interesting characteristics.

Summarizing, this work aims at implementing algorithms that exploit locality in interests to efficiently allow content location and retrieval. We intend to identify communities of hosts in a peer-to-peer system to do so. We believe that properties of these communities can bring benefits to the search mechanisms of different P2P architectures.

1.3 Contributions

Simple algorithms are sufficient to capture interest-based relationships. In light of the goals mentioned in the previous section, the key contributions of the thesis are:

- the implementation of a Gnutella Simulator with heterogeneous characteristics. This simulator can be used by other researchers to investigate possible improvements in the Gnutella architecture;
- a new algorithm based on interest that improves performance of the search mechanisms of P2P systems;
- performance analysis of this algorithm for Freenet and Gnutella.

1.4 Thesis Organization

This thesis is organized as follows: Chapter 2 describes the peer-to-peer architectures used in our experiments. We focus on the way each architecture implements its search mechanism. Chapter 3 describes the work related to this research. It presents different lookup mechanisms proposed by other researchers. It also details one search mechanism that is related to this work. It consists of an implementation of lists of interests in Gnutella in order to retrieve files faster and decrease the load in the network. As algorithm of identification of communities of web pages in the Internet is also presented. This algorithm is the basis of the algorithm of identification of communities of interests. Chapter 4 describes the algorithm of identification of communities of interests we have proposed. Chapter 5 presents our simulation environment and results and chapter 6 our conclusions and future work.

Chapter 2

Peer-to-Peer Architectures

The purpose of a data-sharing P2P system is to accept queries from users, locate and return data (or pointers to the data) to the users. Each node owns a collection of files or records to be shared with other nodes. The shared data usually consists of files, but it is not restricted to them. For example, the collection could be records stored in a relational database. Queries may take any form that is appropriate given the type of data shared. If the system is a file-sharing system, queries may be file-identifiers or keywords with regular expressions, for instance. Nodes process queries and produce results (such as pointers to files) individually, and the total result set for a query is the union of results from every node that processes the query [YGM01].

In this chapter, we present some well-known P2P architectures, emphasizing the way their location protocols are implemented. First, we describe Napster, which was the architecture that many Internet users tried music file sharing for the first time. Then, we describe the Gnutella architecture, maybe the most studied and used in research projects. Afterwards, the Freenet architecture is detailed. Finally, the Gnutella and Freenet architectures are compared.

2.1 Napster

Peer-to-peer networks came to fame with the advent of Napster [Thef], a centralized architecture where the shared items of all peers are indexed in a single location. Queries were sent to the Napster Web site and the results were returned after locally searching the central index; subsequent downloads were performed directly from peers [CFK03].

In Napster, a large cluster of dedicated central servers maintains an index of the files that are currently being shared by active peers [SGG01, ALPH01]. Each peer maintains a connection to one of the central servers through which the file location queries are sent. The servers process the query and return a list of matching files and locations to the user. Then, the peer may choose to initiate a file exchange directly from another peer. The central servers are also responsible for keeping data such as the peers' reported connection

bandwidth and how long the peer has remained connected to the system. This information is useful for the distinction of possible download sites [SGG01]. This approach has inherent scalability and resilience problems: the database is a central point of failure [BKK+03]. Although Napster uses a peer-to-peer communication model for the actual file transfer, the process of locating a file is still very much centralized [RFH+01].

At its peak, Naspter boasted a registered user base of 70 million and 1.57 million simultaneous users [Lee03]. The legal issues which led to Napster's demise exposed all centralized architectures to a similar fate. Internet users and the research community subsequently turned to decentralized P2P architectures, where the search index and query processing, as well as the downloads, are distributed among peers [CFK03]. Next, we describe two decentralized P2P architectures, Gnutella and Freenet.

2.2 Gnutella

Gnutella is a file sharing application and protocol for distributed search. Nodes participating in a Gnutella network are called servents (*serv*ers and cli*ents*). They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set and respond with applicable results. Due to its distributed nature, a network of servents that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline [Thec].

The open architecture, achieved scale, and self-organizing structure of the Gnutella network make it an interesting P2P architecture to study. The topology of this overlay network and the routing mechanisms used have a significant influence on application properties such as performance, reliability, and scalability [RF02].

End hosts join Gnutella by connecting to existing end hosts already in the Gnutella, forming an application-level overlay on top of the physical network. Once attached to the network, nodes send messages to interact with each other [Rip01, AH00]. All end hosts, or peers, can serve files and retrieve files from one another. To facilitate the file sharing, messages are sent between end hosts. Queries for files are broadcast on the overlay network, and replies are routed back to the host which originally generated the query through the overlay network. Ping and pong (a reply to a ping) messages are used to discover hosts on the network [Sri01].

Once attached to the network, peers interact with each other by means of messages. Peers will create and initiate a broadcast of messages as well as rebroadcast others (receiving and transmitting to neighbors) [AH00]. The messages allowed in the network are:

- Ping Messages - Messages directed to a host. A servent uses ping messages to actively probe the network for other servents. From time to time, each node may send ping messages to some or all of its neighbors in order to check if they are still connected to the network. The specification of the Gnutella Protocol makes no recommendations

as to the frequency at which a servent should send ping messages, although servent implementers should make every attempt to minimize ping traffic on the network.

- Pong Messages - A reply to a ping. The pong message contains information about the peer such as their IP address and port as well as the number of files shared and the total size of those files. Peers forward this kind of message to their neighbors so that it is possible to later find other peers. This is needed in case peers disconnect themselves from the network.

- Query Messages - These are messages that query for specific files and can be forwarded throughout the entire network (at least theoretically). Query messages are uniquely identified, but their source is unknown. Messages are flagged with a time-to-live (TTL) field. At each hop, the TTL is decremented. As soon as a peer sees a message with a TTL of zero, the message is dropped, that is, it is not rebroadcast [AH00].

- Query Response Messages - These are replies to query messages, and they include the information necessary to download the file (IP, port, and other location information). Responses also contain a unique client ID associated with the replying peer. These messages are propagated backwards along the path that the query originally took. Since these messages are not broadcast it becomes impossible to trace all query responses in the system. A servent should only reply to a query with a query hit if it contains data that strictly meets the query search criteria.

- Get/Push Messages - Get messages are simply a request for a file returned by a query. The requesting peer connects to the serving peer directly and requests the file. Certain hosts, usually located behind a firewall are unable to directly respond to requests for files. For this reason, the Gnutella protocol includes push messages. Push messages request the serving client to initiate the connection to the requesting peer and upload the file. However, if both peers are located behind a firewall, a connection between the two will be impossible.

Figure 2.1 show how ping and query messages are forwarded in a Gnutella network. On the left side of figure 2.1, the node receives a ping message. Then, it responds this message with a pong message and it forwards the ping message to all its known neighbors. On the right side of figure 2.1, the nodes receive query messages. The query is passed on to the neighbors and if the node has the file which is being searched, it replies to that node with a query hit message.

One of the drawbacks of the search mechanism of the Gnutella architecture is the fact that it is based on flooding of queries. In order to locate a file, a peer sends a query packet to all of its neighbors. Upon receiving a query packet, a peer checks if any locally stored files match the query. If so, the peer sends a query response packet back towards to the query originator. Whether or not a file match is found, the peer continues to flood the query over the overlay [SGG01]. This broadcast method will find the target file very quickly, given that it is located within a radius of TTL(time-to-live). However, broadcasting is extremely

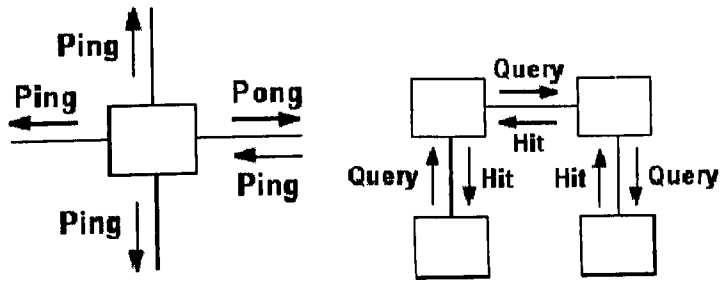

Figure 2.1: Left: Ping Routing. Right: Query Routing

costly in terms of bandwidth. Such a search strategy does not scale well [ALPH01]. Figure 2.2 shows how Napster and Gnutella locate files.

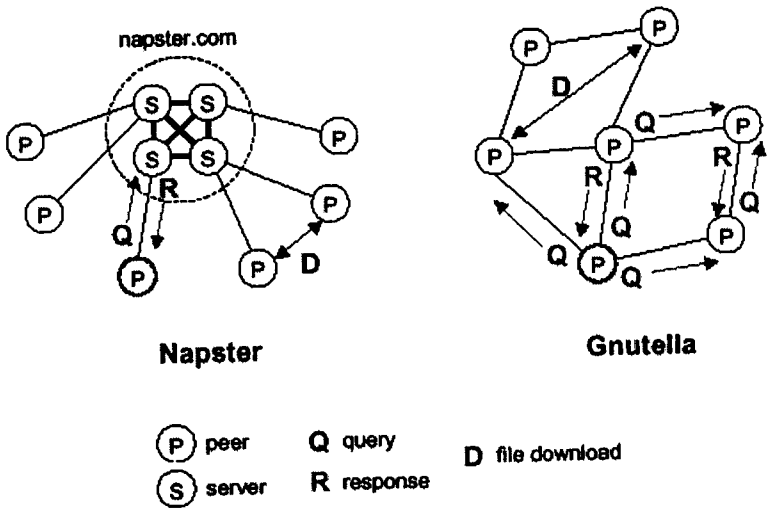

Figure 2.2: File location in Napster and Gnutella

Query broadcasts are not scalable. As more nodes join, more queries are sent. Currently, a peer connected to the network through a modem does not have enough bandwidth to support queries and replies, let alone retrieve content [Sri01]. The "broadcast" approach doesn't scale well because of the bandwidth consumed by broadcast messages and the compute cycles consumed by the many nodes that must handle these messages. In fact, the day after Napster was shut down, reports indicate that the Gnutella network collapsed under the load created by a large number of users who migrated to it for sharing music [BKK+03]. Available data shows the network growing from around 1.000 nodes in November 2000 to over 40.000 in June 2001. However, since the summer of 2001, the network has been steadily shrinking, reaching an average of 16.000 users in January 2002. If the main interest in Gnutella was sharing of music, many users have switched to more efficient specialized services such as Kazaa. In spite of that, as the first widespread decentralized and open protocol, Gnutella is worth studying. Its simple basic protocol also makes it

easy to use in experiments [VBKJ02]. In order to scale Gnutella to more than hundreds of users, it is imperative to stop the flooding of every query and reply [Sri01]. There has been a great amount of research on discovering new search mechanisms to replace the flooding in Gnutella network. The search algorithm based on communities of interests we proposed is also designed to decrease the number of messages nodes in this kind of network send to each other.

2.3 Freenet

Current systems afford little privacy to their users, and typically store any given data item in only one or a few fixed places, creating a central point of failure. Freenet [Theb] operates as a location-independent distributed file system across many individual computers that allows files to be inserted, stored and requested anonymously [CSWW00]. The five main goals of Freenet are:

- Anonymity for both producers and consumers of information
- Deniability for storers of information
- Resistance to attempts by third parties to deny access to information
- Efficient dynamic storage and routing information
- Decentralization of all network functions

The system is also designed to adapt to usage patterns, automatically moving, replicating, and deleting files to make the most effective use of available storage in response to demand [CHMaW].

Freenet is implemented as a peer-to-peer network of nodes that query one another to store and retrieve data files, which are named by location-independent keys. Each node maintains its own local datastore, which it makes available to the network for reading and writing, as well as a dynamic routing table containing addresses of other nodes and the keys that they are thought to hold. No node is privileged over any other, so no hierarchy or central point of failure exists. Joining the network is simply a matter of discovering the address of one of more existing nodes [CSWW00].

To provide anonymity, Freenet avoids associating a document with any predictable server, or forming a predictable topology among servers. As a result, unpopular documents may simply disappear from the system, since no server has the responsibility for maintaining replicas. Furthermore, a search may often need to visit a large fraction of nodes in the system, and no guarantees are possible [BKK+03].

The basic model is that queries are passed along from node to node in a chain of proxy requests, with each node making a local routing decision in the style of IP routing about where to send the query next (this varies from query to query). Nodes know only their

immediate upstream and downstream neighbors in the chain. Each query is given a hops-to-live count, which is decremented at each node to prevent infinite chains. Each query is also assigned a pseudo-unique random identifier, so that nodes can prevent loops by rejecting queries they have seen before. When this happens, the immediately preceding node simply chooses a different node to forward to. This process continues until the query is either satisfied or exceeds its hops-to-live limit. Then, the success or failure result is passed back up the chain to the sending node [CSWW00].

In order to retrieve a file, a user must first obtain or calculate its binary file key. She then sends a request message to her own node specifying that key and a hops-to-live value. When a node receives a request, it first checks its own store for the data and returns it if found, together with a note saying it was the source of the data. If not found, it looks up the nearest key in its routing table (the one with the closest identifier) to the key requested and forwards the request to the corresponding node. If that request is ultimately successful and returns with the data, the node will pass the data back to the upstream requestor, cache the file in its own datastore, and create a new entry in its routing table associating the actual data source with the requested key. A subsequent request for the same key will be immediately satisfied from the local cache; a request for a "similar" key (determined by lexicographic distance) will be forwarded to the previously successful data source. If a node cannot forward a request to its preferred downstream node because the target is down or a loop would be created, the node having the second nearest key will be tried, then the third nearest, and so on. If a node runs out of candidates to try, it reports failure back to its upstream neighbor, which will then try its second choice, and so on [CSWW00, CHMaW, Lan01]. Figure 2.3 depicts a typical sequence of request messages.

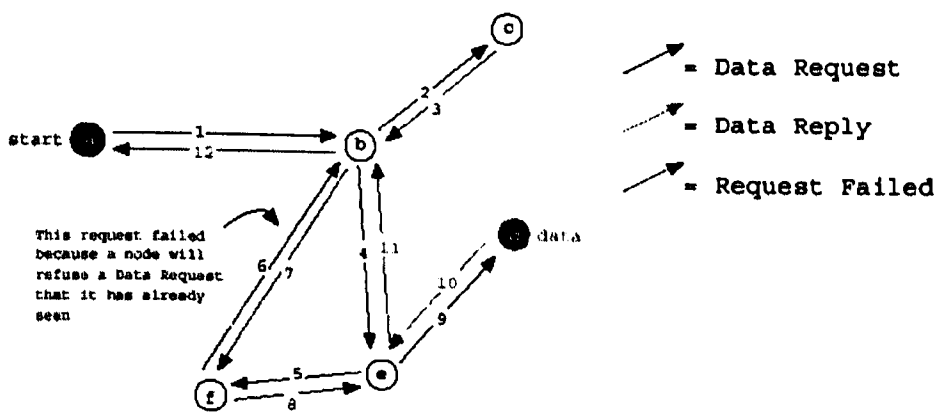

Figure 2.3: A typical request sequence

The user initiates a request at node *a*. Node *a* forwards the request to node *b*, which forwards it to node *c*. Node *c* is unable to contact any other node and returns a backtracking "request failed" message to *b*. Node *b* that tries its second choice, *e*, which forwards the request to *f*. Node *f* forwards the request to *b*, which detects the loop and returns a backtracking failure message. Node *f* is unable to contact any other node and backtracks one step further back to *e*. Node *e* forwards the request to its second choice, *d*, which has

the data. The data is returned from d via e and b back to a, which sends it back to the user. The data is also cached on e, b and a [CSWW00].

The quality of the routing should improve over time for two reasons. First, nodes should become specialized in locating sets of similar keys. Second, nodes should become similarly specialized in storing clusters of files having similar keys. Because forwarding a successful request will result in the node itself gaining a "copy" of the requested file, and most requests will be for similar keys, the node will mostly acquire files with similar keys. Taken together, these two effects should improve the efficiency of future request in a self-reinforcing cycle, as nodes build up routing tables and datastores focusing on particular sets of keys, which will be precisely those keys that they are asked about. In addition, the request mechanism will cause popular data to be transparently replicated by the system and mirrored closer to requestors [CSWW00].

Finally, as nodes process requests, they create new routing table entries for previously unknown nodes that supply files, increasing connectivity. This helps new nodes to discover more of the network [CSWW00].

2.4 Gnutella and Freenet Compared

Summarizing, these are the techniques currently used by Gnutella and Freenet systems:

- Gnutella - Uses a breadth-first traversal (BFS) over the network with depth limit D, where D is the system-wide maximum time-to-live of a message, measured in hops. A source node S sends the query message with a TTL of D to a depth of 1, to all its neighbors. Each node at depth 1 processes the query, sends any results to S, and then forwards the query to all of their neighbors to a depth of 2. This process continues until depth D is reached (i.e., when the message reaches its maximum TTL), at which point the message is dropped.

- Freenet - Uses a depth-first traversal (DFS) with depth limit D. Each node forwards the query to a single neighbor and waits for definite response from the neighbor before forwarding the query to another neighbor (if the query was not satisfied), or forwarding results back to the query source (if the query was satisfied).

If the quality of results in a system were measured solely by the number of results and not by satisfaction, then the BFS technique is ideal because it sends the query to every possible node (i.e., all nodes within D hops), as quickly as possible. However, if satisfaction were the parameter of choice, BFS wastes much bandwidth and processing power because most queries can be satisfied from the responses of relatively few nodes. With DFS, since each node processes the query sequentially, searches can be terminated (i.e., the query message dropped) as soon as the query is satisfied, thereby minimizing cost [YGM01]. Besides that, the node that holds the requested file is not necessarily the best node to download the file from because the selection of such node is based solely on key

proximity; important characteristics such as available bandwidth or availability are not taken into account.

Chapter 3

Related Work

The rapid and widespread deployment of peer-to-peer systems suggests that there are important advantages to such systems. P2P designs harness huge amounts of resources - the content advertised through Napster has been observed to exceed 7 TB of storage on a single day, without requiring centralized planning or huge investments in hardware, bandwidth or disk space. As such, P2P file sharing may lead to new content distribution models for applications such as software distribution, file sharing, and static web content delivery.

Central to any P2P system is the indexing scheme used to map file names (whether well-known or discovered through some external mechanism) to their location in the system. That is, the P2P file transfer process is inherently scalable, but their hard part is finding the peer from whom to retrieve the file. Thus, a scalable P2P system requires, at the very least, a scalable indexing mechanism.

There are two classes of solutions proposed for decentralized peer-to-peer content location. Unstructured content location, used by Gnutella, relies on flooding queries to all peers. Peers organize themselves into an overlay. In order to find content, a peer sends a query to all its neighbors on the overlay. In turn, the neighbors forward the query on to all of their neighbors until the query has traveled a certain radius. While this solution is simple and robust even when peers join and leave the system, it does not scale.

Another class of protocols based on the Distributed Hash Table (DHT) abstraction have been proposed to address scalability. In these protocols, peers organize into a well-defined structure that is used for routing queries [SMZ03]. Data items are inserted in a DHT and found by specifying a unique key for that data. To implement a DHT, the underlying algorithm must be able to determine which node is responsible for storing the data associated with any given key. To solve this problem, each node maintains information (the IP address) of a small number of other nodes (its neighbors) in the system, forming an overlay network and routing messages in the overlay to store and retrieve keys [BKK+03]. To implement DHTs, lookup algorithms have to address the following issues:

- Mapping keys to nodes in a load-balanced way. In general, all keys and nodes are

identified using an m-bit number or identifier(ID). Each key is stored at one or more nodes whose IDs are "close" to the key in the ID space;

- Forwarding a lookup for a key to an appropriate node. Any node that receives a query for a key identifier s must be able to forward it to a node whose ID is "closer" to s. This rule will guarantee that the query eventually arrives at the closest node;
- Building routing tables adaptively. To forward lookup messages, each node must know about some other nodes. This information is maintained in routing tables, which must adapt correctly to asynchronous and concurrent node joins and failures.

Although DHTs are elegant and scalable, their performance under the dynamic conditions common for peer-to-peer systems is unknown [SMZ03].

With DHTs, peers are required to store or index certain data items, not necessarily those items that these peers have contributed to or are interested in. Additionally, some hashing algorithm is used to identify the peers storing a given data item. The connections between different peers are also a function of the architecture. Thus, while DHTs can be very effective for applications where queries involve unique item identifiers, they require that peers store data for the "common good"; they incur much larger overhead than unstructured architectures when peers fail or leave the network; and inherently, they cannot efficiently support partial-match queries [CFK03].

The next section describes two important systems based on distributed hash tables.

3.1 Distributed-Hash Systems

3.1.1 CAN - Content Addressable Network

A hash table is a data structure that efficiently maps keys onto values and serves as a core building block in the implementation of software systems. It is expected that many large scale-distributed systems could likewise benefit from hash tables functionality. The term Content-Addressable Network (CAN) [RFH+01] describes such a distributed, Internet-scale, hash table. Perhaps the best example of current Internet systems that could potentially benefit from a CAN are the recently introduced P2P file sharing systems, which require at the very least, a scalable indexing mechanism.

CANs resemble a hash table. The basic operations performed on a CAN are the insertion, lookup and deletion of (key, value) pairs. In its design, the CAN is composed of many individual nodes. Each CAN node stores a chunk (called a zone) of the entire hash table. In addition, a node holds information about a small number of "adjacent" zones in table. Requests (insert, lookup, or delete) for a particular key are routed by intermediate CAN nodes towards the CAN node whose zone contains that key. The CAN design is completely distributed (it requires no form of centralized, control, coordination or configuration), scalable (nodes maintain only a small amount of control state that is independent of the number of nodes in the system), and fault-tolerant (nodes can route around failures).

The design centers around a virtual d-dimensional cartesian coordinate space. This virtual coordinate space is used to store (key, value) pairs as follows: to store a pair (k_1, v_1), key k_1 is deterministically mapped onto a point p in the coordinate space using a uniform hash function. The corresponding (key, value) pair is then stored at the node that owns the zone within which the point p lies. Efficient routing is therefore a critical aspect of a CAN.

Intuitively, routing in a Content Addressable Network works by following the straight line path through the Cartesian space from source to destination coordinates. A CAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zone of each of its immediate neighbors in the coordinate space. In a d-dimensional coordinate space, two nodes are neighbors if their coordinate spaces overlap. This purely local neighbor state is sufficient to route between two arbitrary points in the space. Using its neighbor coordinate set, a node routes a message towards its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates.

Certain additional problems remain to be addressed in realizing a comprehensive CAN system. An important open problem is that of designing a secure CAN that is resistant to denial of service attacks. This particularly a hard problem because (unlike the Web) a malicious node can act, not only as a malicious client, but also as a malicious server or router. A number of ongoing projects both in industry and research are looking into the problem of building large-scale distributed systems that are both secure and resistant to denial-of-service attacks. In the current design, if one or more nodes become unreachable, a takeover algorithm is executed. It ensures that one of the failed node´s neighbors takes over the zone. However, in this case, the (key, value) pairs held by the departing node are lost until the state is refreshed by the holders of the data.

Additional related problems that are topics for future work include the extension of the CAN algorithms to handle mutable content, and the design of search techniques such as keyword searching built around the CAN indexing mechanism.

The CAN structure was only evaluated so far through simulation. The authors of the project say they are embarking on an implementation project to build a file sharing application that uses a CAN for distributed indexing.

3.1.2 Chord - a Distributed Lookup Service

Chord is a distributed lookup protocol that addresses the problem of efficiently locating the node that stores a particular data item. It provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. Each Chord node needs routing information about only a few other nodes. Because the routing table is distributed, a node resolves the hash function by communicating with just a few other nodes. The Chord protocol specifies how to find the locations of keys, how new nodes join the system, and how to recover from the failure (or planned departure) of existing nodes [SMK+01] .

If we compare the Chord protocol with protocols used by other systems, we can highlight some differences. The Freenet storage system, like Chord, is decentralized and symmetric and automatically adapts when hosts leave and join. Freenet does not assign responsibility for documents to specific servers; instead, its lookups take the form of searches for cached copies. This allows Freenet to provide a degree of anonymity, but prevents it from guaranteeing retrieval of existing documents or from providing low bounds on retrieval costs. Chord does not provide anonymity, but its lookup operation runs in predictable time and always results in success or definitive failure [SMK+01]. Chord also eliminates a single point of failure present in Napster because of its decentralized nature [DBK+01]. Chord can also be compared with other DHTs mechanisms. CAN [RFH+01] uses a d-dimensional Cartesian coordinate space to implement a distributed hash table data structure. CAN operations are easy to implement, but an additional maintenance protocol is required to periodically remap the identifier space onto nodes. On the other hand, the state maintained by a CAN node does not depend on the network size.

Chord can be used as a lookup service to implement a variety of systems. In particular, it can help avoid single points of failure or control that systems like Napster possess, and the lack of scalability that systems like Gnutella display because of their widespread use of broadcasts [SMK+01].

3.2 Content Location using Interest-Based Locality

The authors of [SMZ03] developed a distributed algorithm for peers to self-organize into clusters based on interests. Their design philosophy departs from existing work in that they seek to retain the simple, robust, and fully decentralized nature of Gnutella, while improving scalability, its major weakness.

There are many challenges in providing peer-to-peer content distribution systems. In this paper, the authors address one fundamental challenge: what is the appropriate strategy for locating content given that it may be continuously replicated at many locations in the peer-to-peer system? If it cannot be located efficiently, there is little hope for using peer-to-peer technology for content distribution [SMZ03].

The algorithm has the following principle: if a peer has a particular piece of content that one is interested in, then it is likely that it will have other pieces of content that one is also interested in. These peers exhibit what is called interest-based locality. They proposed a self-organizing protocol, interest-based shortcuts, that efficiently exploits interest-based locality for content location. Peers that share similar interests create shortcuts to one another. Peers then use these shortcuts to locate content. When shortcuts fail, peers resort to using the underlying Gnutella overlay. Shortcuts provide a loose structure on top of Gnutella's unstructured overlay [SMZ03].

When a peer joins the system, it may not have any information about other peers' interests. Its first attempt to locate content is executed through flooding. The lookup returns a set of peers that store the content. These peers are potential candidates to be added to a "shortcut list". In this implementation, when a node has no shortcuts, one

peer is selected at random from the set (returned by the lookup) and added. Subsequent queries for content go through the shortcut list. If a peer cannot find content through the list, it issues a lookup through Gnutella, and repeats the process for adding new shortcuts. There are several design alternatives for shortcut discovery. New shortcuts may be discovered through exchanging shortcut lists between peers, or through establishing more sophisticated link structures for each content category similar to structures used by search engines [SMZ03].

The benefits of such an implementation are two-fold. First, shortcuts are modular in that they can work with any underlying content location scheme. Second, shortcuts only serve as performance-enhancement hints. If a document cannot be located via shortcuts, it can always be located via the underlying overlay. Therefore, having a shortcut layer does not affect the correctness and scalability of the underlying overlay. In general, shortcuts are a powerful primitive that can be used to improve overlay performance. For example, shortcuts based on network latency can reduce hop-by-hop delays in overlay networks. This article explored the use of a specific kind of shortcut based on interests, for content location.

The experiments to evaluate the performance of this algorithm were executed in a Gnutella network (through simulation). Figure 3.2(a) illustrates how content is located in Gnutella. A query initiated by the peer at the bottom is flooded to all peers in the system. Figure 3.2(b) depicts a Gnutella overlay with 3 shortcut links for the bottom-most peer. To avoid flooding, content is located first through shortcuts. A query is flooded to the entire system only when none of the shortcuts have the content. For scalability, each peer allocates a fixed-size amount of storage to implement shortcuts. Shortcuts are added and removed from the list based on their perceived utility.

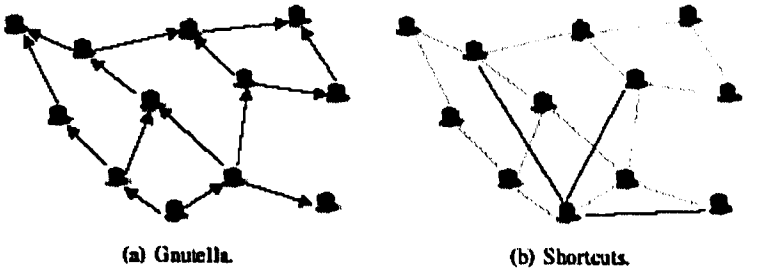

Figure 3.1: Content location paths.

Given that there may be many shortcuts on the list, which one should be used? In this design, shortcuts are ranked based on their perceived utility. If shortcuts are useful, they are ranked at the top of the list. A peer locates content by sequentially asking all of the shortcuts on its list, starting from the top, until content is found. Rankings can be based on many indices, such as probability of providing content, latency of the path, amount of content at the shortcut, and load at the shortcut. A combination of indices can be used based on each peer's preference [SMZ03].

In this work, the performance of Gnutella was compared with and without shortcuts

using five traces collected at different locations and different times. Some traces are from proxy servers while others were collected from real peer-to-peer systems. The shortcuts significantly improved the performance of content location for Gnutella by providing large decreases in the amount of load in the system and the time to locate the content. For instance, the load and scope were reduced by at least by a factor of 3.

The goal of our work is to propose a new algorithm whose main goal is to identify communities of nodes that share interests. These communities can be identified in an analogous way as communities of web pages. The following section describes an algorithm that identifies such web communities.

3.3 Identification of Web Communities

The existence of an increasing percentage of human knowledge and society in hyperlinked form on the web has advantages beyond the commonly stated improvements to information access. The potential for analysis of interests and relationships within science and society are great. However, analysis of content on the web is difficult due to the decentralized and unorganized nature of the web. Information on the web is authored and made available by millions of different individuals, operating independently, and having a variety of backgrounds, knowledge, goals and cultures. Despite its decentralized, unorganized, and heterogeneous nature, the web self-organizes such that the link structure allows efficient identification of communities. Identification of communities on the web is significant for several reasons. Practical applications include focused search engines, content filtering, and complementing text-based searches. More importantly, global community identification allows for analysis of the entire web and the objective study of relationships within and between communities [FLGC02].

The web can be modeled as a graph where vertices are web pages and hyperlinks are edges. The authors of [FLGC02, FLG00] define a community on the web as a set of sites that have more links (in either direction) to members of the community than to non-members. Community membership is a function of both a web page's outbound hyperlinks as well as all other hyperlinks on the web; therefore, these communities are "natural" in the sense that they are collectively organized by independently authored pages. The web self-organizes such that these link-based communities identify highly related pages.

The definition of a web community is NP-complete because it maps into a family of graph partitioning problems [FLGC02]. However, if one assumes the existence of one or more seed web sites and exploits systematic regularities of the web graph, the problem can be recast into a framework that allows for efficient community identification by using a polynomial time algorithm that should scale well to studying the entire web graph. In this way, identifying a web community becomes identical to solving the s-t maximum flow network problem, which has many efficient polynomial time solutions. Members of such a community can be efficiently identified in a maximum flow/minimum cut framework, where the source is composed of known members and the sink consists of well-known non-members. Figure 3.2 shows an example of a web community.

In order to obtain any web community, a crawler was implemented [FLGC02, FLG00]. Figure 3.3 illustrates how the focused crawler retrieves pages and the graph that is induced by the crawl. The crawl begins with the seed web pages, shown as the set (b) in the figure, and finds all pages that link to or from the seed set. Outbound links are trivially found by examining the HTML of the page. Inbound links are found by querying a search engine such as Google.

Once the URLs from set (c) are identified, their HTML are downloaded and all outbound links are recorded. Some of these outbound links may refer to pages already crawled; however, many of the outbound links from (c) will refer to pages now downloaded (from set (d)). These pages corresponding to set (d) are efficiently treated as composite sink vertex, as each is linked to a virtual sink vertex.

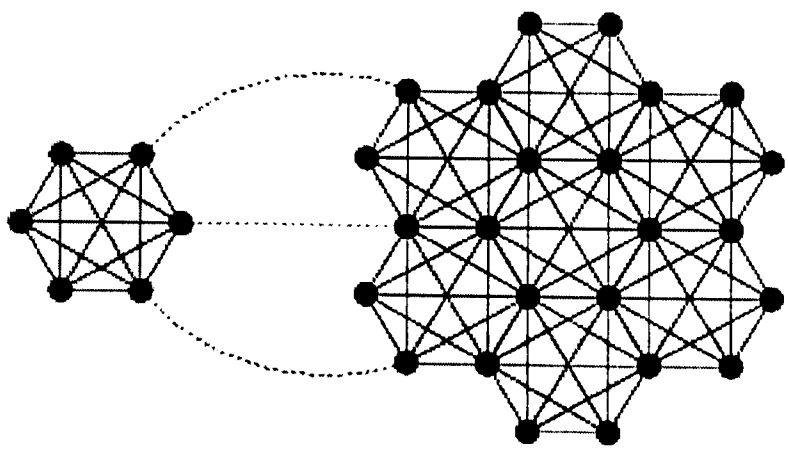

Figure 3.2: Maximum flow methods will separate the two subgraphs with any choice of the source and sink nodes that has the source node on the left and sink node on the right subgraph

After this graph is formed, the maximum flow problem is solved for this graph. The nodes that are in the solution of the problem are the ones that form the community. Figure 3.4 presents the algorithm that identifies the communities of web pages. The procedure EXACT-FLOW-COMMUNITY augments the web graph in three steps: an artificial source, s, is added with infinite capacity edges routed to all seed vertices in S; each pre-existing edge is made bidirectional and rescaled to a constant value k; and all vertices except the source, sink, and seed vertices are routed to the artificial sink with unit capacity. After augmenting the web graph, a residual flow graph is produced by a maximum flow procedure. All vertices accessible from s through non-zero positive edges form the desired result and satisfy our definition of a community. According to this procedure, the graph has depth two, regardless of the artificial nodes. The algorithm imposes no limit on the number of steps the crawler performs; however, the authors noticed that nodes beyond the second level are not usually as related to the seed node as the nodes of the first and second layers. This algorithm can be improved by finding more seeds. These seeds

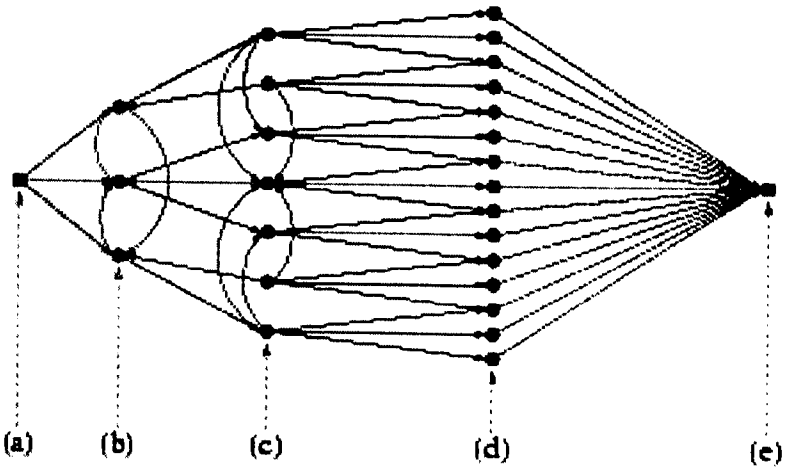

Figure 3.3: Focused community crawling and the graph induced: (a) the virtual source vertex; (b) vertices of the seed nodes; (c) vertices of nodes one link away from any seed site; (d) references to nodes not in (b) or (c); and (e) the virtual sink vertex

are found following the same procedure - the nodes that are part of the solution of the algorithm, that is, the ones most related to the original seed nodes, are added to the set of seed nodes. Therefore, results are improved on each iteration by reseeding the algorithm with additional web sites found in earlier steps.

A test of the algorithm starting at the home pages of biologist Francis Crick, astrophysicist Stephen Hawking, and computer scientist Ronald Rivest yielded groups of sites that are tightly focused on each researcher's life, work and field. The sites are remarkably topically related. [Whi02].

In this chapter we described two different algorithms that are the basis for the algorithm of identification of communities of interests we have developed. The first one describes one possible way of reducing the load in Gnutella networks by identifying nodes that share interests in commom. The second algorithm defines a technique to identify communities of web pages, i.e., a group of subject-related sites. The concept of our algorithm is that it is possible to identify groups of nodes that share interests in a similar way as groups of closely related web pages. These topics are further explored in the next chapter.

procedure EXACT-FLOW-COMMUNITY
  input: **graph**: $G = (V, E)$; **set** : $S \subset V$; **integer** : k .
  Create artificial vertices, s and t and add to V.
  for all $v \in S$ **do**
    Add (s, v) to E with $c(s, v) \equiv \infty$.
  end for
  for all $(u, v) \in E$ **do**
    Set $c(u, v) \equiv k$.
    if $(v, u) \notin E$ **then** add (v, u) to E with $c(v, u) \equiv k$.
  end for
  for all $v \in V$, $v \notin S \cup \{s, t\}$ **do**
    Add (v, t) to E with $c(v, t) \equiv 1$.
  end for
  call : MAX-FLOW (G, s, t).
  output: **all** $v \in V$ still connected to s.
end procedure

Figure 3.4: Algorithm to identify web communities

Chapter 4

Interest-Based Algorithms

Users tend to work and relate to each other in groups. A group of users, although not always located in geographical proximity, tends to use the same set of resources (files). For example, members of a science group access newly produced data to perform analyses or simulations. This work may result in new data that will be of interest to all scientists in the group. Another example, users the same age tend to access and download music of similar styles, usually of the top singers.

File location mechanisms such as those proposed in CAN [RFH+01], Chord [DBK+01] or Tapestry [ZKJ00] do not attempt to exploit this behavior: each member of the group will hence pay the cost of locating a file of common interest [IRF02]. We claim that groups in peer-to-peer file sharing systems can be identified and search mechanisms in these systems should take the existence and characteristics of these groups into account.

We present below one algorithm whose main goal is to group nodes whose interests are similar in order to make file location more efficient.

4.1 Identification of Communities of Interests

We describe here a distributed algorithm for peers to self-organize into clusters based on interests (peer communities). This algorithm is similar to the one proposed in [SMZ03] in the sense they have similar goals and the same motivation. In this section, the algorithm of identification of communities of interests is described in detail and the differences between this algorithm and the one described in [SMZ03] are highlighted.

The algorithm is based on the concept that each peer maintains a community of peers which share similar interests. Peers in the community can be ranked based on different characteristics such as current interests or dynamic performance. Peers that have content in common share the same interests. Figure 4.1 illustrates this relationship. The peer in the middle is looking for content A, B and C, which can all be found at the peer at the far left, the one with more interests in common.

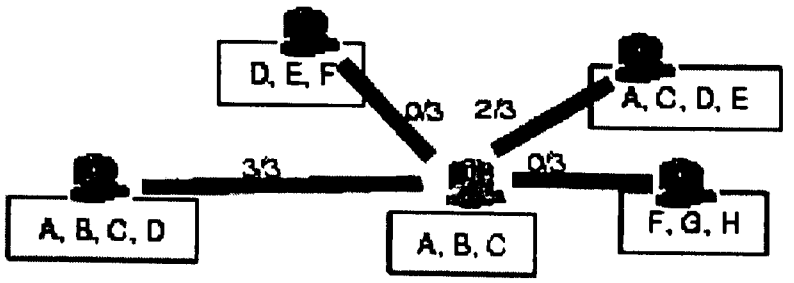

Figure 4.1: Locality in interests relationship

The concept of the algorithm is that there is greater chance to find a file one node is looking for in its own community than in the remainder of the network. Based on this principle, every time a node issues a query for a file, it will first check if it can be found in the community; otherwise, it will be searched in the other hosts following the original Gnutella searching process.

The algorithm based on communities of interests can be applied to any network. In this work, we consider the Gnutella and Freenet networks. Since they have different query processing mechanisms, we ought to distinguish how the algorithm works for both networks.

In Freenet, the structure that guides the searching process is the routing table, hence, queries are sent to nodes chosen according to the information stored in this structure. Suppose node A issues a query looking for file F. According to node A's routing table, the node that is likely to have the file is node C; however, according to its community of interests, node B is the one most likely to store it. In this case, the query is sent to node B because it is in node A's community. So, when a node is searching for a file, the community of interests is examined. If there is a node in the community with considerable interest, the request is made to this node, otherwise, the normal request of the Freenet network is made (using the routing tables). The definition of considerable interest can be configured, that is, it is possible to define that two nodes share interests if they share at least n files. We believe the greatest benefit of the application of this algorithm in a Freenet network is the increase in the number of successful requests, that is, queries for files that were found in the network. Since the searching process in Freenet networks is breadth-first and queries are limited by a time-to-live, few nodes process this query. Although the search is guided by file identifier proximity, there is a chance that the requested file will not be found even though it might be stored by some node in the network.

The algorithm of identification of communities of interests is also very benefitial to Gnutella networks. As it was mentioned before, the search process in a Gnutella network is made through flooding of queries, that is, every query is broadcast to all neighbors of a certain node. The algorithm of communities of interests can be used in a Gnutella network in two different ways. First, queries can be sent to every node in the community regardless of their common interest or other information. Second, the nodes in the communities can be ranked based on some criteria. Then, queries are sent to only one node at a time in the community. If the first node doesn't have the file, the query is sent to the second one and

so on until the query is fulfilled. If the nodes in the community cannot answer it, it is sent to all neighbors of the node (following the usual search process). Different criteria might be used to rank nodes in one's community. Ranking can be done according to common files the nodes share in common or through other data such as bandwidth or amount of available content. If nodes in one's community are not ranked, the queries are sent to every node in the community. This is yet a better approach than the original flooding mechanism, because fewer nodes are queried and the queries sent to the community are not broadcast to the rest of the network.

Therefore, the main benefit of both approaches in the Gnutella network is to decrease the load in each node, i.e., each node has to process fewer queries, since they are not broadcast to every neighbor. Another benefit of the use of the algorithm of communities of interests is to decrease the average time taken to find a file since it is more likely to find it in the community. As it has been explained before, in a Gnutella network, when a node issues a query, it waits for hit messages during a certain time. Each node can define this waiting time. So, the time the download takes to start (latency time) is at least this waiting time, since the download doesn't start before the waiting time expires. If the algorithm of communities of nodes is used and a file cannot be found in the nodes of one's community, this node will have to resort to the common flooding mechanism. The latency time in this case will be the time needed to find out that no node in the community stored the file plus the normal waiting time. Therefore, it can be concluded that if nodes in one's community don't have the file, the latency time will be greater compared to the original flooding mechanism. Thus, it is desirable that most queries be resolved in the community, so the latency time will be considerably less compared to the common search process.

4.1.1 Performance Evaluation

In order to evaluate how successful the implementation of the algorithm of identification of communities of nodes is compared to the original routing mechanism of the Freenet and Gnutella networks, we had to specify the criteria used to evaluate it. Since we are dealing with different architectures, the criteria used to evaluate the use of the algorithm of communities of interests in each one might be a little bit different. The algorithms were analyzed according to common criteria such as:

- Success rate: how often are queries resolved through the communities of interests? If they are usually resolved through them, then performance may be improved. We expect that the percentage of successful queries will increase.

- Average Request Path Length: Path length is the minimum number of hops a query travels before it reaches a peer that has the content. Average Request Path length is defined as the average number of hops that are necessary to resolve a query. For Freenet, it is the number of hops a request traverses until the first copy of content is found. For example, if a peer finds content after asking two nodes, (i.e., the first one was unsuccessful), the path length for the lookup is two hops. In a Gnutella network,

when a node sends a query to all its neighbors, it will receive a number of messages of the query hit type from all nodes that possess the file it is looking for. In this case, we are interested in the minimum reply path length, that is, the minimum number of hops required to find that file. Clearly, the minimum reply path length is desired to be the shortest possible.

Besides these, the performance of the algorithm in Gnutella networks was also analyzed according to:

- Load: the load is defined as the number of queries each node has to process per unit of time. In our experiments, we compute the load for all kinds of messages. Our intention is to diminish the load of query messages in every node. Load is collected only for Gnutella because of the flooding mechanism.

- Scope: the scope is defined as the number of nodes that process a query. It is expected that the scope will be reduced with the use of the algorithm of identification of communities of interests, since queries will be answered in fewer hops. Scope is collected only for Gnutella because of the flooding mechanism.

- Latency time: it is defined as the time the node waits for the download of a file to start. In Gnutella, when a node issues a query, it needs to wait for positive replies during a certain time. When this time is finished, this node chooses one of the nodes that sent hit messages to get the file from. This choice can be made according to different criteria. By using the algorithm of identification of communities of interests, the average latency time is expected to be reduced because files found in the community can start to be downloaded sooner.

The criteria just mentioned before are used to evaluate the effectiveness of the algorithm of communities of interests. However, other data might be collected in order to get a more complete view of the properties of the network. These data are not associated with the evaluation of the algorithm but are quite useful to grasp other characteristics of the peer-to-peer networks. The data that can be collected are:

- Average content: this is defined as the average number of files per node. It helps understand how nodes use their storage capacity. It can also be detected if there are many file replacements, that is, if datastores are filled up very often. This data is collected considering distinct classes of nodes defined based on their storage capacity.

- Number of neighbors: this is defined as the average number of neighbors per node. It helps understanding how connected nodes are. The way nodes connect greatly influence the load in the network. If nodes are tightly clustered, the average load in the network tends to be higher (for Gnutella networks). It also influences the outcome of queries. If nodes are poorly connected, it will be harder to find files.

- Average number of replicas of files: this is useful to identify if there are popular files in the network. Moreover, if files are spread through many different nodes, queries for these files will be successfully answered more often.

- Download time: this is defined as the time taken to get a file from another node. The download time of a file depends on the size of the file and the bandwidth of the two nodes involved in the process. It is important to emphasize that the algorithm of identification of communities of interests is not aimed at reducing the download time.

- Query popularity: this is defined as the average number of times there is a request for the same file.

- Community size: it is defined as the number of nodes that are part of a community. The higher the size of one's community, the higher the number of nodes that share interests with it.

These indices provide a more complete understanding of the characteristics and the performance of the whole network as well as an accurate idea of the benefits of the use of the algorithm of identification of communities of interests.

4.1.2 Description of the algorithm

This section describes the algorithm of identification of communities of interests in detail. This algorithm was developed with the goal of making file location in P2P architectures more efficient. It is based on the idea that peers are part of a community in which they share interests. In addition, peers tend to search files that reflect this interest, for example, people the same age tend to download music files of the same genre. We believe there is higher probability that peers in the community will provide the requested file.

The concept of a community of nodes here is analogous to the one described in [FLG00], according to which communities of web pages are identified regarding the inbound and outbound links. The algorithm of identification of communities of interests defines that these communities of nodes are formed according to the common files they share. Hence, the main goal of the algorithm is to find nodes that store files in common.

Although our algorithm and the one defined in [SMZ03] have similar goals, they differ in some aspects which are described as follows

- In our algorithm, nodes that share interests are identified by probing other nodes in the network while the one described in [SMZ03] are identified while queries are sent. When queries are broadcast, only nodes that store that file will be added to the node's community. In our algorithm, more files are searched during the process of identification of communities, thus more nodes are likely to be found and added to one's community.

- The algorithm described in [SMZ03] heavily depends on nodes sending queries in order to identify nodes with similar interests. In our algorithm, nodes in the network are probed from time to time in order to find nodes that share interests. Thus, nodes that seldom send queries are also able to identify a group of nodes with similar interests.

The community of a node can be viewed as a graph where the vertices are nodes and the edges are connections between nodes. The connections represent common interests, that is, the existence of a connection between two nodes means that they store files in common. Each edge is assigned a weight; it represents the number of files the two nodes store in common.

Figure 4.2 shows a community of nodes. Each community is composed of a seed node. The seed node is the basis of the community. It is the node whose community will be identified. In figure 4.2, the node is represented by the letter *a*. The first step of the algorithm is to find out nodes in the network that store as many files as possible in common with the seed node. These nodes are represented by the group labeled *b* in figure 4.2. We say that these nodes are at depth one of the graph. For each node at depth one, the process of finding nodes with similar interests is repeated and the group of nodes *c* is found. These nodes are said to be at depth two of the graph. The next step is to create an artificial sink in the graph and then to connect each node at depth two of the graph to this artificial sink. The artificial sink is represented by the letter *d* in figure 4.2.

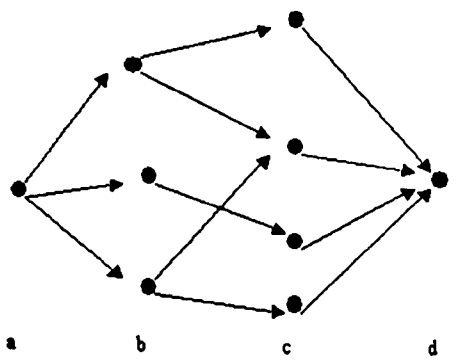

Figure 4.2: A community of nodes. (a) - the seed node. (b) - nodes at depth one. (c) - nodes at depth two. (d) - the artificial sink

At this moment, the graph has an origin, which is the seed of the community and a sink, which was created artificially. The next step of the algorithm is to solve the maximum flow problem on this graph. The solution to the problem will consist of the nodes which are members of the community of the seed node. Since edges in the graph are assigned weights and these weights correspond to the amount of files shared, nodes that share more interests (files) stand greater chances to be part of this community.

Summarizing, the goal of the algorithm is to identify a community of nodes for each node of the network. Therefore, the community depends on the seed node and each node will have its own community. This is necessary so that seed nodes can search for files in their own communities, that is, in the nodes they are more likely to find these files. The purpose of the algorithm is to find nodes that store files in common with the seed node. After these nodes are found, the algorithm is applied one more time, that is, nodes with the same interests as the nodes at depth one are found. Like this, it is possible to find the nodes that share interests with the ones the seed node shares.

The community of interests of a seed node evolves through time. From time to time, each node probes the network with the purpose of finding more nodes that share interests.

Next, we provide a summarized description of the algorithm.

```
1 Randomly choose a subset of the files the node stores.
2 Randomly choose nodes to be queried.
  (any nodes in the network)
3 For each file chosen, send a query to each node in the group
  of the nodes chosen.
4 If the nodes have at least one file in common,
  add the node to the initial community.
5 The weight of a node in the community
  is determined by the number of files stored in common.
6 Repeat the steps for the nodes added to the community.
  A graph will be formed with maximum depth two.
7 Create an artificial sink.
  Every node of depth two is connected to the sink vertex.
8 Calculate the maximum flow of this graph.
  The seed node is the origin.
  The sink is the artificial node.
9 The nodes in the solution of the maximum flow problem
  are members of the community of the seed node.
```

As it can be seen, communities of nodes are identified in a similar fashion as communities of web pages. Though the identification process is very similar, some differences can be highlighted:

- All the edges of the graph from which communities of web pages are identified have the same capacity. This is due to the fact that the edge represents a connection (link) between two pages. In this sense, all links are equivalent because they represent the existence of one link between two pages. In the algorithm of identification of communities of nodes, edges can have different capacities because they can represent different degrees of interests. Nodes that share more interest (files) in common have greater capacity edges connecting them.

- We have implemented a basic algorithm of identification of communities of nodes so far. The identification of communities of web pages can be improved by reseeding the

algorithm with the latest results. We have not implemented such an improvement yet, though we believe it will present good results.

When a node issues a query in a Gnutella network, it will start querying the nodes that are its neighbors; these nodes will do the same and so on. In this way, the query is always restricted at depth one to the same nodes, that is, the neighbors. On the other hand, one node's community might be composed of different nodes in the network. In this way, the community of nodes is not restricted to the neighbors and consequently enhances the search process.

The communities of interests are quite useful to the search process of peer-to-peer systems as the results in the next chapter will show.

Chapter 5

Experimental Results

Algorithms are usually evaluated through simulation. In order to evaluate the effectiveness of the search algorithm for peer-to-peer systems we have proposed, we used simulators for two different P2P architectures: Freenet and Gnutella. The Freenet experiments were undertaken with an open source simulator. The algorithm of identification of communities of interests was implemented on it. To perform experiments in a Gnutella network, we have built a simulator of our own. The goal of this chapter is to describe the environments where the simulations were run and the results we have accomplished.

The first section of this chapter describes the main characteristics of the Freenet Simulator. The second section presents the structure of the Gnutella Simulator. The following sections present the results obtained with the use of both simulators.

5.1 The Aurora Freenet Simulator

The simulations in the Freenet network were made with a Freenet simulator called Aurora [Thea]. Aurora is an open-source simulator written in C++. It has quite poor documentation and it is difficult to completely grasp how its main functionalities are implemented. It simulates a very homogeneous network, in the sense that every node has the same attributes as the others, for example, every node has the same storage capacity. Fundamental characteristics of hosts are not taken into account, such as bandwidth. The participation of hosts is static; they are online all over the simulation. Besides that, all files have the same attributes; they have the same size and are identified by a unique identifier. The simulator uses uniform distributions to generate the identifiers of nodes that will perform operations on the network as well as the identifiers of the files that will be requested.

Aurora has a very peculiar way of collecting statistical data on the network. Aurora defines that the network may be in one of two possible states at a time: frozen and not frozen. When the network is not frozen, the operations (insertions and requests) are executed normally, but statistical data on these operations are not part of the overall statistics of the experiments. From time to time, the network is frozen and a large number

of operations(insertions and requests) are performed so that data on these operations are collected. The network is frozen with the sole purpose of collecting statistical data, that is, they are not executed actually. For example, when a request is made and the network is frozen, the file is not downloaded even if it is found. However, the number of hops taken to find this file and other data are cumulated with other statistics.

One of the outstanding characteristics of Aurora is that it explores three different scenarios, which are described below:

- Train Scenario: The network is composed of 1.000 nodes and has the initial form of a ring, trained over 5.000 time steps. It shows evolution of path length and clustering in a fixed network.
- Attack Scenario: The network is initially a ring composed of 20 nodes. It is grown to 1.000 nodes, and attacked in order of highest degree first until 250 nodes remain. It shows network behavior under targeted attack.
- Growth Scenario: The network is initially a ring composed of 20 nodes. It is grown to 200.000 nodes. It shows network scalability.

The possibility to work with these specific scenarios is one of the benefits of using Aurora.

In Aurora, the network is composed of several nodes. The main structure of each host is the datastore. Each datastore consists of a list of documents and a list of references. Documents represent files the node stores and references can be viewed as "pointers" to other nodes. These "pointers" are a fundamental part of the request mechanism of the network. As explained before, nodes in a Freenet network tend to store files with closer identifiers. These pointers guide the search towards the files with closer identifiers to the one that is being requested.

The main structure of the simulator is shown in the following code extract.

```
for (time=0; time < SIMULATION_TIME; time++)
{
  node = rand(network_size);
  key = rand(key_size);
  event = rand(1);
  if (event == 0)
  {
    request(node, key);
  }
  else if (event == 1)
  {
    insert(node, key);
```

```
    }
}
```

As it can be seen from this pseudo-code, there are two possible events: requests and insertions. Both of them occur with the same probability. In addition, it is possible to see that keys (file identifiers) and nodes are obtained through an uniform distribution. It was not shown in the extract above, but the simulator attempts to query for a file that is present in the network, that is, that at least is stored by one node. This is done so that it is possible to collect data on the search process because the files that are not stored by any node in the network will not spoil the results. Like this, it is possible to know if a file is not found, it means the search mechanism has failed.

5.2 The Gnutella Simulator

This section describes our implementation of a Gnutella Simulator. The main goal of building a simulator was to evaluate the performance of the algorithm of identification of communities of interests. Besides that, we also wanted to investigate the impact of several other features in a Gnutella network, features that were not exploited by other researchers such as nodes having different storage capacities and bandwidth. In this way, the simulator attempts to represent a more heterogeneous network than other simulators do (e.g. the Aurora Simulator). Existing P2P systems treat the majority of their components as equivalent. This purist philosophy is useful from an academic standpoint, since it simplifies algorithmic analysis. In reality, however, some peers have distinct characteristics:

- Computers have different CPUs, memory, storage capacity, network connectivity, and so forth;
- Some computers are professionally managed and highly available while others are not;
- Physically, some computers reside at network hubs, while others are at the edges. Some are locked in machine rooms, while others are public.

Treating all peers as equal forces us to cater to the lowest common denominator. By exploiting differences, we can better tune performance, availability, reliability, security, and attack vulnerability [Kub03].

The Gnutella simulator was written in Java. It reproduces several characteristics of a Gnutella network. The network is composed of several nodes which send messages to each other in order to remain connected to the network and to query for files. In our experiments, the network is composed of a fixed number of nodes, that is, no nodes are inserted or removed from it. Hence, in this work, we do not explore network growth nor its resilience to attacks.

Proper evaluation of a peer-to-peer system must take into account the characteristics of the peers that choose to participate. Few of the peer-to-peer architectures being developed are evaluated with respect to such considerations. This is probably in part due to the lack of information about the characteristics of hosts that participate in these popular systems [SGG01]. Few papers have performed a detailed measurement study of peer-to-peer systems. In order to implement the simulator in a way it would reproduce the characteristics of a real Gnutella network, we modeled its characteristics based on [SGG01, NRS+02]. Although these papers have greatly contributed with analyses of real data obtained from Gnutella users, they have not presented the statistical distributions that these data follow. This makes the construction of Gnutella simulators a hard task. Because of this, we had to use approximate distributions. Since the main goal of this work is to analyze the performance of the algorithm of communities of interests, the use of these approximate distributions don't affect the final result.

The following sections present more detailed information on the design of the simulator.

5.2.1 General Characteristics

Simulations are run through a certain number of time steps. In each time step, one or more possible events can occur. The events that might occur are:

- Request - A node issues a query for a file. It means that this node is going to send a query message to all its neighbors and this message will be propagated to every neighbor of nodes that receive it until a time-to-live is reached.
- Insertion. A node inserts a file in its list of files. It means that a new copy of that file is inserted in the whole network and from this moment on, it is available to be downloaded by other nodes. There is a higher probability of inserting a file in the first time steps than at the remainder of the simulation. This is done so that more files can be distributed among the nodes in the beginning of the simulation.
- Ping. A node sends a ping message to all its neighbors in order to check if they are still online and to find out more neighbors. This message will be propagated to every neighbor of nodes that receive it. Nodes will send pong messages in response.

Each of these events can be triggered by just one node at each time step, that is, at each time step there will be at most one request, one insertion and/or the sending of one ping message. Besides that, the node that performs one operation is not allowed to perform another one at the same time step. Each event might occur or not according to one specific probability. The probabilities associated with each event are shown in table 5.1.

These probabilities were defined based on intuition. These numbers were considered reasonable since there are probably more requests than other operations. Besides that, the Gnutella Specification [Thec] makes no recommendation as to how often pings are sent throughout the network. It only specifies that they should not be sent very often in order

Event	Probability
Request	50%
Initial insert	25%
Insert	5%
Ping	10%

Table 5.1: Probabilities of events

not to increase the load in the network. Insertions were considered to be less frequent operations.

The main structure of the simulator is shown in the following code extract.

```
while (currentTime <= SIMULATION_TIME)
{
  if (random() < REQUEST_PROBABILITY)
  {
    node = getNodeID(PARETO, getNumberOfNodes());
    file = getFileID(PARETO, getNumberOfFiles());
    request(node);
  }
  if (random() < INSERT_PROBABILITY)
  {
    node = getNodeID(PARETO, getNumberOfNodes());
    file = getFileID(PARETO, getNumberOfFiles());
    insert(node);
  }
  if (random() < PING_PROBABILITY)
  {
    node = getNodeID(PARETO, getNumberOfNodes());
    ping(node);
  }
}

for (node=0; node < getNumberOfNodes(); node++)
{
  getNode(node).executeActions();
}
```

As it can be seen, nodes and files are identified and generated according to a Pareto distribution. At each time step, every node may execute several actions defined in the method *executeActions* as follows:

```
executeActions()
{
  // Check if there are messages to process
  processMessages();

  // Check if the time to receive pong messages
  // for a certain ping has expired
  checkEndPing();

  // Check if the time to receive query hit messages
  // for a certain query has expired
  // so it is possible to download a file
  executeGet();
}
```

The method *processMessages* triggers the actions according to the type of message received. For example, if the node receives a ping message, it will forward this message to all its neighbors and reply to the sender node with a pong message.

When a node sends a ping message to all its neighbors, it will wait for a certain time for ping replies. When this time expires, the node removes the neighbors that did not reply from its list of neighbors. The neighbors will not reply only in the case of a dynamic network since they might be offline. If the network is static, they will always reply. The method *checkEndPing* checks if there is some ping message whose waiting time has expired and takes the required actions.

When a node issues a query, the node will also wait for some time for query replies. The method *executeGet* chooses one of the hosts that positively replied to be the one to download the file from.

In the following sections, we describe how some other features of the Gnutella architecture are implemented in the simulator.

5.2.2 Network Topology

In a network, nodes connect to each other in a certain way, which can be organized or not. Recent research has found that some peer-to-peer networks such as Gnutella fall into a class of inhomogeneous networks called *scale-free networks*, where a few nodes have many

connections, but most nodes have only a few connections [KLS02]. Other research has discovered that such networks can also be modeled by small-world graphs [Hay00]. Two characteristics distinguish small-world networks: first, a small average path length, typical of random graphs (here path means shortest node-to-node path); second, a large clustering coefficient that is independent of network size. The clustering coefficient captures how many of a node's neighbors are connected to each other. One can picture a small world graph as constructed by loosely connecting a set of almost complete subgraphs [IRF02]. A small-world graph can be viewed in the middle of figure 5.1.

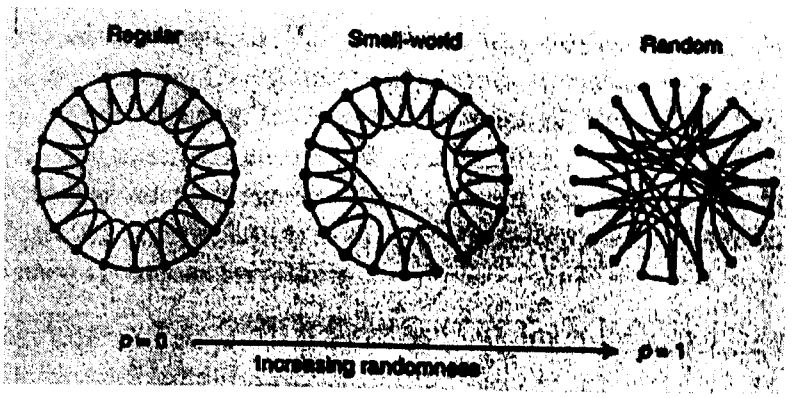

Figure 5.1: A regular lattice (Left) and a random graph (Right). Randomly rewiring just a few edges (center) reduces the average distance between nodes, L, but has little effect on the clustering coefficient. The result is a small-world graph.

Initially, the simulator attempts to create a small-world network by connecting the nodes in a ring-shape way. Therefore, each node connects itself to the previous and next one (considering the identifiers of nodes). There is also a small shortcut probability (5%) of creating an additional edge besides these two. In this case, the node is connected to a random node. It is important to emphasize that no effort was done in order to verify that this network is actually a small-world graph. We have just followed the principle that small-world networks can be constructed in this way [Hay00]. The simulator was implemented in a way that some few nodes are more likely to issue queries and to insert files than the others in the network. Consequently, as the simulations goes on, these nodes will have more connections to other nodes and will store more files. Doing like this, we model the *scale-free* aspect of the Gnutella network.

5.2.3 File characteristics

Previous research [SGG01, NRS+02] has shown that it is possible to find the average size of exchanged files in peer-to-peer systems. However, this research was made a few years ago and a lot has changed in the P2P world. From that time, different kinds of files started to be downloaded in P2P systems. With the spread of the DVD technology for instance, many

users started downloading and making video files available. This type of file is much larger than typical music files, so this characteristic must be taken into account when working with P2P systems. Considering these aspects, it was desirable that the simulator could work with files of different types.

The Gnutella simulator deals with three classes of files: music, sitcom and video. These classes were defined by a file size distribution, which model three different classes of multimedia files. As we have mentioned before, we believe that, in real peer-to-peer systems, these types of files are very popular and it is important to study the effect of the existence of these files in the simulations.

The distribution of file sizes in P2P systems has not been previously characterized. Previous studies /citeGribble, Diego have only come to an average size of files present in this kind of system, disconsidering the variety of types of files. The size of each type of file does not vary a lot. For example, music files have usually between 4.0MB and 5.0MB. For that reason, we consider normal distributions to be a suitable representation of this kind of data. Table 5.2 shows the parameters of such distributions as well as the percentage of files of each type used in our experiments.

Type of file	Average Size (MB)	Standard Deviation (MB)	Percentage of files
Music	4.5	0.5	70%
Sitcom	70	15	15%
Video	700	150	15%

Table 5.2: Distributions of file sizes

These data were obtained based on [NRS+02] and by observing files downloaded in real peer-to-peer systems. Unfortunately, it was not possible to find any work that presented accurate data or distributions on file sizes. Nevertheless, it is expected that normal distributions will represent these data appropriately.

The total number of unique files that can be exchanged in the whole simulation is fixed. It was very difficult to limit the number of unique files since we did not have any idea of how this number relates to the number of unique nodes in the network. We considered that the number of unique files is twice the size of the network.

At the beginning of the simulation, before nodes start to perform operations on the network, each node is assigned a random number of files limited to a maximum. This is a way of initializing nodes with a minimum number of files. In our experiments, the maximum initial number of files is five. Therefore, nodes can have from none to five files at the beginning of the simulation.

5.2.4 Node characteristics

Each node in the simulator represents almost every characteristic of real nodes in P2P systems.

Nodes must have a physical area where they can store files. Real peer-to-peer systems are usually concerned only with the amount of content nodes make available to be downloaded. However, some other important information, is how much of free disk space nodes

are willing to use to store their files. Unfortunately, no work that thoroughly described this feature was found. In spite of that, it is still relevant to represent nodes with different storage capacities.

In order to represent this, each node has a storage capacity. Each node is assigned a capacity according to a fixed probability. The simulator defines three different capacities described in table 5.3.

Type of capacity	Storage capacity(GB)	Probability
Low	1	20%
Medium	5	40%
High	10	40%

Table 5.3: Classes of nodes defined by storage capacities

The capacities were defined based on the idea that computers nowadays have great amount of free disk space, which means that they have a large area to store files from peer-to-peer systems. The size of the storage capacity is very important because it determines the number of files nodes will be able to make available to other nodes.

Some also important information on nodes is their bandwidth. The bandwidth determines how fast nodes download and upload files. [SGG01] has found that nodes are quite different with respect to bandwidth. While some hosts still use modem connections, others have cable or broadband connections. In the Gnutella simulator, there are three different classes of nodes, considering bandwidth. The value of each class of bandwidth and the number of nodes that have each type were defined based on the results found in [SGG01]. The possible bandwidths are shown in table 5.4.

Type of bandwidth	Bandwidth (Kbps)	Probability
Low	64	10%
Medium	256	60%
High	512	30%

Table 5.4: Classes of nodes defined by available bandwidth

It is important to highlight that there are no relations between storage capacity and bandwidth, that is, there might be nodes with great storage capacity and low bandwidth as well as nodes with low storage capacity and high bandwidth. One may think that in real P2P systems, the nodes that usually have great storage capacity are the ones that have very high-speed connections. This is usually true; however, we have not come up with a way of realisticly representing this in the simulator yet. This is left for future work.

Another interesting aspect is how nodes connect to each other as the simulation goes on. Since nodes in a Gnutella network connect to each other by sending a large volume of ping messages, it is possible that after some time, there will be a very large cluster of nodes, that is, a node might be connected to a great fraction of the whole network. It order to avoid this, the simulator limits the number of neighbors for each node. In our experiments, each node can be connected to a maximum of twenty other nodes. When this list is filled up, no more nodes can be added and no nodes are removed. A replacement mechanism

was not implemented yet, but we are already studying some methods of implementing this mechanism.

5.2.5 Messages

The simulator creates a network of a certain number of nodes and connects them to each other according to some topology. Once connected, the nodes start sending messages to each other. The messages are sent in the following situations:

- PING: ping messages are sent in order to discover other nodes in the network. In the simulation, every node may send ping messages to all its neighbors from time to time.
- QUERY: query messages are sent every time a node issues a request for a file.
- GET: get messages are sent when a node decides to download a file from a node.

Other messages are sent only in response to the ones previously mentioned. Next, we describe what actions nodes take when they receive these messages.

- PING: When a node receives a ping message and its time to live (TTL) is greater than zero, it broadcasts the message to all its neighbors and replies to the node that sent the ping with a pong.
- PONG: A pong is the answer to a ping. When a node receives a pong message, it adds the node which sent it to its list of neighbors if the limit size of the list was not reached. If the pong message was sent in response to a ping which was not originally sent by this node, it sends the message back along the same path.
- QUERY: When a query message is received, the node checks its list of files in order to verify if it has the file. If it does, the node sends a query hit message back to the node which emitted the query. Even if it does have the file, the query message is propagated to all its neighbors while the TTL is greater the zero.
- QUERYHIT: When a query hit message is received, the node checks if it was the one which sent the query. If it was, the node which sent the query hit is added to its list of candidate nodes from which the file can be downloaded. Right after the query is issued, the node starts waiting for replies for some time. The Gnutella Simulator defines this waiting time as twice the time-to-live so every node that processes one query will be able to send a hit message in time. When this time is finished, the node chooses a node to download the file from. This decision could be made based on different criteria, such as bandwidth, number of files available or hops taken to get to the node. In our experiments, bandwidth is the criterion to select the node. If the node which received the query hit message didn't issue the query, it just sends back the message through the path the query originally took.

5.3 Freenet experimental results

We present here the results obtained with the use of the algorithm based on communities of interests in a Freenet network. The main difference between this algorithm and the routing mechanism of the Freenet network is that the nodes that are first queried for files are the ones that belong to a certain community, thus sharing interests.

It is important to emphasize that other researchers built the Aurora simulator. We implemented another request mechanism using the simulator in order to evaluate the performance of the algorithm we proposed.

Table 5.5 shows the general information for the Freenet experiments we have performed.

Network size (number of nodes)	1000
Simulation time (number of steps)	5000
Request probability per timestep	50%
Insertion probability per timestep	50%
Generation of nodes	Uniform
Generation of files	Uniform
Initial topology	Ring
Maximum number of documents per node	50
Routing table size	200
Maximum community size	20

Table 5.5: General Information - Freenet Experiments

The simulator runs ten trials and averages their results. All the results presented are averages of 10 runs.

The purpose of our algorithm is to make the search process more efficient, that is, a file is expected to be found in fewer hops. The first parameter analyzed was the average request path length. It is expected that the algorithm we proposed decreases the value of this property. Figure 5.2 shows the time evolution of the average request path length in a Freenet network when the original search mechanism and the algorithm of communities of interests is used.

It can be seen from figure 5.2 that in both cases the average request path length drops through the simulation. As explained before, this is due to the fact that the network is trained over the time of the simulation. In a Freenet network, the nodes tend to be clustered according to the key identifier of the files they store, that is why files are found faster (in fewer hops) through time. It can also be observed that the communities algorithm obtained a better result compared to the original one. As time evolves, the difference in the average request path length of the algorithms increases converging to a reduction of 20This is because the communities grow during the simulation and therefore they are used more often as the end of the simulation is closer. In conclusion, it can be said that the communities algorithm proved to be more efficient than the original routing mechanism of the Freenet network.

Figure 5.3 shows the composition of the communities in different time steps of the simulation. As it can be seen, at the beginning of the simulation, most communities are composed of just one or two nodes and few communities are composed of more than five

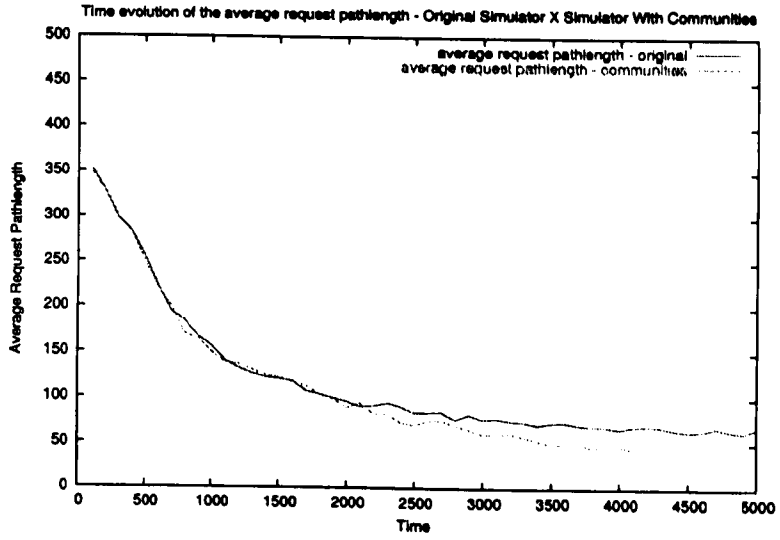

Figure 5.2: Time Evolution of the Average Request Path length - Original X Communities

nodes. On the other hand, at the end of the simulation, the majority of the communities are composed of more than five nodes. We can conclude that at the end of the simulation, the communities are more representative of a group's interests and are more useful to reduce the average number of hops taken to find files.

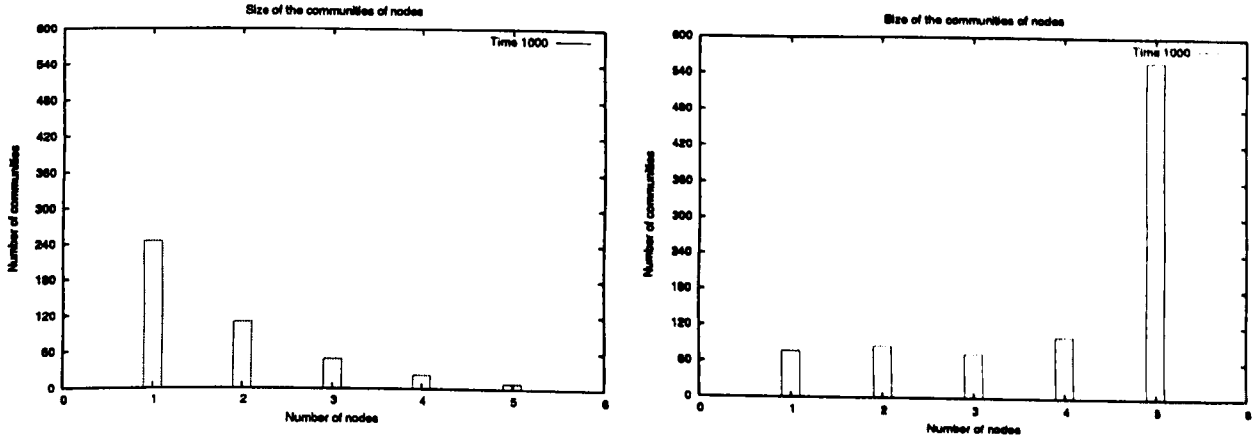

Figure 5.3: Size of the Communities at different time steps - Left: Time step 1000. Right: Time step 5000

Since the algorithm is intended to decrease the average request path length, it is clear that more requests will be successful in fewer hops. Figure 5.4 plots the cumulative distribution of the number of hops per request at different time steps of the simulation.

Figure 5.4 illustrates the cumulative distribution of the number of hops per query. It is desirable that a great number of requests be answered in a few hops. It can be observed

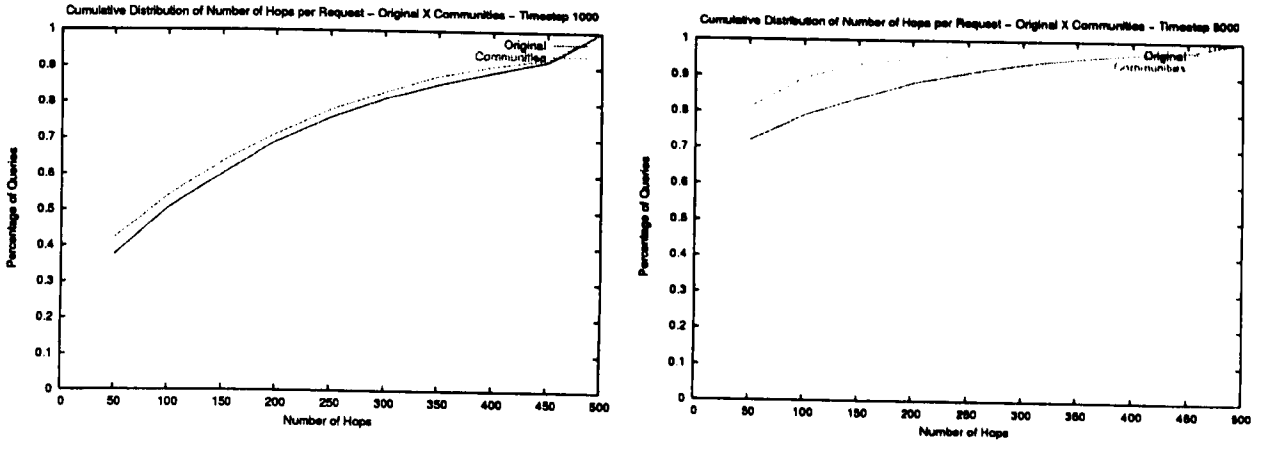

Figure 5.4: Cumulative Distribution of the Number of Hops per Query. Left: Time step 1000. Right: Time step 5000

from figure 5.4 that as time evolves, more requests are answered in less hops. In can also be seen that as the end of the simulation is reached, the benefits of the communities algorithm are clearer because more queries are solved in fewer hops. It is important to emphasize the difference between path length and number of hops. The path length is the minimum path required to find a file. It is typically the number of hops taken to bring the requested file from the node that stores it to the one that is searching it. The number of hops of a query includes not only these hops but also the ones taken to find out that some nodes did not store this file. Hence, the number of hops tends to be considerably greater than the request pathlength since it includes failed attempts to find the file.

It was shown that the size of the communities of nodes increases with the simulation, but we are also interested in knowing if the requests are mainly solved by the use of the original routing mechanism of the Freenet network or by the communities of nodes. At the end of the simulation, 16,24% of the files could be found in the community of nodes.

The data presented in this section provide a clear understanding of the benefits of the use of the algorithm of identification of communities of interests in a Freenet network.

5.4 Gnutella experimental results

We present here the results obtained with the use of the algorithm based on communities of interests in a Gnutella network. Since we have completely implemented the Gnutella Simulator, it was possible to collect more data than in the case of the Aurora Simulator. The main difference between this algorithm and the original routing mechanism of the Gnutella network is that the nodes that are queried first for files are the ones that belong to a certain community, thus sharing interests. We show that by using this new algorithm, files can be found in fewer hops, reducing the time taken to find them. Another benefit of the use of the algorithm is the load reduction since files are more likely to be found in the

community rather than in the other nodes of the network.

Table 5.6 presents general parameters for the Gnutella experiments we have performed.

Network size	1000
Simulation time	10000
TTL of all messages	7
Request probability per timestep	50%
Initial insertion probability per timestep	25%
Insertion probability per timestep	5%
Ping probability per timestep	10%
Maximum number of neighbors	20
Generation of nodes	Pareto
Generation of files	Pareto
Initial topology	Ring
Number of unique files	2000
Maximum number of initial files	5
Shortcut probability	5%
Maximum community size	10

Table 5.6: General Information - Gnutella Experiments

The following sections show the results of the use of the algorithm of communities of interests in a Gnutella network. We compare the performance of the network when the original search mechanism (flooding) is used with the one we have proposed based on the communities of interests.

The results presented as follows were obtained by averaging the data collected in ten runs of the simulator.

5.4.1 Distribution of files in the network

As we have explained before, the identifiers of the files that are requested or inserted in the network are generated by a Pareto distribution. This distribution simulates the fact that a few files are very popular while the majority of them are not. Figure 5.5 shows how each file is distributed over the network at the end of the simulation. The number of copies of files was ranked in decreasing order. There are few files that are located in many nodes and a lot of files that are stored in just a few nodes. Since the requests and insertions for files follow a Pareto distribution, this result was expected. We can observe a linear trend in log-log space, which is characteristic of distributions that follow Power-laws, such as Pareto.

We show these data both when using the original Gnutella search mechanism and when using the algorithm of identification of communities of interests with the sole purpose of showing that our algorithms did not alter these data as expected.

5.4.2 Content

The choice of which nodes perform operations on the network is made based on a Pareto distribution. This distribution simulates the fact that some nodes are quite more active

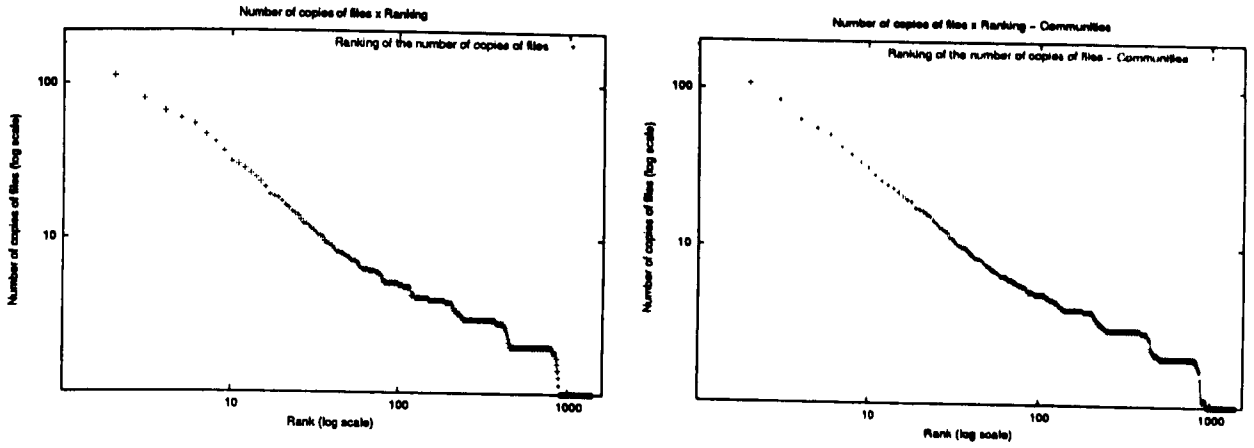

Figure 5.5: Ranking of the number of copies of files. Left: Original Gnutella Algorithm; Right: Communities of Interests Algorithm

than others, that means they insert and request more files than the rest of the network. Knowing how nodes use their storage capacity is important so we can detect if there are nodes that contribute more than others. Figure 5.6 shows the average number of files per node at the end of the simulation. Each line in the graph refers to a class of node, according to its storage capacity. Since the choice of which node is going to perform queries and insertions is made based on a Pareto distribution, there are a few nodes that store a lot of files and many nodes that store almost no files as expected.

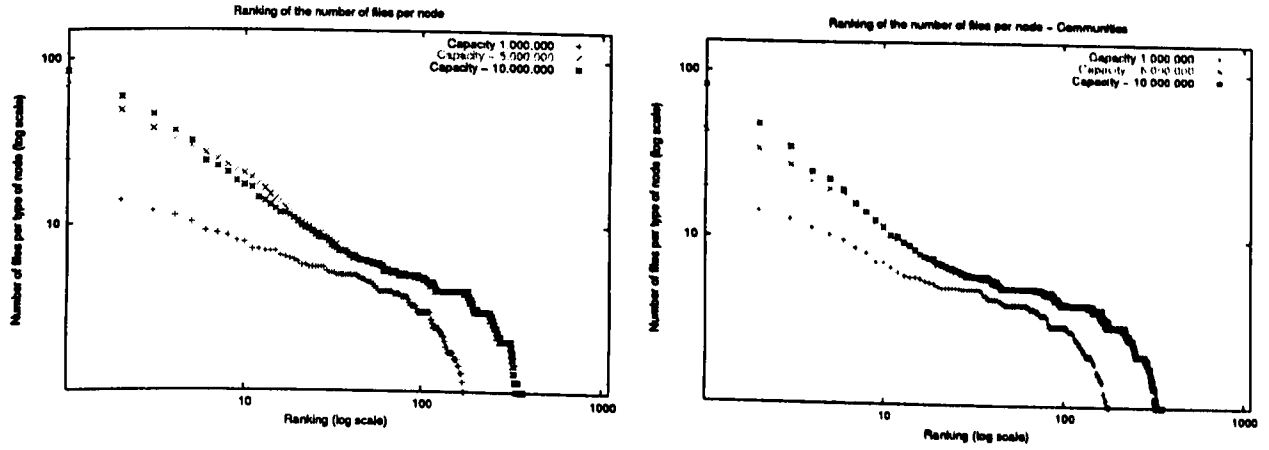

Figure 5.6: Average number of files per node. Left: Original Gnutella Algorithm; Right: Communities of Interests Algorithm

5.4.3 Connectivity

The analysis of a node's connectivity as well as the overall connectivity of the network is of considerable importance since it greatly influences the search process. If nodes are poorly connected, it is more likely that a file won't be found even if it is stored in some node of the network. Figure 5.7 illustrates the time evolution of the average number of neighbors and its confidence interval (90%). As it can be seen, the average number of neighbors grows rapidly after some time of the simulation and then it grows smoothly. This rapid growth is due to the way that nodes connect to each other. As we described, from time to time, a certain node sends a ping message to all its neighbors. This message is propagated to other nodes while the time to live is greater than zero. As time evolves, more nodes send ping messages and fill completely their list of neighbors. Because of this, the confidence interval reduces with time. At the end of the simulation, almost every node is connected to the maximum number of possible neighbors.

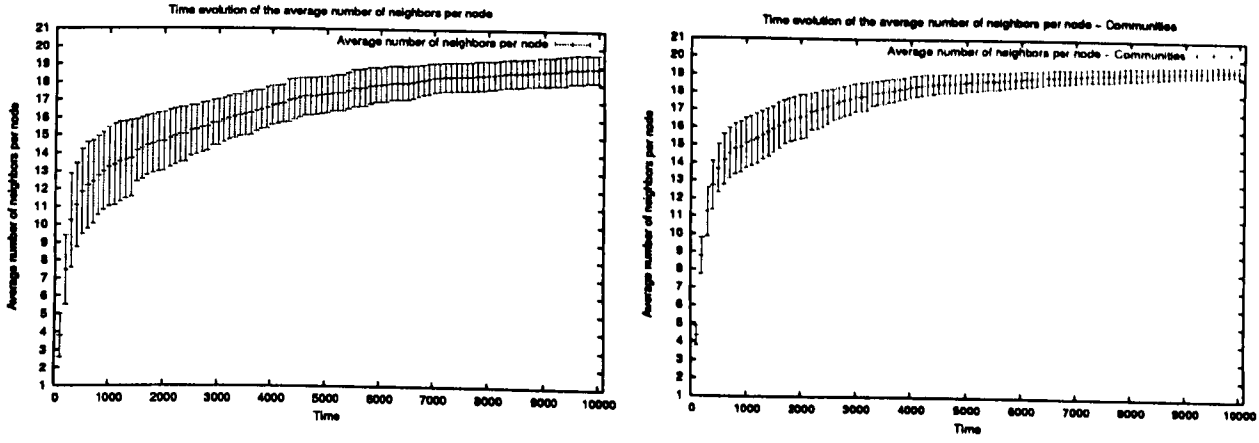

Figure 5.7: Average number of neighbors. Left: Original Gnutella Algorithm; Right: Communities of Interests Algorithm

5.4.4 Query popularity

The identifiers of files that are requested in the Gnutella simulator are generated by a Pareto distribution. This simulates the fact that there are a few files that are very popular and that many other files are quite less popular. Figure 5.8 shows the popularity of every query issued during the simulation. The popularity of the queries reflects that of the distribution of files in the network. Each query is represented by the identifier of the file being searched. As it can be seen from this figure, there are a few queries that are quite popular and a lot that occur very infrequently through the simulation.

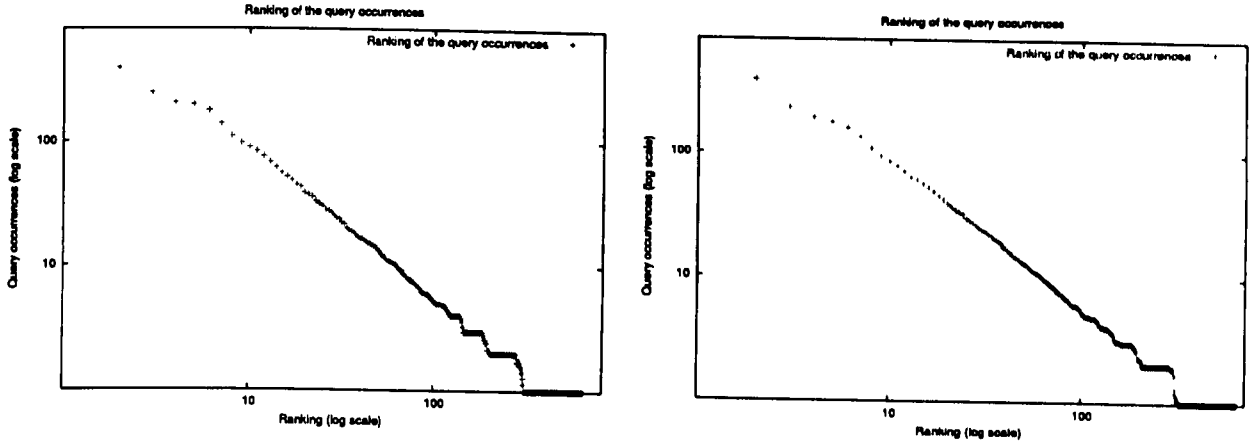

Figure 5.8: Query popularity. Left: Original Gnutella Algorithm; Right: Communities of Interests Algorithm

5.4.5 Load

Load is measured as the number of query messages processed by each peer. The load at each peer is intended to be the least possible. Reducing the load at individual peers is desirable for scalability and it is one of the main goals of the algorithm of communities of interests. Figure 5.9 shows the time evolution of the average number of query messages in each node of the network when both the original and the communities of interests algorithms are applied. Besides the average load, figure 5.9 also depicts the confidence interval (90%). We can see that the load is reduced when the algorithm of communities of interests is applied by more than 20 % at the end of the simulation. Since queries are not broadcast to other nodes, they do not send this kind of message. The average number of messages is calculated as the sum of the messages processed by every node (since the beginning of the simulation) in the network divided by the total number of nodes.

Figure 5.9 does not show the average load produced by other types of messages, but we can say that the number of query messages is greater than the number of the other types of messages because request messages are more likely to be sent than ping messages (according to the probabilities set by the simulator). Besides this, we can mention that get messages are initially sent only some time after the beginning of the simulation. This is because it takes some time for the first download to occur (since nodes wait for some time for hit messages). Moreover, the average number of query hit messages is also reduced because when files are found in nodes in the community, they are the only ones that send hit messages. The number of ping and pong messages are very close since one is the response to the other.

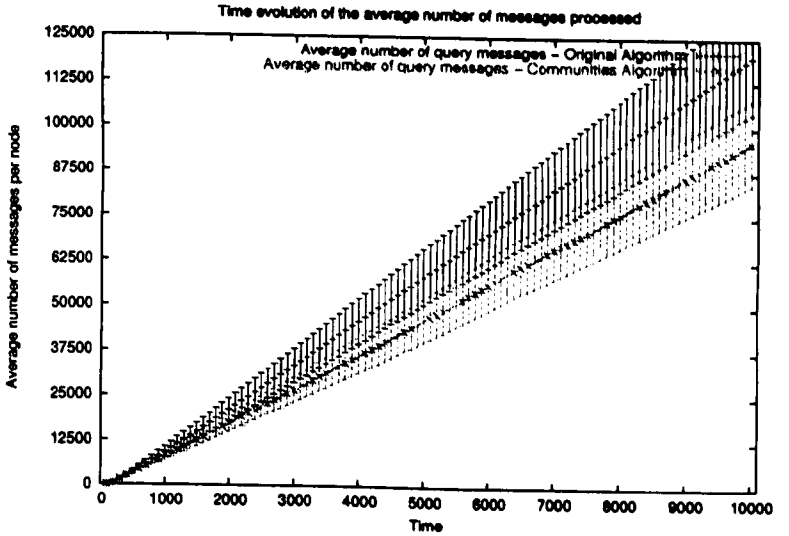

Figure 5.9: Average load (query messages) per node. Original Gnutella Algorithm x Communities of Interests Algorithm

5.4.6 Successful queries

A query for a file is successful when this file can be found in some node in the network. The percentage of successful queries is a fundamental criterion for the evaluation of the algorithm of communities of interests. Figure 5.10 shows the time evolution of the percentage of successful queries in the simulation comparing the original search mechanism with the algorithm of communities of interests. As it can be seen, at the beginning of the simulation few queries are answered successfully. This is because at this time, most nodes have just the files they have initiated the simulation with. As time evolves, more files are inserted, and more queries can be answered successfully. Another characteristic that influences the success of queries is how nodes connect to each other. At first, nodes are poorly connected. As time evolves, they are more connected and therefore it is possible to query more nodes for the same file. Comparing the performance of both algorithms, it is possible to see that the algorithm of communities of nodes raised the percentage of successful queries. This proves that nodes tend to cluster themselves in groups of interests. By querying the nodes in one's community, the chances of finding a file are increased.

5.4.7 Scope

The scope for a query is defined as the number (fraction) of peers in the system that process a particular query. For example, flooding may have a scope of approximately 100% because all peers (except those beyond the TTL limit) process the query. The use of the algorithm of communities of interests yields a much smaller scope if files can be found in nodes in the community. When they can't, they present a scope that can be even higher than flooding.

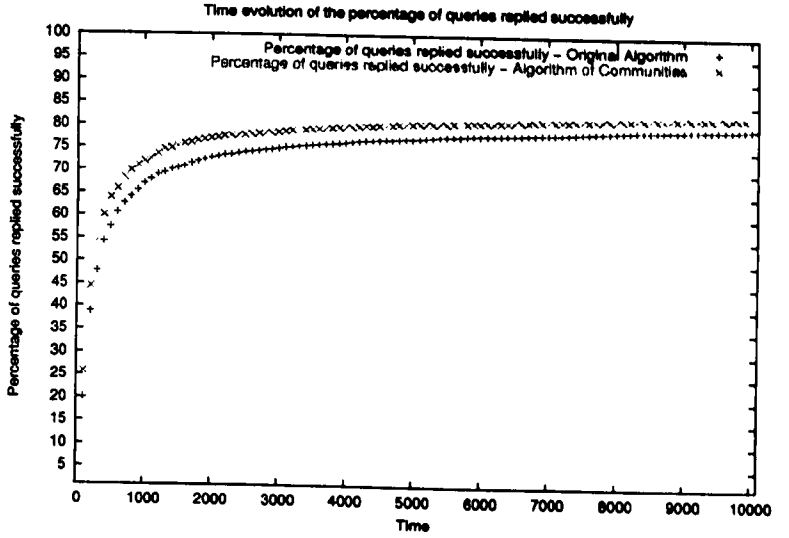

Figure 5.10: Percentage of successful queries. Top: Original Gnutella Algorithm; Bottom: Communities of Interests Algorithm

This is due to the fact that first the query is sent to the nodes in the community and then it is propagated to the rest of the network. Therefore, if the community is composed of nodes that are not reachable by the flooding of the query, the scope is a little bit higher In general, the use of the algorithm of communities of interests drastically reduces the scope of queries as figure 5.11 illustrates. It shows the time evolution of the average scope of queries and its confidence interval (90%) both when the original and the communities of interests algorithms are applied.. As it can be seen from figure 5.11, the average scope raises rapidly. This growth can be compared to the growth of the number of neighbors of each node. Since nodes get more connected after some time, queries are sent to more nodes as the simulation goes on.

5.4.8 Latency time

Latency time is defined as the time difference from the moment the query was sent to the moment the file began to be downloaded. In the original request mechanism, each node that issues a query waits for query responses for some time. Only after this time is expired, can the download start. The simulator defines this waiting time as twice the TTL, so that every node that receives the query is able to answer it in time. It is important to emphasize that every message takes one time step to be sent from one node to another regardless of the bandwidth of the two nodes involved in the transmission. Table 5.7 presents the average latency time when the original flooding mechanism is used as well as when the algorithm of communities of interests is applied. The low and high boundaries of the confidence interval (90%) are also shown.

In our experiments, when the algorithm of communities of interests is applied, if files

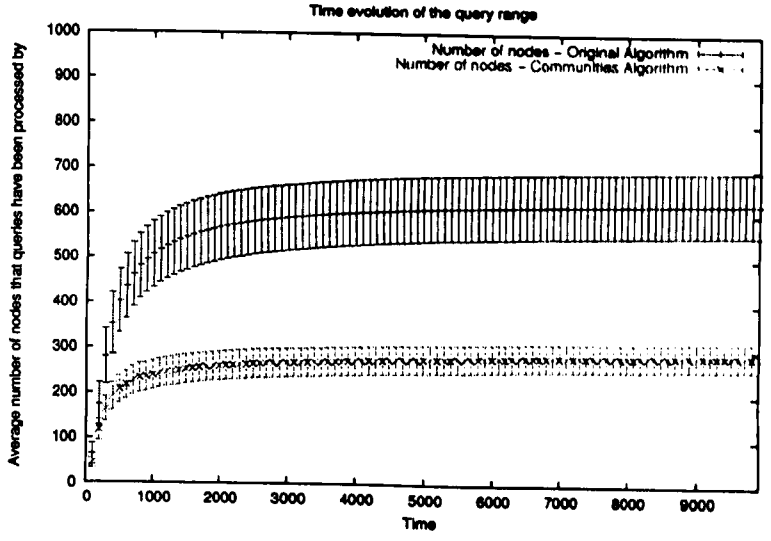

Figure 5.11: Scope. Original Gnutella Algorithm x Communities of Interests Algorithm

Algorithm	Average latency time (s)	Low boundary (s)	High boundary (s)
Original	14	-	-
Communities	10.75	10.37	11.20

Table 5.7: Latency time - Original Gnutella X Community of Interests Algorithm

are found in nodes within the community, the latency time for this query is 2 (one time step taken to send the query and another one to send the response). If they cannot be found within the community, the latency time is 16 (2 time steps to query the nodes in the community and 14 to wait for the hits coming from several nodes). In this case, the algorithm of communities performs worse than the flooding mechanism. However, the benefits when files can be found in the community overcome the drawbacks when they can't be found within the community. The average latency time is reduced by more than 20% with the use of the algorithm of communities of interests.

5.4.9 Download time

The download time of a file is defined as the time required to transfer a file from one node to another. It depends on the bandwidth of the two nodes involved as well as the size of the file being transferred. The bandwidth of the transmission is considered to be the least of those of the nodes involved. The download time heavily depends on the type of file being transferred since they have different sizes. Table 5.8 presents the average download time of each type of file and the boundaries of the confidence interval(90%).

Besides this, it is also relevant to present what the percentage of downloads per bandwidth is so we can know how the average transmission bandwidth affects the average download time. In table 5.9, we can see that most downloads are done through 256Kbps.

The data in tables 5.8 and 5.9 were obtained in experiments undertaken with the original flooding mechanism.

Type of file	Average download time	Low boundary	High boundary
Music	20.66	16.60	24.73
Sitcom	343.16	277.04	409.29
Video	2444.66	2314.14	2575.19

Table 5.8: Download time - Original Gnutella Algorithm

Bandwidth	Average percentage of downloads	Low boundary	High boundary
64	9.83	3.80	15.86
256	62.66	59.52	65.80
512	25.66	18.95	32.37

Table 5.9: Downloads per bandwidth - Original Gnutella Algorithm

The correspondent data obtained by using the algorithm of communities of interests is shown in tables 5.10 and 5.11. As it can be observed, the data obtained with the use of both algorithms are similar. These results were expected since the goal of the algorithm is not to reduce the download time but the latency time and the load in the network, in the case of the Gnutella network. We believe the difference in the results was due to the random character of the workload used.

Type of file	Average download time	Low boundary	High boundary
Music	22.20	19.64	24.76
Sitcom	336.20	305.60	366.82
Video	2531.80	2452.48	2611.12

Table 5.10: Download time - Communities of Interests algorithm

5.4.10 Community size

The average size of the communities is defined as the average number of nodes that comprise each community. The size of the communities influences the outcome of requests: the bigger the community, the higher the chances of finding the requested files in its nodes. Figure 5.12 shows the time evolution of the average size of the communities and its confidence interval (90%). It is possible to notice that the average size of communities grows smoothly. The communities evolve as more files are distributed through the network. The growth of the communities follows that of the spreading of files in the network. The slope of this graph is different from the one in figure 5.7 because it is independent of how connected nodes are. From figure 5.12, it is possible to notice that many nodes in the network have communities composed of very few elements. Moreover, at the end of the simulation, the average size of the communities is 3.5 nodes (the maximum allowed in our experiments is 10). It is clear that there should be a limit on the size of communities because there is a trade-off between the size of the community and load generated by its nodes. In future

Bandwidth	Average percentage of downloads	Low boundary	High boundary
64	10.30	6.11	14.49
256	75.50	70.60	80.40
512	12.50	8.54	16.46

Table 5.11: Downloads per bandwidth - Communities of Interests Algorithm

research, we intend to find out what is the maximum community size in order to optimize the performance of the search process in the network.

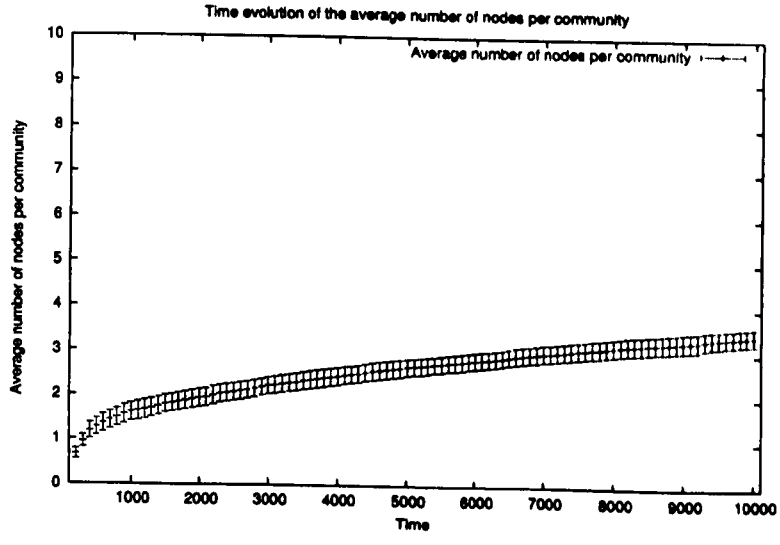

Figure 5.12: Average size of communities

The average size of communities is small because there are few nodes whose communities are composed of the maximum number of nodes allowed. Figure 5.13 presents a rank of the sizes of communities. As we can see, there are few nodes whose communities' size is the maximum allowed and a lot of nodes whose communities are composed of a few members.

As mentioned before, in our experiments, every community has a limited size. If the community reaches its full capacity, no more nodes are added or removed from it. Node replacement mechanisms are being studied and will be implemented as future work. Nodes can be replaced based on several criteria depending on the final goal. For example, if one intends to increase the probability of finding files in the community, nodes with more available content should be chosen to remain in the community. On the other hand, if download time is the parameter of choice, nodes with higher bandwidths may be kept longer in the community.

The identification of communities depends on the number of files a node stores. If a node stores a small number of files, it will have reduced chances of finding many nodes with which it shares interests. On the other hand, nodes that store a large volume of files stand high chances of having a community with the maximum size allowed. We can conclude that possibly nodes that share more files benefit more from the use of communities of interests.

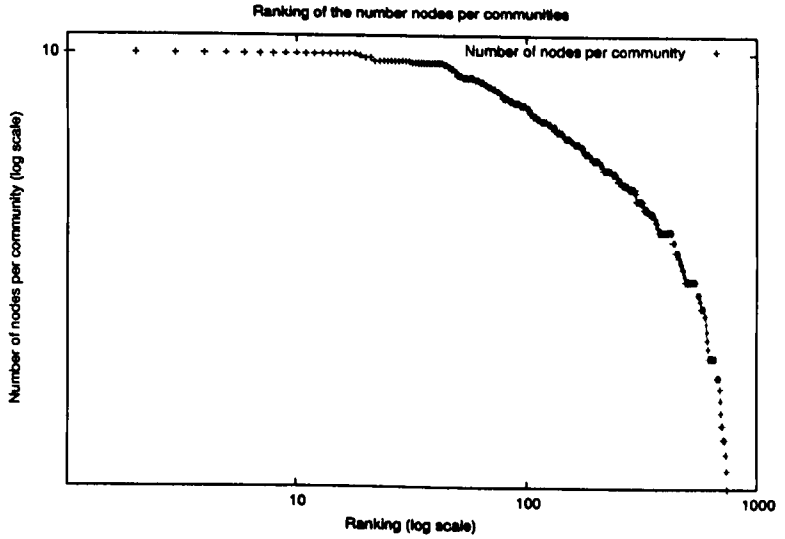

Figure 5.13: Ranking of the sizes of communities

Since these nodes are possibly the most active in the network (the ones that send queries more often), the overall performance of the algorithm is improved.

The next chapter presents a summary of the results achieved and some suggestions for future work.

Chapter 6

Conclusions

This chapter presents our conclusions on this research. It summarizes the results obtained so far and describes some suggestions for future work.

6.1 Summary of the results

The popularity of P2P file sharing applications has created a flurry of recent research activity into P2P architectures [SGG01]. These networks have become, in a short period of time, one of the fastest growing and most popular Internet applications [CFK03].

The key to the usability of a data-sharing p2p system and one of the most challenging design aspect, is efficient techniques for search and retrieval of data. Few papers [SMZ03, SMZ01] have analyzed the importance of taking each node's interest into account during the search process. In this work, we developed and analyzed a new search algorithm that is based on interests. Two nodes have interests in common if they store some identical files. It was shown that a node stands greater chance of finding a file in another one if they share interests(files). The algorithm defines one way to identify communities of nodes with the same interests. Searching the communities of nodes instead of the other nodes in the network proved to be an efficient way of finding files in less hops and thus avoiding load on the network.

It is important to emphasize that the use of the communities of interests is just an attempt to improve the overall performance of the search mechanisms of peer-to-peer networks. If a document cannot be located via communities of interests, it can always be located via the underlying overlay.

The experiments were undertaken by using a Freenet and a Gnutella simulator. The results obtained so far were quite satisfactory and the use of node's interests in search mechanisms in peer-to-peer systems proved to be a very promising area of research.

It is possible to outline our main achievements:

- In Freenet, the average request path length was reduced in 20%.

- In Freenet, more queries could be successfully terminated in less hops. For example, using the original Freenet search mechanism, 70% of the queries took up to 50 hops. By using the algorithm of communities of hops, it was possible to successfully terminate 80% of the queries in less than 50 hops.
- In Gnutella, the load caused by query messages could be reduced in more than 20%. We believe it is possible to reduce it even more with the introduction of slight modifications in the algorithm of communities of interests. The load caused by query hit messages was reduced by almost 50%.
- The percentage of successful queries was increased. This proves that the communities are composed of nodes that may not be reachable by the flooding mechanism.
- The average scope of queries was reduced in more than 50%. This is another result that shows that the average load in the network has decreased.
- The use of the algorithm of communities of interests did not have a negative impact on other characteristics of the network, such as the download time or the connectivity of the network.

The peer-to-peer research area has a lot of challenges, one of the greatest being the lookup problem. This research presented a new algorithm to solve this problem and improve the performance of P2P networks. Another contribution of this research was the implementation of a Gnutella simulator. This simulator is open-source and can be used for the evaluation of other research topics in peer-to-peer research. We consider it an invaluable tool for the ones who are willing to dig into the fabulous peer-to-peer world.

The following section presents our suggestions for future work.

6.2 Future Work

We believe this work was a first step towards improving content location in peer-to-peer systems. The use of the algorithm of communities of interests proved to be efficient but still more analyses can be done. The following subsections present possible improvement in different areas.

6.2.1 Workload

The first improvement that can be done in our experiments is related to the workload. The experiments were undertaken using statistical distributions such as Pareto and even the Uniform Distribution. Since the main goal of this work was the evaluation of the algorithm, we believe that the use of these distributions did not heavily influence our achievements. Despite of this, we still believe it would be desirable to run the experiments using a real workload, for example, user-trace logs from any known peer-to-peer system.

One of the main difficulties is obtaining a workload like this. Some effort was made during this research to obtain workload from the Gnutella network. Since Gnutella is an open protocol, it is possible to modify one client so that it will log every message that it receives and sends. Until now, this was not an effective way of obtaining real data since it is hard to get data on the whole network. This provides information about only one host of the network. Another possible workload is traces from proxy servers. Available traces that could be used are the ones from the Boeing corporate proxy traces [Mea99]. The Gnutella simulator is already implemented to work with traces.

6.2.2 Dynamic participation

The routing strategies described here have all been analyzed under static conditions. A key area for future analyses is the effect of relatively frequent node joins and departures in the system. We believe that even relatively modest costs for these operations could end up dominating overall performance [BKK+03].

It is possible to simulate the dynamic behavior of nodes in a peer-to-peer system in a variety of ways. The Gnutella Simulator assumes the following situation: nodes can participate in the network either statically or dynamically. If the network is static, all nodes are online all over the simulation. Otherwise, every node may be disconnected from the network according to a probability. Once offline, it may continue unavailable or become online again according to another probability. One possible improvement is to relate the availability of nodes to other characteristics such as amount of data shared, number of neighbors, etc...

6.2.3 Replacement policies

One of the limitations of the simulation is the size of the list of neighbors of each node. Although it is clear that the number of neighbors should be limited, the static behavior of the list is not reasonable. We intend to come up with replacement mechanisms for the list of neighbors. As an initial proposal, we could define that when the list of neighbors is completely full and a node finds another possible neighbor, one of the neighbors of the node could be selected to be removed from the list. The removal could be based on different criteria such as bandwidth or amount of available content. The same criteria could be applied for the replacement mechanisms of nodes in the communities of interests.

6.2.4 Possible scenarios

The Gnutella Simulator works with a static network in the sense that no nodes are added or removed from the network during the simulation. We would like to evaluate the performance of the algorithm of identification of communities of interests under different conditions.

Failures in P2P systems are viewed as nodes suddenly and unexpectedly dropping out of the network. Networks such as Gnutella are very robust to failures of random nodes. This robustness is rooted in the inhomogeneous nature of the network. Nodes with few connections will fail with much greater probability than nodes with many connections. Failure detection is the process of detecting isolated random failures of peers on the network. Current Gnutella clients detect a failure when one of their neighbors stops responding to them unexpectedly. The only information that can be derived from this process of detection is that a neighbor has died. There is no indication of how significant the neighbor was in terms of connections to other nodes and how important it was in holding the graph together. Attack detection, on the other hand, is the process of detecting an attack on the network as a whole. Methods for attack detection could take into account the importance of nodes being lost in the system or the frequency of node loss [KLS02].

Considering this, we also intend to explore other scenarios in our experiments, such as network growth and attack. The study of these scenarios is quite important since they represent real scenarios. Peer-to-peer architectures are currently growing really fast and hence the evaluation of new search algorithms under these conditions is quite important.

6.2.5 Network topology

As we have mentioned, the network topology heavily influences the performance of the search mechanism used in a peer-to-peer system. If nodes are highly connected, files can be found in fewer hops and queries are successful more often. In this work, we only exploited the ring topology with shortcuts in order to simulate a small-world network. We intend to exploit different topologies in the future such as a random topology.

6.2.6 Communities

We believe it is possible to find nodes with similar interests by using different variations of the same algorithm. One possible interesting area of study would be the identification of groups with similar interests in the whole network. The algorithm of identification of communities of interests we have presented here defines one community per seed node. Another approach would be to identify a general group of nodes that share interests, regardless of the choice of the seed node. This could be viewed as the identification of global communities of interests. The process of identification of communities of interests is done based on stored files. One possible improvement would be to also consider nodes that successfully answer queries in a similar way as [SMZ03] does. Finally, it would be interesting to implement the algorithm of identification of communities of interests in other peer-to-peer networks.

Bibliography

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. In *First Monday*, October 2000.

[ALPH01] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 2001.

[BKK+03] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, February 2003.

[CF02] Jeffrey Considine and Thomas A. Florio. Scalable peer-to-peer indexing with constant state. Technical report, University of Berkeley, August 2002.

[CFK03] Edith Cohen, Amos Fiat, and Haim Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. *Infocom*, 2003.

[CHMaW] Ian Clarke, Theodore W. Hong, Scott G. Miller, and Oskar Sandberg and-Brandon Wiley. Protecting freedom of information online with freenet. for review purposes only.

[CSWW00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W.Hong. Freenet: A distributed anonymous informationstorage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[DBK+01] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, DavidKarger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *8th IEEE Workshop on Hot Topics in Operating Systems*, pages 71–76, Elmau/Oberbayern, May 2001. MIT Laboratory for Computer Science.

[FLG00] Gary W. Flakes, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. *ACM KDD*, 2000.

[FLGC02] Gary W. Flakes, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 2002.

[GKR98] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.

[Hay00] Brian Hayes. Graph theory in practice: Part ii. *American Scientist*, 88(1), January 2000.

[IRF02] Adriana Iamnitchi, Matei Ripeanu, and Ian Foster. Location data in (small world?) peer-to-peer scientific collaborations. *First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, March 2002.

[KLS02] Pedram Keyani, Brian Larson, and Muthukumar Senthil. Peer pressure: Distributed recovery from attacks in peer-to-peer systems. *IFIP Workshop on Peer-to-Peer Computing*, 2002.

[Kub03] John Kubiatowicz. Extracting guarantees from chaos. *Communications of the ACM*, February 2003.

[Lan01] Adam Langley. *Freenet*, chapter 1. O'Reilly, 2001.

[Lee03] Jintae Lee. An end-user perspective on file sharing systems. *Communications of the ACM*, February 2003.

[Let03] Richard Lethin. Technical and social components of peer-to-peer computing. *Communications of the ACM*, February 2003.

[MA01] Daniel A. Menasce and Virgilio A. F. Almeida. *Capacity Planning for Web Services. Metrics, Models and Methods*, chapter 2. Prentice Hall, 2001.

[Mar02] E. P. Markatos. Tracing a large scale peer to peer system: an hour in life of gnutella. *Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[Mea99] J. Meadows, March 1999. ftp://researchsmp2.cc.vt.edu/pub/boeing.

[Mil67] Stanley Milgram. The small-world problem. *Psychology Today*, 1967.

[NRS+02] D. Nogueira, L. Rocha, J. Santos, P. Araujo, V. Almeida, and W. Meira Jr. A methodology for workload characterization of file-sharing peer-to-peer systems. *IEEE Workshop on Workload Characterization*, 2002.

[PRU01] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *42nd annual IEEE Symposium on the Foundations of ComputerScience (FOCS)*, 2001.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[RF02] Matei Ripeanu and Ian Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *IPTPS*, 2002.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, RichardKarp, and Scott Shenker. A scalable content-addressable network. In *ACM Sigcomm 2001 Technical Conference*, San Diego, CA, August 2001.

[Rip01] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical report, University of Chicago, 2001.

[SGG01] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical report, University of Washington, 2001.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. FransKaashoek, and Hari Balakrisnan. Chord: A scalablepeer-to-peer lookup service for internet applications. In *ACI Sigcomm 2001*, San Diego, CA, August 2001.

[SMZ01] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Enabling efficient content location and retrieval in peer-to-peer systems. Technical report, Carnegie Mellon University, 2001.

[SMZ03] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Efficient content location using interest-based locality in peer-to-peer systems. *Infocom*, 2003.

[Sri01] Kunwadee Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. Technical report, Carnegie Mellon University, 2001.

[Thea] The aurora home page. http://www.doc.ic.ac.uk/ twh1/longitude/.

[Theb] http://freenet.sourceforge.net.

[Thec] The gnutella protocol specification. http://www.clip2.com.

[Thed] The kazaa home page. http://www.kazaa.com.

[Thee] The morpheus home page. http://www.morpheus.com.

[Thef] The napster home page. http://www.napster.com.

[VBKJ02] Jean Vaucher, Gilbert Babin, Peter Kropf, and Thierry Jouve. Experimenting with gnutella communities. *Distributed Communities on the Web International Workshop*, 2002.

[Whi02] John Whitfield. Search engines makes social calls. *Nature*, March 2002.

[YGM01] Beverly Yang and Hector Garcia-Molina. Efficient search in peer-to-peer networks. Technical report, Stanford University, October 2001.

[ZKJ00] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of Berkeley, April 2000.