2007

Design and evaluation of new search paradigms and power management for peer-to-peer file sharing

Graciela Perera
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd
Part of the American Studies Commons

Scholar Commons Citation

This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Design and Evaluation of New Search Paradigms and Power Management for Peer-to-Peer File Sharing

by

Graciela Perera

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Ken Christensen, Ph.D.
Miguel Labrador, Ph.D.
Rafael Perez, Ph.D.
Dewey Rundus, Ph.D.
Gregory McColm, Ph.D.
Nagarajan Ranganathan, Ph.D.
Wilfrido Moreno, Ph.D.

Date of Approval:
May 21, 2007

Keywords: P2P, protocols, networks, energy efficiency, performance evaluation

© Copyright 2007, Graciela Perera
Dedication

To God

To the Blessed Virgin Mary mother of God under her titles of Virgin of Coromoto and Mystical Rose.

To my father and mother, Sergio and Graciela.
Acknowledgements

I would like to acknowledge the support, advice and many hours of work from my advisor, Dr. Kenneth J. Christensen. Without his dedication, it would not have been possible to successfully complete this dissertation. He provided many valuable ideas and lessons throughout my Ph.D. studies that I will not forget. I would also like to thank my committee: Dr. Miguel Labrador, Dr. Rafael Perez, Dr. Dewey Rundus, Dr. Nagarajan Ranganathan, Dr. Wilfrido Moreno, and Dr. Gregory McColm. I would like to thank Dr. Rafael Perez for his time and guidance throughout my Ph.D. program, and Dr. Dewey Rundus for his informative morning talks about research and academia. A very special thanks to Dr. Gregory McColm for his valuable time and enthusiasm, as well as for the example he set for me to follow. I would also like to express my deepest gratitude to my father, mother, sisters and nephews for their unconditional support, love and understanding.
# Table of Contents

List of Tables ..................................................................................................................... iv  
List of Figures ...................................................................................................................... v  
Abstract ............................................................................................................................ viii  

Chapter 1: Introduction ....................................................................................................... 1  
1.1 Background .............................................................................................................. 1  
1.2 Motivation ............................................................................................................... 5  
1.3 Contributions ........................................................................................................... 6  
1.4 Organization of this Dissertation ......................................................................... 7  

Chapter 2: Literature Review .............................................................................................. 9  
2.1 A History and Overview of P2P File Sharing .......................................................... 9  
2.1.1 P2P Predecessors .............................................................................................. 10  
2.1.2 Structured P2P Networks .................................................................................. 12  
2.1.2.1 Chord .......................................................................................................... 13  
2.1.2.2 Content Addressable Network ...................................................................... 14  
2.1.2.3 Pastry and Tapestry .................................................................................... 16  
2.1.3 Unstructured P2P Networks ............................................................................. 18  
2.1.3.1 Napster ....................................................................................................... 18  
2.1.3.2 Gnutella ....................................................................................................... 19  
2.1.3.3 Freenet ........................................................................................................ 20  
2.1.3.4 Fasttrack (Kazaa) ....................................................................................... 21  
2.1.3.5 BitTorrent ................................................................................................... 22  
2.1.4 Ethical Issues in P2P Networks ......................................................................... 23  
2.2 Gnutella Details ........................................................................................................ 24  
2.2.1 Gnutella Version 0.4 ....................................................................................... 24  
2.2.2 Gnutella Version 0.6 ....................................................................................... 30  
2.3 Improving Search in P2P Networks ......................................................................... 36  
2.3.1 Exploiting Power-Law Properties .................................................................... 38  
2.3.2 Reducing Query Traffic .................................................................................... 39  
2.3.3 Improvements in P2P File Sharing Networks ............................................... 42  
2.4 Energy Use of P2P Networks .................................................................................. 44  
2.5 Characterization of File Distribution in P2P Networks ........................................... 46
2.6 Overview of Set Cover Algorithms........................................................................49
  2.6.1 The Set Cover Problem as NP Complete.........................................................50
  2.6.2 The Greedy Algorithm....................................................................................51

Chapter 3: Exploiting Known File Distributions – Targeted Search..........................54
  3.1 Premise and Promise of Heavy-Tailed File Distributions.....................................54
  3.2 New Targeted Search Method..............................................................................56
    3.2.1 Proof of Optimality.....................................................................................57
  3.3 Performance Evaluation of Targeted Search Method............................................61
    3.3.1 Analytical Models for Cost and Time..........................................................62
    3.3.2 Selection of Parameter Values......................................................................67
    3.3.3 Numerical Results......................................................................................68
    3.3.4 Discussion of Results.................................................................................69
  3.4 Implementation of Targeted Search.....................................................................71
    3.4.1 Gnutella-Compatible P2P Host.................................................................71

Chapter 4: Changing the Search Paradigm – BULLS....................................................75
  4.1 Broadcast Updates Local Look-Up Search – BULLS Protocol...............................75
  4.2 BULLS Protocol..................................................................................................77
    4.2.1 Description of BULLS...............................................................................78
  4.3 Flow Models for Gnutella and BULLS.................................................................84
    4.3.1 Flow Model for Gnutella.............................................................................88
      4.3.1.1 Storage Requirement..........................................................................89
      4.3.1.2 Overhead Traffic...............................................................................89
    4.3.2 Flow Model for BULLS..............................................................................90
      4.3.2.1 Storage Requirement..........................................................................90
      4.3.2.2 Overhead Traffic...............................................................................91
  4.4 Performance Evaluation of BULLS.....................................................................93
    4.4.1 Selection of Parameter Values......................................................................93
    4.4.2 Numerical Results – Representative Parameters........................................95
    4.4.3 Numerical Results – Ranged Parameters....................................................96
    4.4.4 Discussion of Results.................................................................................96

Chapter 5: Power Management in P2P File Sharing Networks...................................98
  5.1 Potential Savings from Power Management........................................................98
  5.2 Set Cover Model for P2P Power Management.....................................................100
  5.3 Random Map Out (RMO) - A Distributed Set Cover Algorithm..........................100
  5.4 Performance Evaluation of RMO.........................................................................103
    5.4.1 Selection of Parameter Values.....................................................................105
    5.4.2 Description of Experiments.........................................................................110
    5.4.3 Numerical Results......................................................................................111
      5.4.3.1 Experiment #1 – Representative Values.............................................112
      5.4.3.2 Experiment #2 – Number of Hosts.....................................................114
      5.4.3.3 Experiment #3 – Number of Files......................................................118
    5.4.4 Summary and Discussion of Results............................................................121
List of Tables

Table 2.1 Gnutella messages.................................................................27
Table 5.1 Experiment #1-Representative values for one processor........113
Table 5.2 Experiment #1-Representative values for $M$ processors........113
List of Figures

Figure 1.1 P2P file sharing network ................................................................. 3
Figure 2.1 DNS hierarchy of local, root, and authoritative name servers [66] ....... 11
Figure 2.2 Mapping of three hosts to a Chord circle with 8 identifiers [108] ........... 14
Figure 2.3 A 2-dimensional coordinate space partitioned for 5 CAN hosts [69] ...... 15
Figure 2.4 Message routing in Pastry from host 37A0F1 to find key B57B2D [6] .... 16
Figure 2.5 Three steps to locate and download a file in Napster [46] .................. 19
Figure 2.6 Three steps to download file Choping.mp3 in BitTorrent ................ 23
Figure 2.7 Gnutella version 0.4 FSM ................................................................. 28
Figure 2.8 Gnutella version 0.6 ultrapeer FSM .............................................. 33
Figure 2.9 Gnutella version 0.6 leaf FSM .......................................................... 35
Figure 2.10 Set cover problem definition [28] .................................................. 49
Figure 2.11 Resource allocation example for set cover problem ......................... 51
Figure 2.12 Greedy algorithm [28] ................................................................. 52
Figure 2.13 Detailed description of Greedy algorithm ....................................... 53
Figure 3.1 P2P network where shared files distribution is heavy-tailed .......... 55
Figure 3.2 Targeted Search method ................................................................. 56
Figure 3.3 Variables for model of Targeted Search ........................................... 61
Figure 3.4 Targeted Search mean time results ............................................... 66
Figure 3.5 Targeted Search mean cost results ............................................... 67
Figure 5.11 Experiment #2-One processor with $K$ slightly peaked.........................115
Figure 5.12 Experiment #2-One processor with $K$ highly peaked .........................115
Figure 5.13 Experiment #2-$M$ processors with $K$ uniform..................................116
Figure 5.14 Experiment #2-$M$ processors with $K$ slightly peaked.........................117
Figure 5.15 Experiment #2-$M$ processors with $K$ highly peaked........................117
Figure 5.16 Experiment #3-One processor with $K$ uniform.................................118
Figure 5.17 Experiment #3-One processor with $K$ slightly peaked.........................119
Figure 5.18 Experiment #3-One processor with $K$ highly peaked..........................119
Figure 5.19 Experiment #3-$M$ processors with $K$ uniform..................................120
Figure 5.20 Experiment #3-$M$ processors with $K$ slightly peaked.........................120
Figure 5.21 Experiment #3-$M$ processors with $K$ highly peaked..........................121
Figure 5.22 Conditions for RMO to lose file $b$ .......................................................124
Figure 5.23 Machine failure problem definition .....................................................126
Figure 5.24 Probability of a lost file ($P_{T,0}$) .........................................................128
Design and Evaluation of New Search Paradigms and Power Management for Peer-to-Peer File Sharing

Graciela Perera

ABSTRACT

Current estimates are that more than nine million PCs in the U.S. are part of peer-to-peer (P2P) file sharing overlay networks on the Internet. These P2P hosts generate about 20% of the traffic on the Internet and consume about 7.8 TWh/yr equal to $630 million per year. File search in a P2P network is based on a wasteful paradigm of broadcasting query messages. Reducing P2P overhead traffic to reduce bandwidth waste and enabling power management to reduce electricity usage are clearly of great interest.

In this dissertation, two new search paradigms with reduced overhead traffic are investigated. The new Targeted Search method uses statistics from previous searches to target future searches. Targeted Search is shown to reduce query overhead traffic when compared to broadcast-based search used by Gnutella. The new Broadcast Updates with Local Look-up Search (BULLS) protocol enables new capabilities including power management and reduces overhead traffic by enabling a local look-up of shared files. BULLS hosts periodically broadcast changes in their list of files shared and build a table of shared files by all other hosts.
Power management in P2P networks is studied as an application of the minimum set cover problem. A reduction in overall energy consumption is achieved by powering down hosts that have all of their shared files fully shared (or covered) by other hosts. A new set cover heuristic – called the Random Map Out (RMO) algorithm – is introduced and compared to the well-known Greedy heuristic. The algorithms are evaluated for minimum set cover size and computational complexity (number of comparisons). The RMO algorithm requires significantly less comparisons than Greedy and still achieves a set cover size within a few percent of that of Greedy. Additionally, the RMO algorithm can be distributed and independently executed by each host with reduced complexity per host where the Greedy heuristic does not reduce in complexity by being distributed. With RMO there is a non-zero probability of a given file being “lost” (not in set cover). The probability of this event is modeled and numerical results show that the probability of a file being lost is practically insignificant.
Chapter 1:

Introduction

1.1 Background

The Internet has changed the way people communicate and distribute information in all parts of the world. In 2005, about one billion people used the Internet [89] to exchange emails and instant messages, access information on web servers, and ever increasingly share files with Peer-to-Peer (P2P) networks [55, 109]. The emergence of P2P networks makes it possible for Internet users to be simultaneously sharing and accessing digital content in many forms like audio and video files.

The popularity and use of P2P file sharing networks started with the Napster phenomenon, reaching 20 million users who downloaded hundreds of millions of files before it was shut down in 2000 [18] due to copyright infringement [66]. Napster had a centralized directory with the list of shared files (music files) of every IP address (host) in the network. This centralized directory was updated constantly as hosts connected or disconnected from the network. Hosts connected to the network used the centralized directory to search for a file. The file search result contained the list of hosts from which the searched file could be directly downloaded. Therefore, the searched file was directly downloaded from the host sharing the file.
A decentralized system for sharing music files called Gnutella emerged in the year 2000 [118] to replace Napster. Gnutella offered a new distributed search paradigm that many popular P2P file sharing networks like Limewire [109] and Kazaa [55] use today. These networks are overlays constructed on top of the Internet and account for 20% of the total aggregated traffic of the Internet [56]. An overlay network is a logical network on top of another network (e.g., on top of the Internet).

The Internet is a resource in itself that must be conserved by maintaining minimum overhead traffic. Traffic consumes “bandwidth”, which is a finite resource. In addition, the Internet itself relies on electricity to operate its hosts, links, and gateways. Current electricity usage by the Internet in the U.S. has been estimated at 74 TWh/year or $5.9 billion dollars/year at 8 cents/kWh [34] (2% of the total electricity consumed by office and network equipment in the U.S.) [42]. The usage growth of the Internet in the U.S. between 2000 and 2005 with respect to the population is 113% [51]. Based on this usage growth, it can be expected that the number of hosts in the Internet will continue to grow. Therefore, methods to reduce the electricity consumption of the Internet will become more and more important as its electricity use increases with its usage growth. There are feasible methods to save Internet electricity usage as shown by Gunaratne et al in [22]. This dissertation addresses the conservation of both Internet bandwidth and electricity in the context of P2P file sharing networks. The focus of this dissertation looks at how to reduce overhead traffic in P2P file sharing networks and investigates how to enable power management in P2P hosts.

The goal of a P2P file sharing network is to allow users to simultaneously download and upload files directly from the host sharing it. The message used to search
for a file is called a Query. The response to a Query message is called a Queryhit message. The Queryhit message contains the data required to download a file, which is both the IP address and the file identification number of the host sharing the file that is being searched for. The amount of Query and Queryhit messages generated to find a file is the overhead traffic that must be reduced in a P2P file sharing network. If overhead is reduced then response time to find a file improves.

Figure 1.1 illustrates the flow of Query, Queryhit, and file download messages when a user finds and downloads a file. Files are downloaded via the hypertext transfer protocol (HTTP) using the GET command, thus including web server functionality in a P2P host.
A user at a host searching for a file will broadcast a Query message by forwarding it to neighbor hosts that it is directly connected to (hosts one hop away). All hosts that receive a Query message will forward the message to their neighbors. If a host has the file that is being queried for, it will respond with a Queryhit message. Multiple Queryhit messages are possible since a Query message can be satisfied by multiple hosts sharing the same file. A file download message is used to download the desired file from a particular host.

The P2P file sharing network of 12 hosts shown in Figure 1.1 generates a total of 20 messages to download the file shared by two hosts. The number of messages to download a file depends on the number of hosts that share the queried file and the topology of the network. There are 13 Query messages and 6 Queryhit messages forwarded between hosts through the network depicted in Figure 1.1. Thus, the total overhead to download one file is 19 messages for this trivial example. The overhead traffic is a function of the number of hosts searching for files and the number of hosts connected to the network.

Because of the increasing popularity of P2P file sharing networks, it has recently been estimated that there are more than nine million P2P hosts in the U.S. [80, 86]. As in the case of instant messaging, it is very likely that in the near future PCs running P2P applications will stay powered on 24/7 because users will include the P2P applications in their startup set of applications to share files, personal photos, and even videos. Reducing the amount of overhead traffic and enabling power management (i.e., enabling P2P hosts to save electricity by powering down) would allow P2P file sharing networks to save bandwidth and electricity, which translates into saving money.
1.2 Motivation

Recent studies have shown that P2P file sharing overhead traffic is one of the largest fractions of traffic on the Internet [55, 109]. In [55], Karagiannis et al. determined that P2P file sharing overhead traffic is continuing to grow and that P2P traffic and available bandwidth will increase as a function of time. If the absolute amount of P2P traffic and use increases, two things will occur respectively:

1. The response time of the Internet will increase as congestion from P2P file sharing networks will affect not only P2P applications but all applications.
2. Energy use by the Internet will significantly increase and raise economic and environmental costs.

It is therefore critical to investigate how P2P file sharing networks can become more efficient while still maintaining their robustness (i.e., search performance does not degrade when P2P hosts are removed from the network). That is, reduce overhead traffic without compromising performance level in terms of being able to quickly find files. In the past three years, approximately one hundred papers have been published on the general topic of improving P2P file sharing network search method.

The impact of the possible energy savings of incorporating power management in P2P will grow as Internet usage increases [51, 86]. The number of hosts currently in P2P file sharing networks is more than nine million in the U.S. [86, 109]. The estimated energy savings of enabling power management for nine million P2P hosts such that 10% of the hosts can power down is 0.78 TWh/year or 63 million dollars/year. Thus, potential savings have considerable economic impact. This calculation is based on three assumptions. The first assumption is that all P2P hosts consume 100W when they are
powered on, the second assumption is that all P2P hosts are powered on for 24 hours
during the day, and the last assumption is that one P2P host will consume 876 kWh/year
or 70 dollars/year (assuming average retail price of electricity at 0.08 dollars/kWh from
Energy Information Administration in September 2006 [34]).

1.3 Contributions

The contributions of this dissertation are in reducing P2P overhead traffic and energy use
by P2P hosts. The specific contributions are:

1. The design and evaluation of an improved query search method called Targeted
   Search was achieved. The direct trade-off of the Targeted Search method is search
time versus overhead traffic. Targeted Search is applicable to existing P2P file
sharing networks such as Gnutella. Targeted Search was presented at the 2005
International Performance Computing and Communications Conference (IPCCC)
[84].

2. The design and evaluation of a novel P2P protocol called Broadcast Updates
   Local Look-up Search (BULLS) was completed. BULLS is an entirely new
search paradigm for P2P file sharing networks which uses local look-ups instead
of broadcast queries. BULLS was presented at the 2006 ACM Southeast Regional
Conference [85].

3. The first investigation on how to enable power management in P2P file sharing
   networks was conducted. Power management was viewed as a minimum set cover
problem. A new heuristic for the minimum set cover problem called Random Map
Out (RMO) was developed and evaluated experimentally. The complexity of
RMO is shown to be much less than the well known Greedy heuristics for the
minimum set cover problem. RMO has comparable minimum set cover size. Additionally, when compared to the Greedy heuristic, RMO can be more easily distributed among the hosts of a P2P network.

1.4 Organization of this Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 is a literature review of P2P file sharing networks, efficient search methods, and set cover algorithms. The Gnutella P2P file sharing network is described and general issues are explained. The energy use and characteristics of Gnutella P2P file sharing networks are covered. Finally, recent work in set cover algorithms is described.

- Chapter 3 is the design and evaluation of the new Targeted Search method that exploits the heavy tailed distribution of the number of shared files. The optimality of the method with respect to cost of finding a file is proven, and the performance evaluation is presented via numerical results. A Gnutella compatible implementation of the Targeted Search method called Ditella is described.

- Chapter 4 presents a new search paradigm based on Gnutella called BULLS. Flow models are developed for both BULLS and Gnutella. Measurements from actual distribution for the number of shared files are described. The performance evaluation is based on the amount of overhead traffic generated.

- Chapter 5 presents the application of BULLS to power manage a P2P file sharing network. The power management problem in P2P is defined as a set cover problem. Centralized and distributed algorithms are presented. The measurements used to evaluate the performance of the set cover algorithms are described. A new
algorithm called RMO is proposed, analyzed, and evaluated. The application of RMO to power management using BULLS is described. Experimental evaluation of P2P power management is presented, and the achievable energy savings are estimated.

- Chapter 6 summarizes the dissertation and provides future directions to further investigate the design of new P2P search paradigms that exploit user characteristics and energy efficiency in P2P file sharing networks.
Chapter 2:

Literature Review

This chapter covers P2P file sharing networks in terms of the protocols used, search methods in P2P networks, improved search methods that reduce the overhead traffic of finding a file, characteristics of the Gnutella file sharing network, energy management and set cover algorithms.

2.1 History and Overview of P2P File Sharing

A P2P network is a “computer network that uses the computing (e.g., CPU), storage (e.g., disk), and bandwidth resources (e.g., links) of the participant hosts” [46]. Each host provides and consumes the available resources, thus acquiring server and client functionality [66, 107]. P2P networks are “overlays built on top of an underlying network” [118] like the Internet [107].

The idea of P2P networks can be traced back to the birth of the Internet through the implementation of the Advanced Research Projects Agency Network (ARPANET) by the U.S. Department of Defense [66]. The main attributes of the ARPANET were resiliency to attacks (e.g., fault tolerance) and the sharing of unused resources [99]. These attributes were accomplished by the fact that hosts in the ARPANET connected to each other in a P2P fashion, that is, by directly accessing resources [66]. Today’s P2P networks preserve these properties and provide the user methods and mechanisms to locate resources from
the enormous amount available across the Internet [118]. In this context, a P2P file sharing network is a P2P network in which hosts have server and client functionality [84], upload or download audio and video files [39], and search for files via keywords contained in the file name [9, 119]. Hosts in P2P file sharing networks are called servents (i.e., have server and client functionality), but the term has not been widely disseminated [9].

2.1.1 P2P Predecessors

P2P file sharing networks can be thought as having two predecessors: Usenet and DNS (Domain Name System) [83].

Usenet, like P2P, allowed files to be copied between hosts without centralized control (directly accessing the host storing the file needed). Usenet is based on the Unix-to-Unix Copy Protocol (UUCP) [83]. A protocol defines the format and order of how messages are exchanged between hosts, actions taken when messages are transmitted, and actions taken when messages are received [66]. Therefore, UUCP allows Unix hosts to automatically connect, exchange files and disconnect from other Unix hosts. Usenet was initially used in 1980 to exchange messages, patches, emails, and files between the University of North Carolina and Duke University [83].

DNS is a solution to the mechanism used to share a single flat file called hosts.txt in the early days of the Internet [66]. This file was copied periodically through the Internet and contained the list of mappings between a domain name and the host IP address (e.g., www.usf.edu mapped with the IP address 131.247.2.1). DNS uses a hierarchy of name servers (hosts storing mappings between domain names and host IP addresses). The name servers used by DNS operate both as a server and a client [76].
No single name server has the complete list of all the mappings for all the hosts in the Internet. Instead, all mappings are distributed among all the name servers.

There are three levels of name servers: local name servers, root name servers, and authoritative name servers [66]. The DNS hierarchy of name servers is shown in Figure 2.1. The numbers inside the circles beside the arrows represent the order in which name servers are requested to resolve the domain name. When a host issues a DNS query (host request to resolve a domain name), the local name server looks-up the domain name. If the local name server cannot resolve the domain name, it will act as a client and forward the DNS query to the root name server. If the root name server can answer the DNS query, it will reply to the local name server that will then reply to the requesting host. If the root name server cannot answer the DNS query, it will forward the DNS query to the authoritative name server. When an authoritative name server receives a DNS query, it replies to the root name server that will then reply to the requesting host from which the DNS query originated.
DNS is similar to P2P file sharing networks because hosts both have server and client functionality. DNS name servers allow the content of the file `hosts.txt` to be distributed across the network. Unlike DNS distributing the content of a single file, today’s P2P file sharing networks are used to share millions of files [120] and are classified in two categories, structured and unstructured.

Structured P2P file sharing networks have a mechanism that controls the placement of files, that is, a host does not decide which shared files to store. Unstructured P2P file sharing networks have no centralized control, and the search method is based on a broadcast-based technique [46]. P2P network protocols (P2P protocols) define the format and order of how messages are exchanged between hosts and the actions taken when messages are transmitted or received on a P2P network [99]. Using these two categories (unstructured and structured P2P networks), a description of existing P2P file sharing networks and protocols used is presented below.

2.1.2 Structured P2P Networks

In structured P2P networks, the overlay network is tightly controlled, and files are deterministically placed at specific locations for query efficiency. Each host maintains a distributed routing table (DRT) that is used to route queries towards the host storing the file being queried [46]. The DRT provides the mapping between the file identifier (e.g., reference identifying a file) and the location where the file is being stored (e.g., the IP address of a host).

Disadvantages of structured P2P networks are that complex queries (e.g., query for a file by the keywords contained in the file) have yet to be efficiently implemented [46],
have not been widely deployed, and their ability to handle unreliable hosts has not been tested [6, 46, 69].

A brief summary of well-known and popular structured file sharing systems are described below. These P2P structured files networks were chosen because they have a significant traffic impact on the Internet. The main differences are due to the implementation of the DRT used and on the hosts that will store files.

2.1.2.1 Chord

Chord is a distributed look-up protocol that efficiently locates the host that stores a particular data item (e.g., a file) identified by a key (e.g., file name) in a P2P network [108]. All hosts and data items in Chord have an identifier with length $m$. The value for $m$ is sufficiently large enough to make the probability of two host identifiers hashing to the same identifier negligible (also holds for data item identifiers) [69]. The identifier for a host or data item is obtained using a hash function such as SHA-1 [69]. A host’s identifier is generated by hashing the host’s IP address. The data item identifier is the hash of its key [6]. The term host will be used in this dissertation to refer to both the host and the host identifier (hash of IP address). The term key will be used to refer to both the data item key and the image of hashing the key [108].

All hosts and keys are arranged in a Chord circle order using modulo $2^m$. Figure 2.2 shows a Chord circle with three host storing three keys. In Figure 2.2 the Chord circle has eight identifiers (0, 1, 2, 3, 4, 5, 6, and 7). Of the eight identifiers available only identifiers 0, 1, and 3 are actually used and have keys 6, 1, and 2 assigned respectively. Key numbered $k$ is assigned to the first host identified by a number equal to $k$ or to the
next host with the number greater than $k$. Queries for a given key are routed towards the host storing the key by each host maintaining a DRT denominated finger table.

The entries in the DRT are pointers to the successors of the hosts stored. Queries for a given key are passed around the Chord circle using the entries in the DRT until the host that is storing the key is reached. To increase the efficiency of locating keys, the DRT includes additional pointers to host successors allowing the time for key look-ups to be resolved in $O(\log N)$ for a P2P network of $N$ hosts. An example of an improvement over Chord, called Kademlia [6, 73], brings flexibility to the hosts. Hosts are allowed to select query routes based on latency or have parallel asynchronous queries [73].

2.1.2.2 Content Addressable Network

Content Addressable Network (CAN) applies the concept of hash tables to P2P file sharing networks, mapping a file name to a file location in a multi-dimensional Cartesian coordinate space [90]. The coordinate space is partitioned in zones. There are as many
zones as hosts so that each host can be assigned to a separate zone [69]. Figure 2.3 shows the zones assigned to five hosts (A, B, C, D, E) in a 2-dimensional coordinate space between [0, 1] \times [0, 1]. The key (identifier of the file) is mapped onto a point \( P \) in the coordinate space. The coordinates of \( P \) determine the zone it belongs to and the host that stores the key. Each host maintains a routing table of all the hosts that are neighbors (surrounding zones) [6]. Queries are forwarded along the path that approximates a straight line between the host that issued the request and the host storing the key. Upon receiving a query, a host forwards it to the neighbor closest to the host storing the key (ties are arbitrarily broken) [90].

When a host fails, joins or leaves the network, its zone must be reassigned to one of its neighbors. CAN uses a host-reassignment algorithm that merges and combines the zones assigned to hosts [69, 90].
2.1.2.3 Pastry and Tapestry

Pastry is a scalable distributed object location layer for large-scale P2P networks [96]. Examples of applications based on Pastry are SCRIBE (publish and subscribe system) [20] and PAST (global persistent storage utility) [33]. Pastry is a self-organizing decentralized overlay network. Each Pastry host is randomly assigned a 128-bit identifier (host identifier) when it joins the network [96]. The host identifier is the position occupied by the host in a circular 128-bit identifier space that is uniformly distributed [6]. A Pastry message (e.g., query for a file) has a numeric key associated (e.g., hash of the file name). Messages are routed towards the host that is numerically closest (e.g., least number of IP routing hops) to the key being searched for. Figure 2.4 shows an example of how the host identified as 37A0F1 locates the key B57B2D. In Figure 2.4, each Pastry host identifier has the matching prefix digits with the key in bold. Messages are forwarded to the host whose host identifier matches the prefix of the numeric key with at least one digit more than the prefix match between the current host identifier and the key.
numeric key. To support the routing procedure, a Pastry host must maintain routing state information.

A similar approach to Pastry is Tapestry, the overlay network used by OceanStore (scalable global persistent data store) [65]. Both Pastry and Tapestry routing procedures are based on matching prefixes with host identifiers [33, 65]. Incorporating the localization of objects (e.g., files) independently of their physical name and rapid adaptation from hosts entering and departing the network is described in [65]. Improvements and extensions of Pastry and Tapestry are ongoing areas of research as shown by the recent project called Chimera [113].

Another interesting approach to structured networks is SkipNet, a distributed generalization of Skip Lists (sorted link list in which certain hosts have pointers that skip over many list elements). The scalable overlay network that is constructed supports two locality properties: content locality and path locality. Content locality is the ability to explicitly place data on specific hosts or distribute the content within a given organization (users decide where to place content). Path locality is the ability to guarantee that the traffic is routed within the hosts of the same organization [45].

Other DRT-based structured P2P networks have been developed. Zhang et al in [119] present an improvement of the look-up latency in DRTs called look-up-parasitic random sampling (LPRS). DRT-based structured P2P networks have theoretical fundamentals that guarantee finding files if they exist [46] under the assumption that all hosts are reliable collaborators [6].
2.1.3 Unstructured P2P Networks

In an unstructured P2P network, hosts join and leave the network using loose rules, and files are stored independently in the overlay. Queries are sent across the overlay using a broadcast-based technique [6]. Broadcast is effective in locating popular files (e.g., highly replicated files) and resilient to hosts joining and leaving the network, but poorly suited in locating rare files (e.g., file without replicas) [6, 69]. The main characteristics of unstructured P2P networks are the decentralized nature of file placement and the uncertainty of a file query returning the location of a host storing a particular file [87]. The unstructured P2P networks described below differ in the search method used to locate files and are presented in chronological order.

2.1.3.1 Napster

In the year 1999, Napster was the first P2P network to demonstrate the potential of large scale P2P file sharing networks [51, 67, 68]. It had over 40 million hosts by early 2000 [83] but was shut down in 2001 due to copyright infringement [67, 68, 115].

There are two different types of hosts in Napster: a client and a server [107]. Clients are hosts that store the shared files in the network. The server is a host storing a central directory that is used to locate the client sharing a given file [118]. The centralized directory stored by the server is a table with the IP address, files shared, and bandwidth of each client [6, 46]. A client joins Napster by contacting the server and reporting its IP address, shared files, and bandwidth [46].

Figure 2.5 outlines the procedure to look-up and download a file in Napster. The requesting client in Figure 2.5 sends a file query to the server. The server then sends a reply to the requesting client with the list of clients from which the file can be directly
downloaded [69]. This approach of maintaining a central directory, however, has two main drawbacks. The first are load balancing issues (e.g., a single server can only handle a limited number of file queries) [46]. The second drawback is that there is a single point of failure, the server [83].

2.1.3.2 Gnutella

In 2000, Gnutella emerged as a decentralized P2P file sharing network overlay [70, 83]. A host joins the P2P network by randomly selecting hosts to connect to. The selection is made from a list of P2P file sharing hosts (e.g., list of IP addresses and port numbers) [111] that previously belonged to the network [6, 69] and are acquired from a cache (e.g., GWebCache) [62, 111]. Broadcasting a Query containing keywords of the file name is the method used to search for a file in the network. Once a response to a Query (called a Queryhit) has been received, the file is directly downloaded from the host storing it [46].

Figure 2.5 Three steps to locate and download a file in Napster [46]
When a host broadcasts a query, it forwards the query to every host one hop away (neighbors). All hosts receiving the query will continue to forward the query to their neighbors. Gnutella limits how queries are forwarded in two ways. The first is using a TTL (Time-To-Live) value to control the number of hops a query is forwarded. Each time a host receives a query it decrements the TTL value by one before it forwards it. A host receiving a query with a TTL value of zero will then drop the query, that is, not forward it. The second way Gnutella limits the forwarding of queries is by detecting duplicate queries (e.g., queries that have been previously forwarded due to loops in the network), and then not forwarding them on. Each host stores a query routing table with a unique identifier for each query already forwarded. Query duplicates are detected by a look-up in the query routing table. Queries whose unique identifier is found in the table are dropped.

Even with Gnutella’s two restrictions on broadcasting queries, considerable query processing overhead for each query is still generated. The load (e.g., number of queries processed and forwarded) at each host, that is query traffic, increases as query rate increments and additional hosts join the network [70]. File searches in Gnutella by broadcasting queries do not scale [54, 93]. Section 2.2 describes in detail the functionality and recent improvements of Gnutella. Section 2.3 is a review of scalable solutions for Gnutella.

2.1.3.3 Freenet

In 2000, another decentralized P2P network overlay similar to Gnutella called Freenet emerged. The main objective of Freenet is for hosts to freely donate part of their storage to anonymously store and retrieve content (e.g., files) [26]. It differs from other P2P
networks by maintaining the anonymity of the hosts that request and supply content. Each Freenet host has a local datastore (the storage made available to other hosts) and a dynamic routing table [69]. The routing table contains a mapping between the key of a file (e.g., hash of the file name) and the most provable location (the host) storing the file. When a host receives a request for a file that is not found in its local datastore, it forwards the query to the host with the nearest key match in its routing table (determined by lexicographic distance). A successful response from a host (i.e., the requested key is found in its local datastore) is routed back to the requesting host by the host that forwarded the request [46]. Freenet is scalable because hosts make decisions based on local information. It is only limited by the amount of memory that is available to store the routing table [26].

2.1.3.4 Fasttrack (Kazaa)

Fasttrack (Kazaa) was available in 2001 and is one of the most popular P2P file sharing networks today with more than one million users [81]. Fasttrack combines Napster’s idea of a centralized directory and Gnutella’s file search by broadcasting queries [67]. The overlay network is composed of hosts called supernodes. Hosts that have high bandwidth, storage, and processing power can volunteer to be supernodes [60]. Hosts that are not supernodes are called peers. When a host connects to the network, it can become a supernode or a peer depending on its capacity (e.g., bandwidth). A peer subscribes to an existing supernode and sends its shared file index to the supernode [46]. Searches are broadcast between supernodes only [6]. Each supernode contains the index of the shared files of each of the peers registered. If a supernode determines that one of its peers shares a queried file, it will answer the query on behalf of the peer from which the file can then
be directly downloaded [6, 46]. Although supernodes can efficiently search for files, Kazaa can exist without any supernodes. A network of only peers, however, would increase query latency [6] by loading peers with queries without considering the bandwidth, storage, or processing power of each peer [46].

2.1.3.5 BitTorrent

BitTorrent emerged in 2003 and is a P2P network used to distribute very large files such as video files [6, 88]. The goals of BitTorrent are to enable fast downloads of popular files while conserving bandwidth and discouraging free riders (hosts that do not share files) [88]. A recent study in [55, 109] determined that in June 2004, BitTorrent was responsible for 53% of the total P2P network traffic. The basic idea of BitTorrent is to divide a single large file (e.g., 100MB) into fixed-size pieces of 256 KB and then distribute the pieces among the hosts in the network [17]. The file is downloaded by establishing simultaneous TCP connection to each of the hosts storing a file piece [17]. For each file shared, BitTorrent stores a .torrent in a centralized host (e.g., www.supernova.org) [6]. A .torrent is a file that contains the name, length, hash value (e.g., SHA-1 of the file), and URL for the tracker of the shared file [88].

The tracker of a file is a program that keeps track of all of the hosts that store, upload or download a piece of a shared file. Figure 2.6 shows the process of downloading a file in BitTorrent. In Figure 2.6, the file, Choping.mp3, is divided into pieces called Piece 1 and Piece 2. The pieces and previous downloaded pieces are distributed across the network. In BitTorrent, a host that wants to download a file follows three steps shown in Figure 2.6 [88]. The first step is to locate the .torrent of the file to be downloaded (target file). Using the .torrent located in the first step, the host
connects to the tracker of the target file in the second step, which then returns list of hosts that have the pieces of the target file. The third step is for the host that wants to download the target file to select the hosts from which the file pieces can be downloaded (e.g., Host 2 and Host 3 in Figure 2.6) [17]. For a host to download a file, it must barter uploading and downloading the pieces of the targeted file. Figure 2.6 shows the bartering of file pieces between Host 1 and Host 3. Therefore Host 1 is downloading Piece 1 from Host 3 and uploading Piece 2 (previously downloaded from Host 2) to Host 3. This bartering of file pieces discourages free riders [6].

2.1.4 Ethical Issues in P2P Networks

P2P file sharing networks have revolutionized the way people share their music files but have also brought about a legal controversy between the Recording Industry Association of America (RIAA) and users. The path taken by many P2P file sharing networks to promote legal file sharing is for users to pay a small amount of money (e.g., $0.99) and
download songs from a supervised central site like iTunes. Laws that control legal file sharing risk controlling the Internet as well as intervening with technology evolution. The advent of P2P file sharing networks like BitTorrent, in which video files are shared, enhances the problem of enforcing copyright laws because of the expense involved in prosecuting a significant portion of the millions of Gnutella and BitTorrent users [81, 86]. Internet service providers (ISPs) may block P2P networks in the future to avoid being involved in copyright violation. Legal file sharing will continue to be an open problem, but that is not the focus of this dissertation.

2.2 Gnutella Details

P2P file sharing networks like Gnutella have caught the attention of researchers because decentralized P2P file sharing networks may very well reshape the development of network applications [27, 37]. The Gnutella protocol will be referred to as Gnutella throughout the rest of this dissertation, since it defines in detail the behavior of the Gnutella P2P file sharing network. Although Gnutella’s concepts are not new, its millions of users [86] certainly prove the feasibility of network applications adopting the technology. Curiously, Gnutella was built with the purpose of sharing recipes, not music files. Frankel and Pepper from Gnulsoft invented Gnutella in early March of 2000 [83]. They are best known for developing Winamp in 1997, the application used for playing digital music files [83]. A detail description of the Gnutella version 0.4 and the improvements reflected in version 0.6 is presented.

2.2.1 Gnutella Version 0.4

The Gnutella protocol version 0.4 is structured in three phases: connecting to the network, searching for files by broadcasting queries, and downloading a file. After
describing each phase briefly, a summary is presented using a finite state machine (FSM) representation of the behavior. For the remainder of this section, the Gnutella protocol version 0.4 will be referred to as Gnutella because it defines the core behavior.

In the first phase, a host joins Gnutella by obtaining a list of hosts (bootstrap list) from a bootstrapping host cache (bootstrapping host). The bootstrap list contains the IP address and port number of the hosts that have participated in the network (e.g., connected to the network). A host joins Gnutella by directly connecting to six random hosts from the bootstrap list [57, 70, 118]. The host connects to the six hosts via a permanent TCP/IP connection (one connection per host) [6]. Each host that accepts a connection is called a neighbor, and the set of all hosts connected is denominated neighborhood. When a host loses a neighbor, that neighbor is replaced by a host not belonging to the neighborhood selected from the cache [57]. Discovering new hosts in the network is done by requesting a host’s address and port number using a Ping message. Ping messages are broadcasted, thus pings are forwarded to all neighbors. If a host receiving a ping can accept additional neighbors it will answer the request with a Pong message containing its address and port number [6]. A Pong message is routed back by the host that forwarded the ping. All hosts maintain a routing table that registers the unique identifier of a ping as well as the identifier of the host from which the ping was forwarded from. If a host receives copies of the same ping from different neighbors, it will only register the first copy received in the routing table. All other copies are not registered and will not be forwarded. This assures that the pong is routed back to the host that initiated the ping through a unique path.
The second phase is initiated once the host has connected to Gnutella and has established its neighborhood. The host shares a collection of files that can be downloaded by other hosts. These files are stored in the shared file directory from which files are downloaded. A user at a host can search for a file by broadcasting a query containing file name keywords [6]. The host receiving the query matches the keywords contained in the query received with the keyword file names stored in the index of shared files. The index of files shared is the data structure that associates with each file a list of file name keywords used to answer the queries [117]. All P2P hosts have the capability to search by broadcasting queries [46].

The third phase of downloading a file occurs only after a queryhit is received for a query broadcast. The host that wants to download the file connects directly to the host storing the file via an HTTP GET. To download the file, the host needs the IP address, port number, and file identifier obtained from the queryhit [27, 37, 111].

The Gnutella protocol specification version 0.4 [111] defines the five types of messages used by Gnutella and how they are routed. Gnutella messages are summarized in Table 2.1.

The behavior of Gnutella version 0.4 is represented by the FSM shown in Figure 2.7. The FSM in Figure 2.7 was created to model the key aspects of Gnutella and is not part of the protocol specification version 0.4. The main functionality of Gnutella shown in Figure 2.7 can be summarized by two operations: file search by query broadcast and selection of the file to download from the multiple queryhit responses. The notation for the FSM diagram used in this dissertation shows states as vertical lines and transitions as horizontal arrows indicating the directions of the transition. Transitions are initiated when
the input or condition specified above the arrow is met. The output or actions are specified below the arrow and occur simultaneously while making the transition. The dotted arrows are the initial and final transitions of the diagram. The initial transition does not have an originating state, and final transitions do not have a destination state.

The FSM for Gnutella covers the behavior related to the exchange of messages (i.e., query and queryhits), which is the overhead traffic of the network. Low data rate operations, such as exchange of ping and pong messages, are omitted from the FSM. The final file download (one per queried file found) is the only traffic that is useful (i.e., is not overhead). In Figure 2.7, four states are defined: INITIALIZE, IDLE, SEARCH, and SELECT. The states and their transitions are described below.

<table>
<thead>
<tr>
<th>Messages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Contains the file name of the file being searched for. If a host shares the file, it will respond with a queryhit. A query may have more than one queryhit response.</td>
</tr>
<tr>
<td>Queryhit</td>
<td>Response to a query. It provides IP address, port number, and result set (files matching search criteria). A file from the result set is then selected and downloaded.</td>
</tr>
<tr>
<td>Ping</td>
<td>Used to discover Gnutella hosts. If host accepts connection, it will send a pong. A ping may have many pong responses.</td>
</tr>
<tr>
<td>Pong</td>
<td>Response to a ping. Contains the IP address and the amount of bytes shared by the host accepting a connection.</td>
</tr>
<tr>
<td>Push</td>
<td>Allows firewall host to be incorporated in the network.</td>
</tr>
</tbody>
</table>
• **INITIALIZE**: A node enters this state by requesting neighbor addresses from a specialized bootstrapping node. Upon receiving a response from the bootstrapping node, it establishes a permanent TCP/IP connection using the neighbor addresses it receives and then transitions to **IDLE**.

• **IDLE**: In this state, a node can do one of the following four actions: 1) initiate a file search by sending a query and transition to **SEARCH**; 2) receive a query for
which it has a file, repeat the query to all of its neighbors, and respond with a queryhit; 3) receive a query for a file it does not have and repeat the query message; or 4) quietly depart the network.

- **SEARCH**: In this state, the node waits to receive query responses (queryhits). It will either transition to **IDLE** if no responses are received, or transition to **SELECT** if one or more responses are received. The time spent waiting for responses before transitioning to **SELECT** or **IDLE** is not considered in this paper because it does not impact the overhead traffic amount.

- **SELECT**: In this state, a node from which to download a file is selected. The user manually chooses the node from the set of nodes that responded with a queryhit. Once the node downloads the selected file, it updates the shared file list and transitions to **IDLE**.

There are two transitions that impact the amount of overhead traffic generated. The first is the transition from **IDLE** to **SEARCH** resulting in the broadcast of a query message to all nodes. The second is the transition from **IDLE** to **IDLE** that results from a file found (a queryhit is received). Queryhit messages are routed back the same path the query arrived using a queryhit routing table. Each node keeps a routing table with the query id and node id from which the query was received. Queryhit messages are routed back through the same path the query traveled from because they have the same id as the query they are responding to. This generates significant traffic for popular files (i.e., files that have many replicas).

Gnutella’s unstructured approach does not impose a centralized coordinator or a structure on how or where files should be stored. This approach makes Gnutella self-
organized and fault-tolerant in addition to the robustness provided by broadcasting queries. Gnutella’s large scale usefulness is limited by two factors. The first limiting factor is the percentage of hosts that are free riders (hosts that download files but do not share files). A user at a host in Gnutella should join the network with the intention of downloading and uploading files, but this is not always necessarily the case. The percentage of free riders in the last four years has decreased from more than 50% in 2000 to about 13% in 2005 [3, 64, 97, 120]. A large number of free riders can create congestion (e.g., increase query traffic and delay downloading files) and then degrade the performance of the network. In [39], the authors show that networks like Gnutella modeled with a close queueing network can tolerate a significant number of free riders because hosts sharing files can process an increased number of queries and provide files to be downloaded. The second limiting factor is its search method – file search by broadcasting queries does not efficiently use bandwidth [54, 93]. The four most significant changes made to Gnutella version 0.4 focuses primarily on improving its search method and expanding searches (e.g., search by universal resource name).

2.2.2 Gnutella Version 0.6

The summary of the four changes that reduce the messages exchanged are introduced in the new version of Gnutella version 0.6. They are as follows:

The first change is with respect to ultrapeers and leaves. The new version 0.6 [62] introduces a hierarchical overlay categorizing hosts as either leaves or ultrapeers. Each host determines if it should become either an ultrapeer or a leaf. A host elects itself to be ultrapeer capable if it has sufficient bandwidth (bandwidth of at least 120 kbps for downloads, 80 kbps for uploads), an uptime (amount of time a host remains connected to
the network) of a few hours, and enough resources to store routing tables and process queries. Depending on the number of existing ultrapeers, an ultrapeer capable host becomes an ultrapeer. The number of existing ultrapeers in the network is estimated when new connections are established. A Gnutella host with version 0.4 is considered an ultrapeer to the hosts with version 0.6 but will not be using the query routing protocol (QRP) described below. A host is a leaf if is not ultrapeer capable. Leaves only connect to hosts who are ultrapeers. An ultrapeer acts as a proxy to the leaves it is connected to and decides which queries to forward to its leaves using QRP.

The second change is with respect to QRP. The ultrapeers using QRP forward queries to their leaves if it determines that a leaf can answer the query, that is, the query is matched with a file stored by the leaf. The rules established in QRP that allow ultrapeers to filter queries for their leaves are:

- Each leaf using a hash function stores all the hashed keywords of its shared files in a hash table. A keyword is a word contained in the file name of a shared file and whose length is greater than three.

- When a leaf connects to an ultrapeer, it forwards the hash table so that the ultrapeer can filter the queries the leaf receives. A leaf failing to forward its hash table to an ultrapeer will not be disconnected. The ultrapeer will not filter queries for the leaf, it will forward all queries to the leaf.

- An ultrapeer that receives a query performs a look-up in the hash tables of its leaves. Upon a successful match the ultrapeer will only forward the query to the leaves matching the query. If the look-up does not generate a successful match, it
will forward the query to the ultrapeers it is connected to and also to the leaves who failed to send their hash table.

- The hash table updates of a leaf are periodically sent to all the ultrapeers the leaf is connected to.

The third change is caching pongs. To reduce the bandwidth consumption from pings and pongs, ultrapeers respond to incoming pings on behalf of their leaves. Ultrapeers store pongs in a cache that is periodically refreshed. When an ultrapeer receives an incoming ping message, it responds with ten pong messages from its cache rather than forwarding the ping message. Caching ping and pong messages reduces traffic because fewer of messages are forwarded in the network.

The last change is for the support of queries. Gnutella version 0.6 supports file searches based on metadata (e.g., represented by XML), universal resource names (URNs), or a hash value from SHA-1 [6, 69]. To search for any of these search extensions, a prefix of the data (e.g., metadata, URN or hash value of SHA-1) is stored in query.

The main functionality of Gnutella version 0.6 can be summarized by two operations: 1) file search by query broadcast to ultrapeers; and 2) selection of the file to download from the multiple queryhits responses. For simplicity, it is assumed that all leaves send their hash table to the ultrapeer when they join the network. Otherwise, they behave as a host with Gnutella version 0.4. Figure 2.8 is the FSM representation of Gnutella version 0.6 created for an ultrapeer host. The FSM in Figure 2.8 is not included in the protocol specification version 0.6. The FSM in Figure 2.8 has the same four states as the FSM for Gnutella version 0.4: INITIALIZE, IDLE, SEARCH, and SELECT for
Figure 2.8 Gnutella version 0.6 ultrapeer FSM
The description of states SEARCH and SELECT remain the same as Gnutella version 0.4. The modified states and their transitions for the Gnutella version 0.6 ultrapeer FSM in Figure 2.8 are as follows:

- **INITIALIZE**: An ultrapeer host enters this state by requesting neighbor addresses from a specialized bootstrapping host. On receiving a response from the bootstrapping host, it establishes a permanent TCP/IP connection using the neighbor addresses of other ultrapeers or leaves that it receives, and then transitions to IDLE.

- **IDLE**: In this state, a host can either: 1) initiate a file search by sending a query message to connected ultrapeers only and transition to SEARCH; 2) receive a query for which it has a file, repeat the query to all of its ultrapeer neighbors, and respond with a queryhit; 3) receive a query for which a neighbor leaf has the file, repeat the query to all the leaves that have the file; 4) receive a query for a file it does not have and repeat the query message to its ultrapeer neighbors; or 5) quietly depart the network.

Figure 2.9 is the FSM representation of Gnutella version 0.6 for a leaf host. Four states are defined as Gnutella version 0.4; INITIALIZE, IDLE, SEARCH, and SELECT for the FSM. The description of states SEARCH and SELECT remain the same as Gnutella version 0.4. The modified states and their transitions for the Gnutella version 0.6 leaf FSM are as follows:

- **INITIALIZE**: A leaf host enters this state by requesting neighbor addresses of ultrapeers from a specialized bootstrapping host. On receiving a response from the
bootstrapping host, it establishes a permanent TCP/IP connection using the neighbor addresses of ultrapeers that it receives, and then transitions to IDLE.

- **IDLE**: In this state a host can either: 1) initiate a file search by sending a query message to the ultrapeers that it is directly connected to, and then transition to SEARCH; 2) receive a query for which it has the file and respond with a queryhit; or 3) quietly depart the network.

The states SEARCH and SELECT remain the same as Gnutella version 0.4. There are two transitions from the Gnutella version 0.6 ultrapeer and leaf FSMs that impact the
amount of overhead traffic generated. As in Gnutella version 0.4, they are the transition
from IDLE to SEARCH resulting in the broadcast of a query message to ultrapeer hosts,
and the transition from IDLE to IDLE that results from a popular file found (i.e., the file
is shared by many ultrapeers). The changes included in Gnutella version 0.6 with respect
to searching for keywords in file names opens the possibility to improve keyword search
Gnutella based protocols by using an efficient data structure.

2.3 Improving Search in P2P Networks

Broadcasting queries limits the adoption and potential use of sharing files in large scale
P2P networks [13, 21, 90, 108]. P2P users have increased in the order of millions in the
recent year, thus search methods that reduce traffic are very important in P2P network
design [81, 86, 116]. There are three examples of this. The authors in [13] discussed the
feasibility of using P2P networks for an end-to-end storage network. In [89], the authors
described a solution for streaming video on a P2P network. Lastly, an on-line algorithm
was designed to overcome the variability of download rates that occur when a host from
which a file is downloaded leaves the P2P network [89].

Both unstructured and structured P2P scalable solutions seek to increase the
number of hosts (network size) without increasing traffic (i.e., query and queryhit traffic)
or increasing the delay of file searches. Many studies have been based on the premise that
as network size increase; broadcasting queries generates much traffic and increases delay.
The authors in [90, 108] argued that broadcasting queries is not an efficient method to
search for a file because each file query causes the network to be congested [90, 108]. In
P2P networks like Gnutella queries are broadcast creating congestion because the number
of queries in P2P networks increment with the size of the network [70, 97, 118]. Another
interesting argument is explained in [21], the authors conclude that the load at each host grows linearly with respect to the total number of queries, which itself grows with the number of hosts [21]. Another study in [99], determined that Gnutella creates congestion in real network scenarios because P2P networks have numerous loops that generate duplicated query messages causing query traffic [90, 99].

The limitations of broadcasting queries have led to the design of many P2P protocols like CAN [90] and Chord [108]. It has also led to the improvements made to Gnutella through version 0.6 described earlier in section 2.1.2. Two studies have focused on P2P protocols that combine the ideas of structured and unstructured P2P networks. The first is the study in [90] improves broadcasting queries by designing and implementing a protocol and uses a distributed software layer called Saxon. Saxon builds an overlay with qualities like low overlay latency, high overlay bandwidth and low hop-count distance. Gnutella can naturally operate with Saxon due to its unstructured and decentralized properties [90]. In the second study in [37], the authors designed and evaluated YAPPERS; a look-up service in P2P networks that combines the advantages of Gnutella and Distributed hash tables (DHT) [37]. A small DHT is constructed and maintained by each host in the network. The DHT contains the information of the nearby hosts. The search mechanism intelligently uses this information to traverse the different DHTs stored by each host. The algorithm supporting the search divides the P2P network overlay into small overlapping neighborhoods (collection of P2P hosts). The content stored by each neighborhood is controlled and partitioned among the hosts. The look-ups are first directed towards the hosts in the neighborhood. If the look-up is unsuccessful, then it will intelligently forward the query to nearby neighborhoods or to the complete
network if necessary. YAPPERS simulation over a snapshot of Gnutella showed that it reduced the number of hops (look-ups) by one order of magnitude compared to Gnutella thus improving file search in unstructured P2P networks [37].

Other directions to improve P2P network search by broadcast are divided into three sections. The first section includes two studies that exploit power-law properties of P2P overlay network topology to reduce query and queryhit traffic. The second section describes two investigations that reduce the queries broadcasted by first directing queries to hosts that share the same subset files. The third section provides the most important improvements to in P2P file sharing networks.

### 2.3.1 Exploiting Power-Law Properties

It is known that the distribution of host degree (i.e., number of neighbors) of a P2P host exhibits a power-law [1, 35, 79]. A power-law is an expression that relates to quantities $x$ and $y$ by two constants $a$ and $k$ such that $y = ax^k$ [1, 35]. A linear fit of a power-law in a log-log plot must have a correlation coefficient of more than 96% [1, 35]. The Zipf distribution, Pareto distribution, and heavy-tailed distribution are commonly used as synonyms of *power-law distribution* because their cumulative distributions have a power-law form [35]. In the rest of the dissertation a power-law will be referred as a heavy-tailed distribution.

The investigation in [87] uses the heavy-tailed distribution of host degree to design and evaluate a new search technique based on Gossip at the application layer. The technique is called Deterministic Rumor Mongering (DRM) and exploits the heavy-tailed distribution of host degrees (i.e., number of connections each host has). Evaluation by simulation in a discrete time simulator in Java determined that DRMs cost is lower than
Gnutella by about 60%. (Cost is defined as the average number of messages per host generated from a single file query. Messages are forwarded by the hosts visited during the broadcast.) DRM results show that at a cost lower than Gnutella, it can still reach 96% of the hosts in the network, thus improving the search method used by Gnutella.

Another investigation is found in [30]; the authors in [30] describe how to exploit the heavy-tailed properties of P2P network topology. The work is based on the premise that heavy-tailed distributions have been used to model the traffic in communication networks and to describe traffic patterns over the Internet [30]. Intuitively, the size of the files transmitted across the Internet and the human computer interaction exhibit heavy-tailed distribution [1, 31]. Observations in [1] found simple heavy-tailed distribution for the Internet topology. Gnutella also exhibits similar heavy-tailed properties that can be exploited to reduce network search traffic. In [1], search is executed by random walks that intentionally select high degree hosts. This local search strategy scales sublinearly with the size of network (number of hosts). In a similar study found in [70], the authors define object popularity (i.e., query distribution) in P2P networks as a heavy-tailed distribution. The authors in [70] used a heavy-tailed distribution to evaluate the search method based on a random walk [70].

2.3.2 Reducing Query Traffic

Reducing the query traffic can be achieved by limiting the number of queries broadcasted in a P2P network. One way to reduce the number of queries broadcasted is by first directing queries to the hosts that share the same subset files or have a common interest. The principle of interest-based locality has been shown to reduce query traffic in two investigations. This principle asserts that if a P2P host shares a file-interest with another
host that also shares the same file, then it is likely that both hosts will have other files they both share. The study in [106] applies the principle of interest-based locality to create shortcuts. Hosts that share similar files create shortcuts to one another and use them to directly search for files. If a host does not successfully answer a query directly, the host will broadcast the query similarly to Gnutella. The maintenance and selection of the shortcuts is based on the success rate (ratio between the numbers of times the shortcut resulted in a successful search divided by the total number of times it was used to search for a file).

Evaluation is trace driven. Interest-based locality was demonstrated to exist in traces from the Kazaa and Gnutella P2P networks. Simulation results determined that the shortcuts resolved queries quickly while reducing the total load (queries processed) of the network. The evaluation of interest-based locality only uses the data from queries, and assumes that all queries always cause files to be successfully downloaded. Also, the evaluation does not include the shared files that the user added but were not downloaded from any host. This does not represent the behavior of a real P2P network [106].

An investigation of an improvement of the work in [106] is described in [9]. It extends the idea of creating an interest-based shortcut towards interest-based community. The community of a host is a graph on which the vertices of the graph are the hosts of the network and the edges represent the interest (i.e., shared files) between two hosts. An edge exists between two hosts if they share a common subset of files. The edge is weighted by the number of files both hosts have in common. Communities of hosts are created by locating the hosts that share common files. File search uses the knowledge of the shared file between hosts like in [106].
The performance evaluation of the algorithms implementing interest-based communities is based on the open source Aurora Freenet simulator in [114]. The performance metrics of the evaluation are the same as in [106]. The interest-based community approach achieves a reduction up to 21% for the average file search latency and up to 13% for the load of the network. When compared to the original idea of [106], it has 31% less average latency with no significant load increase.

Other approaches to reduce query traffic are presented in [52] and in [118]. The authors in [52] based their investigation on the fact that 70% of the query messages are redundant in unstructured P2P networks such as Gnutella. Broadcast is improved by the design of a search technique called FloodTrail, which reduces the redundant messages from a broadcast. This technique utilizes the trail information, which is defined as the collection of P2P links from which a query message from a broadcast reaches a new host (the host has received the query message for the first time). The links used to transmit redundant messages are eliminated from the trail, which provides an optimal multicast tree.

Queries are forwarded along these trails and a global unique ID is kept in every host that forwards the query for a certain period of time. This ID is used to recognize and discard redundant query messages. Trails are refreshed and additional links may be added to overcome the partial damage caused by hosts entering and leaving the network. The performance evaluation is done by trace-driven simulations. Results show that FloodTrail could reduce the traffic of redundant messages by up to 57% [52].

In [118] Yang et al. described the design and evaluation of three simple techniques to efficiently search in P2P file sharing networks like Gnutella. The techniques are called
Iterative Deepening, Directed BFS (breath first search) and Local Indices. Gnutella uses BFS with a depth limit $D$, equal to the time-to-live (TTL) value. Iterative deepening consists of multiple BFS file searches with each successive BFS file search having larger values for $D$. Searching stops when a query returns a queryhit or $D$ has reached the maximum value established. Directed BFS is similar to Iterative Deepening but instead of forwarding queries to all neighbors it selects a subset of neighbors to forward the query to. The neighbors selected are the hosts from which successful answers from queries have been received previously. Lastly, in Local Indices, each host maintains an index of the data stored within $r$ (radius) hops of itself. When a host receives a query, it can answer the query on behalf of every host within $r$-hops away. Maintenance of the indices is required each time a host enters the network, departs or updates its data. Each technique is evaluated by numerical analysis. Results show that the techniques reduce the aggregated bandwidth and the number of queries processed by the network. [118].

Other techniques that improve the broadcast search method in P2P file sharing networks includes the use of machine learning techniques and probabilistic methods.

### 2.3.3 Improvements in P2P File Sharing Networks

In this section two investigations that improve the broadcasting queries method used by P2P file sharing networks are described. The first investigation incorporates machine learning in P2P file search and the other describes a probabilistic file searching method.

The study in [15] describes a machine learning methodology which selects a set of good neighbors, that is, hosts from which files have been downloaded. Decision tree learning and the Markov decision process are used to derive the policy for selecting an adequate set of good neighbors. Using dynamic programming algorithms, the Markov
decision process is solved. Preliminary experiments of popular file searches resulted in quickly finding hosts with high bandwidth connections from which files can be downloaded [15].

A similar study in [112] describes a new adaptive and bandwidth efficient algorithm based on Gnutella called Adaptive Probabilistic Search (APS). APS search scheme uses the random walk search method described in [70]. Random walks limit flooding by selecting a random number of neighbors equal to \( k \) (\( k \)-walkers) to forward queries. It has been shown that APS reduces the overhead of queries by an order of magnitude but does not adapt query load at the hosts. APS uses \( k \)-walkers to discover files; instead of selecting the walker randomly like in [70], APS does it probabilistically. Previous search results are used to route queries to the neighbors from which successful queries have previously been received. Evaluation of the algorithm by simulation shows that APS achieves low-bandwidth consumption and high success rates (finds files). Its performance is a tradeoff between the success rate and the overhead messages produced by the query of a file [112]. Similar work to APS can be found in [16], where random walkers and estimates of the popularity of the resource (files) are used to optimize searches.

In this section, improvements in the broadcast search method of P2P networks has been investigated and solutions have been summarized. The next section describes current efforts and studies to reduce the electricity consumed by the devices such as hosts, links and gateways that the Internet relies on to operate. The investigations described below justify the need for P2P hosts to be energy aware by enabling P2P hosts
to power down during idle periods, that is include power management capabilities in P2P hosts.

2.4 Energy Use of P2P Networks

P2P file sharing networks like Gnutella to have millions of users [86] and millions of simultaneous hosts connected to the Internet sharing files. It is very likely that there are many popular files (i.e., files users download the most) duplicated among hosts, and that hosts will remain always connected to the network as is currently the case with shared disks in desktop PCs.

In 2003, an investigation published in [42] described how energy consumption of the Internet could be reduced by eliminating wasted energy at the hubs, routers and switches. The authors in [42] provided three reasons why hubs, routers and switches have high energy cost. Firstly, the devices remain powered on during idle periods, thus consume energy 24/7. Secondly, unlike monitors, network equipment does not have different energy saving states when idle, and the Energy Star program does not provide explicit recommendations. Lastly, network design primary factors include maximizing throughput and minimizing delay, but rarely aim at minimizing electricity consumption [42].

New power management methods for desktop PCs connected to the Internet were investigated in 2004 by Christensen et al[22]. The authors in [22] argued that newly shipped desktops and operating systems do not need to be fully powered because they can easily adopt the solutions for the current wake-up and message response problems. The potential impact of managing power at desktops can equal $80 million dollars if 1 TWh/year could be saved at 8 cents/kWh. Regarding P2P, measurements taken at the
University of South Florida (USF) undergraduate dormitories during March 2003 suggest that small time-scale power management is possible [22]. Given the fact that P2P has tremendously increased in use during the last year [86], this premise certainly holds for current P2P networks.

Current research to reduce the electricity consumption of the Internet includes networking devices and links (e.g., desktops, LAN switches with proxying, split TCP connections, and scaling link speed). A more recent investigation in 2005, focused on developing and evaluating methods to reduce the Internet’s electricity consumption [41]. The significance of the work in [41] is saving the wasted electricity of equipment fully powered on during idle times. In 2000, total energy use of office and network equipment was estimated to be 74 TWh per year; that is about 2% of the total electricity use in the USA [59, 95]. For this reason, powering down hosts gains significance as electricity consumption of office and network equipment like PCs grows. Particularly for P2P networks, the investigation in [41] addressed the need for Gnutella hosts to reduce their electricity consumption. There are two possible solutions that P2P networks could adopt. The first is proxying Gnutella on a NIC or within the first level LAN switch. The second is extending the concept of managing TCP connections to Gnutella. The authors in [41] proposed managing TCP connections under the client/server paradigm. The methods described in the paper, even if only modestly adopted, could result in savings of $2.7 billion per year in the USA [41].

The studies described before illustrate how methods to reduce the energy consumed by hosts could significantly save billions of dollars per year. Thus powering down hosts in P2P file sharing network can contribute to saving a lot of money in energy savings [22,
41, 42, 58, 59, 95]. To power down hosts in P2P file sharing networks during idle periods, however, requires the study of how files are distributed. The next section reviews the literature regarding file distributions in P2P networks.

2.5 Characterization of File Distribution in P2P Networks

Enabling hosts to power down in P2P file sharing networks requires shared files to be duplicated at hosts across the network. The set of shared files by each host in a P2P network dynamically changes over time [92, 120]. These changes are caused by users sharing new files from disks, deleting files from the set of files shared, or more commonly, from users sharing a downloaded file [85].

The file size distribution in a P2P network is a snapshot that captures the number of files shared by each host (cardinality of the set of files shared) and how the shared files are distributed among the hosts (i.e., which hosts share a given file) [24, 97]. Given that files are downloaded and content is replicated, the number of instances a file occurs defines the degree in which files are replicated in the network. A file with only one instance has no replicas; whereas a file with more than one instance has replicas. The number of replicas for a file is obtained by subtracting one from the number of instances of a file. The larger the number of replicas of a file there is, the more popular a file is considered [97, 120]. The file distribution characteristics include:

- Number of hosts in the network,
- Total number of files shared in the network,
- Maximum number of files shared by a host,
- Probability distribution of hosts sharing a number of files (file size distribution),
• Probability distribution of hosts sharing a file (file instance distribution or file replica distribution).

The characterization of the file size distribution is also used to define the set cover problem for P2P networks. This is described in detail in section 5 of this dissertation. Three studies that have characterized file size distributions in P2P are described below in ascending chronological order:

In 2001, Saroiu et al. [97] tried to characterize the population of hosts that participate in Gnutella and Napster. The Gnutella trace captured 1,239,487 Gnutella hosts, from which 1,180,205 had unique IP addresses. The trace covered eight days from May 6 until May 14 of 2001. The Gnutella file distribution measurement collected by the crawler only included the number of shared files by each host; no other measurement was presented. It was concluded that the distribution of the number of shared files by the hosts in the Gnutella trace followed is heavy-tailed because 7% of the hosts share more files than the rest of the hosts combined. The percentage of hosts that do not share any files (i.e., free riders) is about 25%, while 75% of the hosts share 100 files or less (including free riders), and only 7% of the hosts share more than 1000 files [97].

Another investigation in 2004 conducted in [24], described the availability of hosts (i.e., which hosts share files or have files available to download), the file popularity (i.e., what percentage of the shared files by a host are popular files or have been queried the most), and the locality of reference of downloaded files. The data was collected from Gnutella and Napster between February 24, 2002 and March 25, 2002. Data was collected from 20,000 Gnutella based P2P file sharing networks. BearShare [24] and SwapNut [24] were the only file sharing networks that allowed their shared files to be
probed. Results determined that about 10% of the most popular files accounted for 50% of the total number of files stored; hence it is a heavy-tailed distribution. Downloaded files are also heavy-tailed since 10% of the most popular downloaded files account for 60% of the files downloaded [24].

A recent study in 2006 characterized files in Gnutella [120]. In [120] the number of shared files collected was on average more than 2.5 million. The length of the period measured was divided into three periods of about two weeks each in the months of June, August and October. The crawler, called Cruiser, was used to capture the snapshots of the Gnutella version 0.6 [120]. Six main conclusions were drawn from this characterization as described below:

- Free riding is 13%. A decrease of 12% compared to 25% reported in [97].
- Both the number of files shared and the amount of storage contributed by each individual host follow heavy-tailed distributions.
- File popularity (query distribution) follows a Zipf distribution.
- The most popular file type shared is MP3 and accounts for two-thirds (1.6 million files) of the total number of shared files (2.5 million files). Video files, storage and file popularity tripled over the past few years.
- Files are randomly distributed in the network; and there is no strong correlation between the files shared by hosts that are one, two or three hops away.
- Shared files by hosts change slowly over time (timescale of days); and popular files experience variations in their popularity.
Characterization of file size distribution and file replica distribution affect when a P2P network host decides to power down. Determining which P2P network hosts can power down is a set cover problem.

2.6 Overview of Set Cover Algorithms

The minimum set cover problem, or the set cover problem for short, has been well studied, and its application remains an active area of research [12, 43, 44, 71, 82, 102, 105]. In 1972, R.M. Karp was the first to prove that the set cover problem is NP-complete [72]. Typical applications of set cover problem include: air-crew scheduling, the art gallery problem, and genome sequencing [12]. Recently, it has been applied within the computer network field to monitor link delays and failures [47]. The large applicability of the set cover problem within a variety of areas demonstrates how important and interesting it truly is.

The set cover problem is defined as the minimum number of sets (called a cover) from a collection of input sets. The union of the sets that belong to the cover must contain all the elements of the universe and the input sets are subsets of the universe [28, 38, 105]. The formal definition of the set cover problem is shown in Figure 2.10.

Figure 2.10 Set cover problem definition [28]

Given a finite collection of subsets $S = \{S_1, S_2, \ldots, S_j\}$ from a set $U$ such that $U = \bigcup_{i \leq k} S_i$, the set cover problem is to find the minimum number of subsets $C = \{C_1, C_2, \ldots, C_j\}$ of $S$ such that $j \leq i$ and $U = \bigcup_{i \leq k} C_k$. 

- Characterization of file size distribution and file replica distribution affect when a P2P network host decides to power down. Determining which P2P network hosts can power down is a set cover problem.
2.6.1 The Set Cover Problem as NP Complete

The set cover problem has been proven to be NP complete by a reduction from 3-SAT [38]. Set cover models many resource selection problems and is an abstraction of many common combinatorial problems. The idea behind the combinatorial complexity of the minimum set cover problem concept is shown in the example below. This example is a resource allocation problem with the set $U$ having the resources to allocate.

Suppose we have a set $U = \{1,2,3,4,5,6,7,8,9,10,11,12\}$ and a family of subsets $P$, $P = \{P_1,P_2,P_3,P_4,P_5,P_6\}$, from $U$ such that the cardinality of $P$ is 6 ($|P| = 6$). The set cover problem can be stated as the collection of sets, $C$, with the minimum cardinality, such that $C \subseteq P$. Figure 2.11 shows the Venn diagrams for the example described above for $U$ and set $P$. A brute force approach to finding the solution would be to list all the possible subsets of $U$. Let $P_i \subseteq U$ such that the cardinality of $P_i$ is $i$; that is, $|P_i| = i$.

The number of sets $P_i$ for a given cardinality is as follows:

- Number of possible sets $P_i$ with cardinality one is 4
- Number of possible sets $P_i$ with cardinality two is 6
- Number of possible sets $P_i$ with cardinality three is 4
- Number of possible sets $P_i$ with cardinality four is 1

In the example outlined in Figure 2.11, multiple solutions for the minimum set cover containing only two elements are possible. Two solutions are possible for the minimum cover set $C$, either the set $\{P_1,P_2\}$ or $\{P_1,P_6\}$.
Given finite sets $U$ and $P$ with large cardinalities (i.e., greater than 25), it will be very unlikely to find an optimal solution, or it will be very hard to find an algorithm that can solve the problem in polynomial time. However, a simple greedy heuristic can be used to design a Greedy approximation algorithm (Greedy algorithm) with a logarithmic approximation ratio. Thus, the Greedy solution is within a logarithmic factor of the optimal solution with $n$ being the number of elements of the universe [102].

### 2.6.2 The Greedy Algorithm

The Greedy algorithm was first proposed by Chvátal in 1979 [38, 102] and was shown to have an upper bound that is exactly $\ln(n) - \ln(\ln(n)) + \Theta(1)$ [102]. Since then, no other algorithm has been able to significantly improve the logarithmic approximation ratio of the Greedy algorithm; that is, as the size of the input increases the size of the approximate solution grows logarithmically relative to the size of the optimal solution [28, 38, 105].
The Greedy algorithm is shown in Figure 2.12. It has as input a collection of sets \((S)\) and the universe \((U)\). The output of the algorithm is a set cover \((C)\). Initially, \(U\) contains all the elements of the universe and \(C\) is empty, this is shown in the first two lines of Figure 2.12 respectively.

The set with the most uncovered elements \((Z)\) becomes a member of \(C\) (where ties are arbitrarily broken) after each iteration. After adding \(Z\) to \(C\), the elements belonging to \(Z\) are removed from the set of uncovered elements \((X)\). Thus, \(Z\) and its subsets are never chosen in subsequent iterations. Because the Greedy set cover algorithm minimizes the number of uncovered elements, the number of possible sets that can be chosen for the set cover is also reduced.

A more detailed description to understand the procedure that determines the set that belongs to the cover is shown in Figure 2.13. The set \(Z\) stores for each iteration of the while loop, the set which contains the most uncovered elements. \(Z\) is added to the collection of sets in the set cover and removed from the collection of sets that will be analyzed in the next iterations. The for each \(V \in Y\) loop selects the set \(Z\) from all
possible sets of \( Y \) (set not in \( C \) containing uncovered elements). The set \( V \) used by the for each loop, is a temporary set used to iterate over all sets in \( Y \).

Enhancements to the Greedy algorithm have been intensively researched by defining the set cover problem as an optimization resource problem [19, 40, 47]. Most recently, solutions have been sought by local improvements [71], randomized rounding [105], genetic algorithms [53], and mean field annealing [82]. These techniques and others may contribute to the significance of applying the set cover problem in new fields for innovative practical solutions.

```plaintext
// GreedySetCover(U, S)
X ← U
Y ← S
C ← ∅
while X ≠ ∅
    Z ← ∅
    for each \( V \in Y \)
        if \(|V ∩ X| > |Z ∩ X|\)
            Z ← V
    Y ← Y − \{Z\}
    X ← X − Z
    C ← C ∪ \{Z\}
return C
```

Figure 2.13 Detailed description of Greedy algorithm
Chapter 3:
Exploiting Known File Distributions - Targeted Search

In this chapter, a new Gnutella-compatible method to search for a file is designed and evaluated. The hypothesis is that hosts can “learn” the distribution of files (objects) and use this to improve file searches. This new method, called Targeted Search uses statistics from previous file searches to first search the hosts with the most files (most probable host to find a given file) and avoid broadcasting file searches to all hosts. Thus, the amount of query traffic can be reduced. Targeted Search is a provably optimal search method for P2P file sharing networks and exploits heavy-tailed distributions that are known to exist for file location distribution. In this chapter, the premise of heavy-tailed file distribution is investigated and shown to be the case for a trace in a P2P file sharing network. The Targeted Search method is described and its performance evaluated. Analytical models for search time and cost are developed and used in the performance evaluation. Lastly, the implementation of Targeted Search in a Gnutella-compatible prototype called Ditella is described.

3.1 Premise and Promise of Heavy-Tailed File Distributions

Characterization of P2P networks has shown that file sharing follows a heavy-tailed file distribution where few P2P hosts contain the majority of files shared (i.e., peaked) and many hosts, or free riders, may share no files at all [97]. These characterizations clearly
show that file distribution between hosts is not uniformly distributed.

Figure 3.1 depicts an example of a P2P network with multiple hosts that share objects (i.e., files). The number of files that a host shares follows a distribution, which is often peaked or may even be a heavy-tailed distribution [97]. A heavy-tailed distribution as defined in chapter 2 is as a power-law relationship. Figure 3.1 shows a P2P network with heavy-tailed file distribution. Also, in Figure 3.1, host (4) shares 15 files, host (2) 5 files, and host (3) 2 files, while hosts (1) and (5) do not share any files. Hosts (1) and (5) are thus free riders. Host (4) shares more files than all other hosts combined. This attribute of few hosts sharing the most files and most hosts sharing the least files is characteristic of a heavy-tailed distribution.

Figure 3.1 P2P network where shared files distribution is heavy-tailed
> **Step 1** Send a direct query iteratively to the hosts in the frequency list starting with the first listed host. This step terminates when $M_{top}$ hosts have been queried or the file has been found.

> **Step 2** If **Step 1** did not find the file then broadcast a query to all $M$ hosts.

> **Step 3** If **Step 1** or **Step 2** found the file in a host then download the file and update the frequency list.

**Figure 3.2 Targeted Search method**

Seeking to exploit the non-uniform distribution of files in hosts to improve searching is a provable hypothesis. The hypothesis is that individual hosts can “learn” the distribution of files in hosts as a function of their query distribution and use this knowledge to improve searching.

### 3.2 New Targeted Search Method

The Targeted Search method uses a frequency list to direct queries to hosts with a high probability of containing a file. The data structured used in Targeted Search is a list of tuplets $<host\_id, hit\_count>$ that is sorted in descending order by $hit\_count$. The value of $hit\_count$ is how many previous queries were successful (i.e., resulted in a file being found in this host). Thus, hosts are ranked in the frequency list by order of previous search success.

The Targeted Search method that executes in a host is shown in Figure 3.2. In **Step 1**, each query sent is followed by a time-out period during which time a response is waited for. A response indicates that the searched for file has been found and comes directly from the host that contains the file. In **Step 2**, a time-out is also used to wait for a response. The frequency list is updated at the end of **Step 3**. In the update, the tuplet
with the matching host_id has its number of hits incremented. If the host_id is not in the list, a new tuplet is created for the new host_id with hit_count set to one and is then added to the end of the frequency list. Sorting the frequency list is of low complexity. For a non-uniform distribution of file location, most sorting occurs in the $M_{top}$ entries and involves at most one change in location to update the list.

3.2.1 Proof of Optimality

In discussion with Allen Roginsky (personal communications, Spring 2004), the proof of optimality of the search method was developed. The goal is to determine the optimal search method for a given time constraint $k$. That is, the search must be completed within no more than $k$ steps. Each step can be comprised of sending a query (to one or more hosts) and waiting for a response. The response will indicate whether a queried host contains the searched for file. Suppose that there are $M$ hosts and that a single file is located in exactly one host. Furthermore, suppose that the probability of it being located in any given host is $1/M$. At each of $k$ steps, the hosts $s(1), s(2), \ldots, s(k)$ are checked (each host is checked only once) so that:

$$s(1) + s(2) + \cdots + s(k) = M.$$  (3.1)

The condition in equation (3.1) assures that all hosts are checked in the worst possible case; otherwise, the time constraint of $k$ would not be satisfied. The choice of the $s(i)$ is the strategy. The $s(i)$ can be any non-negative integers as long as equation (3.1) holds. The cost $C$ of finding our file is $s(1) + s(2) + \cdots + s(j)$, where $j$ is the step in which the file is discovered. Also, the cost associated to check host for a file is one. The cost $C$ is a
random variable. The objective is to find a strategy, that is, a set of $s(1), s(2), \ldots, s(j)$ that minimizes the expected value of $C$ denoted here.

Lemma 1:

The mean cost $E(C)$ of strategy $s(1), s(2), \ldots, s(j)$ is:

$$E(C) = M/2 + \left( s(1)^2 + s(2)^2 + \cdots + s(k)^2 \right)/2M.$$  \hfill (3.2)

Proof of Lemma 1:

If a file is found at step $j$, then the cost of finding it is $a(j) = s(1) + s(2) + \cdots + s(j)$. To find the probability of finding our file at step $j$:

$$\Pr[\text{the file is not found at steps 1, 2, \ldots, } j-1 \text{ and the file is found at step } j]$$

$$= \Pr[\text{the file is not found at steps 1, 2, \ldots, } j-1 \text{ and (the file is found at step } j \text{ given that the file is not found at steps 1, 2, \ldots, } j-1)]$$

$$= \left[ 1 - (s(1) + s(2) + \cdots + s(j-1))/M \right] \left[ s(j)/(s(j) + \cdots + s(k)) \right]$$

$$= \left( 1/M \right) \left[ s(j)/(s(j) + \cdots + s(k)) \right]$$

$$= s(j)/M.$$  

Therefore,

$$E(C) = \sum_{i=1}^{k} a(i) \Pr[C = a(i)] = \sum_{i=1}^{k} a(i) \frac{s(i)}{M} = \frac{1}{M} \sum_{j=1}^{k} \sum_{i=1}^{j} s(j) s(i). \hfill (3.3)$$

If $E(C)$ is doubled, $2E(C) = [2s(1)^2 + 2s(2)\langle s(1) + s(2) \rangle + 2s(3)\langle s(1) + s(2) + s(3) \rangle + \cdots + 2s(k)\langle s(1) + \cdots + s(k) \rangle]/M = \left[ s(1)^2 + s(2)^2 + \cdots + s(k)^2 + M^2 \right]/M$

and the statement of Lemma 1 now immediately follows from this.

End of proof of Lemma 1.
Lemma 2:

The strategy \( s(1), s(2), \ldots, s(j) \) where \( E(C) \) takes its minimum is such that:

A. If \( N \) is a multiple of \( k \), \( s(1) = s(2) = \ldots = s(k) = M/k \) and

\[
E(C) = M(k + 1)/2k.
\]

B. If \( N \) is not a multiple of \( k \), then set

\[
s(1) = r(2) = \ldots = s(k - m) = (N - m)/k \quad \text{and set}
\]

\[
s(k - m + 1) = s(k - m + 2) = \ldots = s(k) = (M - m)/(k + 1) \quad \text{where} \ m \text{ is defined as the smallest positive integer such that} \ m = M(mod \ k).
\]

Statement A is a special case of statement B. The optimal cost does not depend upon the order of the \( s(j) \).

Proof of Lemma 2:

Let \( s(1), s(2), \ldots, s(k) \) be the optimal strategy in terms of minimizing \( E(C) \) under the constraint that the search time does not exceed \( k \). Such an optimal strategy exists since there are a finite number of strategies. If more than one strategy leads to the same \( E(C) \), then choose any one of them. It will be shown that for any \( i, j \), it is true that:

\[
\text{abs}(s(i) - s(j)) \leq 1. \tag{3.4}
\]

Indeed, if equation (3.4) were not true, then there is some \( i, j \), such that \( s(i) - s(j) \) is greater than or equal to 2. Then another strategy can be found with \( s(i) - 1 \) instead of \( s(i) \) and \( s(j) + 1 \) instead of \( s(j) \), and the value of \( E(C) \) will be reduced by

\[
\left[ s(1)^2 + s(j)^2 - (s(i) - 1)^2 - (s(j) + 1)^2 \right] / 2M = [s(i) - s(j) - 1]/M > 0,
\]

so the initial
strategy was not optimal. Hence all \( s(i) \) are different by no more than one, and they also have to satisfy \( s(1) + s(2) + \cdots + s(k) = M \). This defines them uniquely.

To prove this and find them, suppose that \( k - m \) of the \( s(i) \) are equal to some number \( n \) and the remaining \( m \) of them are equal to \( n + 1 \). Then \( (k - m)m + m(n + 1) = M \), hence \( kn + m = M \), so \( n = M \mod k \) and \( n = (M - m)/k \). This gives us the values of the \( s(i) \)'s.

End of proof of Lemma 2.

If a file is found at step \( j \), then the cost of finding it is \( a(j) = s(1) + s(2) + \cdots + s(j) \).

For non-uniformly distributed files where the above lemmas hold, the search is now weighted as:

\[
E(C) = \sum_{i=1}^{k} a(i)P(C = a(i)) = \sum v(i)\left(p(v(i) - 1) + \cdots + p(v(i))\right) \tag{3.5}
\]

where:

\[
v(i) = \sum_{j=1}^{i} s(j). \tag{3.6}
\]

Here \( p(i) \) is the probability of finding the file in bucket \( i \), \( p(1) \geq p(2) \geq \cdots \geq p(M) \) and

\[
\sum_{i=1}^{N} p(i) = 1. \tag{3.7}
\]

In summary, an optimal strategy in terms of minimizing \( E(C) \) for a given time constraint \( k \) is possible. The Targeted Search method performance is evaluated in the following section and is based on the optimal search explained above. Targeted Search exploits a non-uniform distribution of files in hosts to achieve an efficient search in terms of search time and cost.
3.3 Performance Evaluation of Targeted Search Method

In this section, an analytical model of Targeted Search for a non-uniform distribution of files is developed. It is assumed that an efficient search returns the location of the searched file quickly at a low cost. Cost can be measured as the total host utilization per search. That is, the cost to find a file is equal to the number of hosts queried. Using the analytical model developed in the next section numerical results are generated to show mean time and cost to find a file. A simulation model is used to study performance for cases where the analytical model cannot be used. Figure 3.3 summarizes both independent and dependent model variables. The analysis of Targeted Search has the following three assumptions:

1. All files stored in the $M$ hosts are unique (i.e., there are no duplicated files between hosts).

---

**Independent variables:**

$M$ = Number of hosts all storing unique files  
$M_{top}$ = Number of hosts that are directly queried  
$\alpha$ = Number of times $\Pr$ [file in host 1] is greater than $\Pr$ [file in host >1] (peakness)

**Dependent variables:**

$E[time]$ = Mean time to find a file  
$C[cost]$ = Mean cost to find a file

---

**Figure 3.3 Variables for model of Targeted Search**
2. A requesting host will send queries directly (and one at a time) to up to $M_{top}$ number of hosts ($M_{top} \leq M$) and will then, if the file has not yet been found, broadcast a query to all $M$ hosts.

3. The requesting host can effectively send a direct query to a given host (targeted host). Thus there are no hops in the P2P overlay between the requesting host and the targeted host.

### 3.3.1 Analytical Models for Cost and Time

The first step in the evaluation of Targeted Search is to develop a simple distribution that models the file location distribution in a P2P network. That is, the number of files shared by hosts should be roughly a power-law. A simple distribution modeling the principal characteristic of a power-law will have one host sharing most of the files and the rest of the hosts sharing few files. This characteristic is achieved by varying file distribution from uniform in all hosts to peaked in one host. For $M$ hosts, a uniform distribution has

$$Pr[\text{file in host } i] = \frac{1}{M} \quad \text{for } i = 1, 2, \ldots, M.$$ 

The parameter $\alpha$ is introduced to adjust a uniform distribution so that $Pr[\text{file in host } i = 1]$ is $\alpha$ times greater than $Pr[\text{file in host } i > 1]$. That is,

$$Pr[\text{file in host } i] = \begin{cases} \frac{\alpha}{M + \alpha - 1} & i = 1 \\ \frac{1}{M + \alpha - 1} & i = 2, 3, \ldots, M \end{cases}$$

(3.8)

For $\alpha = 1$, equation (3.8) is a uniform distribution with the $Pr[\text{file in host } i] = \frac{1}{M}$ for $i = 1, 2, \ldots, M$. For a large $M$ and by increasing $\alpha$,
\( \text{Pr[ file in host 1]} \) approaches 1 while the \( \text{Pr[ } i > 1 \text{]} \) approaches 0. The sum of the probabilities in equation (3.8) is 1.

\[
\sum_{i=1}^{M} \text{Pr[ file in host } i \text{]} = \frac{\alpha}{\alpha + M - 1} + \sum_{i=2}^{M} \frac{1}{\alpha + M - 1} = \frac{\alpha}{\alpha + M - 1} + \frac{(M-1)}{\alpha + M - 1} = 1. \tag{3.9}
\]

The expected value and variance for the probability distribution of equation (3.8) can be found. Given random variable \( X \) (finding a file in host \( i \)), the \( E[X] = \sum_{i=1}^{M} i \text{Pr}[X = i] \) has the following closed form,

\[
E[X] = \left( \frac{M+1}{2(M+\alpha-1)} \right). \tag{3.10}
\]

The variance of \( X \) is defined by \( \text{Var}[X] = E[X^2] - E[X]^2 \) and the expected value of the square of \( X \) or \( E[X^2] \) is defined as \( E[X^2] = \sum_{i=1}^{M} i^2 \text{Pr[ file in host } i \text{]} \). The variance is

\[
\text{Var}[X] = \frac{M(2M^2 + 1)}{6(\alpha + M - 1)} - \left[ \frac{(M+1)(M-1)+2\alpha}{4(\alpha + M - 1)^2} \right]. \tag{3.11}
\]

The new probability distribution defined by (3.8) is called the “peaked distribution” in this dissertation. In the peaked distribution, \( \alpha \) determines the probability for \( X = 1 \) and \( X = j \ (j \in \{1\ldots M\}) \) such that \( \text{Pr[ } X = 1 \text{]} = \alpha \text{Pr[ } X = j \text{]} \). The value of \( \alpha \) is used to “tune” how many files are shared in one particular host (with no loss of generality, host 1) versus all the other hosts (host 2, host 3, …, host \( M \)). With a large \( \alpha \), the distribution of files is skewed, or roughly power-law in that one host shares most of the files and the other hosts share significantly fewer files.

The peaked distribution is useful as it can be tuned from uniform to highly skewed; characteristic of interest in P2P file location distribution. The peaked distribution is not
strictly a power-law because it does not have a linear trend when graphed on a log-log plot. Because a peaked distribution is skewed as in a P2P file location distribution, it is assumed that the frequency list in the Targeted Search in a requesting host will empirically match the P2P file request distribution. This will occur after “many” searches. In the next section, this assumption is explored and the evaluation of how many searches are needed for the frequency list to approach the actual file location distribution is discussed.

The analytical model for the evaluation of the Targeted Search method is defined for the mean time and cost to find a file. The time to find a file is the number of queries sent until the file is found. If the file is found in the first query, the time is 1. For direct queries, the maximum time is $M$ (i.e., all hosts are queried and the file is found in the last host). For a broadcast query, the time to find a file is 1. Because the cost to find a file is equal to the number of hosts queried, a broadcast query has a cost of $M$ (and time of 1). The tradeoff in the Targeted Search method is time versus cost. The value of $M_{top}$ can be used to control this tradeoff.

The mean time to find a file in Targeted Search or $E[\text{time}]$, requires that the probability to find a file in host $i$ be known for $i = 1, 2, \ldots, M$. Let $f(i)$ be defined as this probability such that $f(i) = \Pr[\text{file in host } i]$. Then, the expression for $E[\text{time}]$ has two terms. The first term is the time to find a given file in one of the $M_{top}$ hosts. The time to find a file in one of the $M_{top}$ hosts is the number of hosts queried. For example, if the file is found in the second $M_{top}$ host, then the time to find the file is 2 because it is necessary to directly query the first two $M_{top}$ hosts. The second term is sum of the time to directly
query all \( M_{\text{top}} \) hosts without finding the file \( \left( M_{\text{top}} \text{Pr[file in host} > M_{\text{top}}] \right) \) and the time to find the file in the \( M - M_{\text{top}} \) other hosts is \( \left( \text{Pr[file in host} > M_{\text{top}}] \right) \).

Thus, \( E[\text{time}] \) is:

\[
E[\text{time}] = \sum_{j=1}^{M_{\text{top}}} j \cdot f(j) + \left( M_{\text{top}} + 1 \right) \left( 1 - \sum_{j=1}^{M_{\text{top}}} f(j) \right).
\] (3.12)

In summary, the first term in equation (3.12) is the time for direct queries. The second term captures the probability of not finding the file in the \( M_{\text{top}} \) direct queries (and thus incurring an additional time of 1 to the already expended time of \( M_{\text{top}} \)). Using the definition of \( f(i) \) as in equations (3.8) and (3.12), \( E[\text{time}] \) simplifies to the closed form:

\[
E[\text{time}] = \frac{2M_{\text{top}}^2 - M_{\text{top}} + 2M + 2\alpha}{2(M + \alpha - 1)}.
\] (3.13)

The cost to find a file in Targeted Search or \( E[\text{cost}] \), follows the same reason as \( E[\text{time}] \) except for the cost of not finding a file in one of the \( M_{\text{top}} \) hosts. \( E[\text{cost}] \), like \( E[\text{time}] \) has two terms. The first term is the mean cost associated to find a file in one of the \( M_{\text{top}} \) hosts. This first term for \( E[\text{cost}] \) is equal to the first term of \( E[\text{time}] \) because the cost and time associated with finding the file in one of the \( M_{\text{top}} \) hosts is the same because the number of hosts searched is the same as the number of direct queries. The second term for \( E[\text{cost}] \) is similar to the second term of \( E[\text{time}] \) with the difference that the time and cost to find a file in host \( j \) for some \( j \in \{M_{\text{top}} + 1, \ldots, M\} \) is not the same. The time is 1 while the cost for a broadcast query is \( M \) (since all \( M \) hosts are queried).
Thus, the mean cost to find a file is:

\[
E[\text{cost}] = \sum_{j=1}^{M_{\text{top}}} j \cdot f(j) + \left( M_{\text{top}} + \alpha \left( 1 - \sum_{j=1}^{M_{\text{top}}} f(j) \right) \right).
\]  

(3.14)

In summary, the first term for mean cost and time are the same. In the second term of equation (3.14), an additional cost of \( M \) is incurred to the already expended cost of \( M_{\text{top}} \) with the probability that the file was not found in the first \( M_{\text{top}} \) direct queries. Using the definition of \( f(i) \) as in equations (3.8) and (3.14), \( E[\text{cost}] \) simplifies to:

\[
E[\text{cost}] = \frac{2M^2 - M_{\text{top}}^2 + M_{\text{top}} + 2\alpha}{2(M + \alpha - 1)}.
\]  

(3.15)

Equations (3.13 and 3.15) were compared with the results from a simulation model and found to be the same. Using the expressions for \( E[\text{time}] \) and \( E[\text{cost}] \), the effect of \( M_{\text{top}} \) and \( \alpha \) (peakness) can be studied on the performance of Targeted Search compared to a full broadcast search. A broadcast search will always have time = 1, and cost = \( M \).
For $M = 10$, $\alpha$ increasing, and $M_{top} = 1, 2, \ldots, M$ Figure 3.4 shows $E[time]$, and Figure 3.5 shows $E[cost]$. It can be seen that as $M_{top}$ increases, the mean time increases while the mean cost decreases. As $\alpha$ increases, both the time and cost decrease, and for large $\alpha$ values (i.e., more than 1000) the file is always found in the first host queried and thus $E[time] = 1$ and $E[cost] = 1$ for all $M_{top} > 0$.

3.3.2 Selection of Parameter Values

To evaluate Targeted Search for a representative case, a trace file collected from a real Gnutella P2P network was used to empirically form a file distribution. The trace file used comes from an eight day trace collected by Saroiu and Gribble from the University of Washington [97]. The trace was collected from May 6 to May 14, 2001 and contained the number of files shared by each of the 82,281 Gnutella hosts. The total number of files shared was almost 85 million. The average number of files shared was 1,032. It was found that 93.5% of the hosts shared less than 1000 files, while the maximum number of
files shared were 33.5 million and the minimum was 0. It is not known how many files were unique (i.e., not duplicated between multiple hosts).

Figure 3.6 shows the rank versus number of files shared on a log-log plot. The linear fit shown on the graph shown as the dark black straight line indicates that file distribution in hosts is power-law. This can be concluded because a power-law has a linear trend when graphed in a log-log plot. For our performance evaluation, it is assumed that all files were unique. A trace of 987 Gnutella hosts was collected for 3 days in July 2004. Of the 331,096 files discovered available for sharing, 97.8% of them were unique or had different file names.

### 3.3.3 Numerical Results

Numerical results for Targeted Search for the Saroiu and Gribble trace data are shown in Figure 3.7. The solid line shows $E[\text{time}]$ and the dashed line shows $E[\text{cost}]$ for $M_{top} = 1, 2, \ldots, 100$. For a broadcast query, the time is 1 and cost is $M$. The results show
that Targeted Search in a real P2P network can significantly reduce search time and cost when compared to broadcast search.

For example, for $M_{top} = 2$, the search time is roughly doubled, but the cost is reduced by 63% (on average 29,769 hosts are queried, not the full 82,281 hosts). This can be seen in Figure 3.7 as a sharp drop in the $E[\text{cost}]$ as $M_{top}$ increases.

**3.3.4 Discussion of Results**

In random walk search [70], one or more “walking queries” are routed through the P2P network. A walking query randomly chooses hosts. Previously queried hosts in a given search are not re-queried. Thus, random walk is random sampling without replacement where there are $M_{walkers}$ samplers (walkers). The mean time and cost for a single random walker is $(M + 1)/2$ and is independent of the distribution of files.

As the number of walkers, $M_{walkers}$ ($1 \leq M_{walkers} \leq M$), is increased, the time decreases proportionally, but the cost remains the same. For $M_{walkers}$ much less than $M$,
the mean time and cost are $E[\text{time}] = \frac{(M+1)}{(2M_{\text{walkers}})}$ and $E[\text{cost}] = \frac{(M+1)}{2}$. For the trace data, random walk has significantly greater $E[\text{time}]$ for all values of $M_{\text{walkers}}$ than does Targeted Search for all values of $M_{\text{top}}$. $E[\text{cost}]$ is greater for random walk for most values of $M_{\text{top}}$.

The Targeted Search hosts build their frequency lists by learning from previous searches (see step 3 in the Targeted Search method in Figure 3.2). It was evaluated how fast the learned frequency list converges to the actual frequency list (the distribution of files by location). Using simulation on the Saroiu and Gribble trace data, 20, 40, and 80 files were uniformly randomly chosen to be searched for. The trace data was used to determine the probability to find a file in a host. For each file chosen the probability that it was located in the first host is calculated. The resulting cumulative probability of each set of files chosen to be searched for, is the learned frequency list. This is plotted in Figure 3.8 Targeted Search convergence for trace.
Figure 3.8, and shows the first 100 hosts (of 82,281 hosts in the trace data). The heavy line is the actual or trace data. It can be seen that after 40 to 80 hits, the learned frequency list is converging very closely to the actual frequency list. This is expected for the heavy-tailed distribution case where the first two hosts contain almost two thirds of all files shared. Figure 3.8 shows in a list the four probabilities for a file being found in the first host. Note how the probabilities converge quickly to the actual as the number of hits increase.

3.4 Implementation of Targeted Search

The Targeted Search method has been implemented in a P2P host software release named Ditella [32]. The name Ditella comes from the prefix di and suffix tella meaning that which is woven or the webbed communication between hosts.

The Ditella host prototype is compatible with Gnutella hosts that use the Gnutella protocol version 0.4. Ditella is written in C (and is about 800 lines of code) for the Microsoft Windows. Figure 3.9 shows the main functionality of the Ditella host prototype. The Ditella specification, source code, and executable can be found on the project web site [32].

3.4.1 Gnutella-Compatible P2P Host

Ditella directly queries hosts where a file has been previously found (i.e., using the Targeted Search frequency list) before broadcasting a query to all hosts. The connection to the P2P network is accomplished by a three-way handshake and a bootstrapping procedure. Once connected to the network, a Ditella host can issue and respond to messages in a Gnutella-like fashion. The directly queried hosts are selected using the
statistics of successful searches in a frequency list. The purpose of the prototype is to test the feasibility of directly asking a host for a file before flooding.

Before a file search is issued, the Ditella must first connect to the Gnutella network by asking the user for the IP address of a known Gnutella host. The host is then able to accept connections from other hosts, send pings, send and forward pongs, send a query, identify a queryhit, and download a file using HTTP. Each connection to a host is handled by a separate process so it can have parallel TCP connections open, and a

Figure 3.9 P2P network with Ditella host
separate process is created to accept user input so performance can be optimized for response time.

The implementation of Ditella maintains statistics on the hosts responding to pings as well as those responding with queryhits. The statistics are kept in two files: statistics.txt and the pong.txt. The statistics.txt file maintains the IP address, the port number, the file size and the file names received from queryhits. This file is used to create and update the frequency list for the Targeted Search method. The pong.txt file stores the responses to ping requests created or forwarded by the host. This file is used to verify connectivity to the network. The behavior of Ditella is shown in a FSM representation of in Figure 3.11. The two different colored areas (gray and white) show the difference between Gnutella and Ditella. The states INITIALIZE, and SELECT in Ditella are as in Gnutella. The white area highlighted in Figure 3.11 is the transitions common to Gnutella and Ditella. The gray area delimits three transitions which make Ditella different from Gnutella. These three transitions redefine the SEARCH state for Ditella, and one transition affects the IDLE state such that a file search causes a direct query message the $M_{top}$ hosts (one at a time). In the modified SEARCH state for Ditella, the host waits to receive a direct query response (queryhits). If it does not receive a direct query response it will transition to IDLE and repeat a query message to all of its neighbors, or if it does not receive a query response it will transition to IDLE or transition to SELECT if one or more responses are received from a direct query response or a Queryhit.
transitions that make Ditella different from Gnutella

Figure 3.10 Ditella FSM
Chapter 4:

Changing the Search Paradigm – BULLS

In this chapter, the hypothesis is that by broadcasting file updates and using local search, the amount of overhead traffic can be reduced over a broadcast query paradigm. That is, the query search paradigm of Gnutella is reversed to one having all hosts periodically broadcast changes in their list of files shared instead of broadcasting file queries. Each host then builds a table of host names and files shared which makes it possible for searches to be local to a host. The new protocol named Broadcast Updates with Local Look-up Search (BULLS), includes desirable properties such as reducing overhead traffic and enabling power management. Gnutella and BULLS are represented using finite state machines (FSM). Flow models for the FSMs are constructed to evaluate the overhead traffic generated in messages per second. To reduce the overhead traffic of BULLS a Bloom filter representation for the list of files shared by a host is proposed.

4.1 Broadcast Updates Local Look-Up Search – BULLS Protocol

Unstructured Peer-to-Peer networks such as Gnutella [70] distribute content (files) in a decentralized manner, are self-organized, and are robust. P2P file sharing network applications including Limewire, Kazaa, and BitTorrent comprise the majority of the Internet’s traffic [55, 109, 110]. Much of this P2P traffic is overhead from the flooding
of query messages and the associated queryhit response messages from searches for popular files.

Flooding is suitable for a wide range of applications that have not been explored by existing P2P networks. Many P2P networks focus on limiting query flooding and do not allow hosts to know what files are shared by other hosts. P2P file sharing networks like FastTrack (i.e., Kazaa) use the concept of supernodes to proxy search requests from other hosts called leaves to limit flooding [6]. Flooding excludes the leaves with low probability of responding to queries from file searches. Supernodes store the directory of the files shared by each of its assigned leaves. Although, supernodes know the files shared by its leaves, they do not know what files are shared by other supernodes. FastTrack cannot determine the entire set of files shared in the network. Like FastTrack users in a Gnutella file sharing P2P network search for files by broadcasting queries or flooding the network with queries.

A file search requires the user to know the entire name of the file searched for or a substring contained in the filename. Queries in Gnutella are thus substring searches over filenames. In Gnutella searching for a specific file is equivalent to one substring search. For multiple files with no common substrings in their filename, a query message for each file must be made. Valid searches in Gnutella have a substring with length greater than three and do not contain any wildcards. In addition to these search restrictions, a user at a host cannot determine which files are being shared in the network. That is, it is not possible to have knowledge of the entire set of files shared in the network. There are two reasons for this. The first is that a host lacks a method to make available to all other hosts its list of shared files. The second is that it is not possible to make a single query
message for all the shared files in the network. Thus, multiple queries are needed and a large number of hosts must be queried. If this is done, the overhead traffic in queries and queryhits is very high. If it were possible for all hosts to have the knowledge of the files shared by all other hosts, then new and significant capabilities could be implemented. Such novel capabilities include:

- Power management: Hosts sharing redundant content could be powered down and energy savings achieved.
- Ethical file sharing: Since hosts make explicit their files shared it is unlikely they would want to share “illegal” content.
- Affinity groups: Users can establish social connections based on the knowledge of the similar content shared by other users (e.g., based on common musical tastes).

BULLS enables all nodes in Gnutella P2P file sharing networks to acquire the knowledge of the files shared by all other nodes, thus enabling new capabilities in P2P hosts. BULLS differs from the existing P2P file sharing networks described in chapter 2 in two aspects 1) all hosts are knowledgeable of what others in the network share explicitly, and 2) most unstructured P2P file sharing networks use a query-broadcast paradigm to search for a file. BULLS reverses this paradigm and explores broadcasting file updates instead of queries. The next section describes in detail the BULLS protocol.

4.2 BULLS Protocol

BULLS is a P2P protocol offering the same functionality as Gnutella. All hosts connect to the network in Gnutella style. However, unlike a Gnutella host, BULLS host stores a global directory data structure that contains the information of the files shared by each host in the network. Once a host has established a permanent TCP/IP connection with its
neighbors, it floods the network with the complete listing of its shared files. The file shared listing is repeated by hosts via an update message. If there is one update message for each entry (filename) in the listing of shared files, the network is flooded as many times as files shared. Similarly, the network is flooded with an update message each time a file to be shared has been added or deleted by a user. Additionally, each time a host disconnects (departing host) a depart messages is broadcast. Depart messages inform all hosts that files cannot be downloaded from the departing host.

4.2.1 Description of BULLS

The main functionality of BULLS can be summarized by two operations, 1) a local look-up file search (no overhead traffic is generated in the network) and 2) broadcast of update messages. All hosts repeat the update messages received, cache the updates, and receive and repeat depart messages. These two operations depend on the global directory data structure used by BULLS. A detailed description of the global directory is presented first and is followed by the FSM representation for BULLS. There are two FSM representations for BULLS. The first FSM for BULLS is based on the Gnutella protocol version 0.4, and the second is based on Gnutella protocol version 0.6.

The global directory data structure is shown in Figure 4.1. The structure stored by each host in BULLS is a table and remains the same for each of the BULLS FSM. Each row in Figure 4.1 represents the data stored for a host in the network. The columns in Figure 4.1 represent the two basic types of data stored. The first column is the hostName, it is used to identify a host in the network (IP address or host identification number). The second column is the list of fileNames. This column stores the file share listing (set of filenames shared) in lexicographical order of the host in a given row. The storage
requirements for the global directory data structure are evaluated later in this paper and are shown to be reasonable, even for large P2P networks.

Figure 4.2 is the BULLS FSM based on Gnutella version 0.4. Four states (as in Gnutella version 0.4) are defined, INITIALIZE, IDLE, SEARCH, and SELECT. The global directory data structure shown in Figure 4.1 will be referred to as “data structure” in the FSM’s description and in the rest of the dissertation, since it is the only data structure used by BULLS. The states and transitions are:

- **INITIALIZE**: A host entering the network can be in this state by requesting to receive neighbor addresses and downloading the data structure from a specialized bootstrapping host. On the reception of a response with the requested neighbor addresses, the host connects to its neighbors, forwards its own shared file list (one update message per file shared) and transitions to IDLE.

- **IDLE**: In this state a host can 1) make a file search by a local look-up in the data structure and transition to SEARCH, 2) detect a change in the data structure, repeat via an update message the changes in data structure (one update per change) and remain in IDLE, 3) receive an update message, modify the data structure with the update received, store it in the cache, repeat it (send update...
message to all neighbors except the one from which the message was received from), and remain in IDLE, 4) receive a depart message, update data structure by modifying departing host’s row entry and repeat depart message, or 5) disconnect from the network by sending a depart message.
- SEARCH: In this state the host waits for results from a local look-up and it can
  1) transition to SELECT if local look-up is successful or 2) transition to IDLE if
  local look-up does not return results.
- SELECT: In this state a host from which to download a file is selected. The set
  of possible hosts to select from is returned by the successful local look-up
  executed in the SEARCH state. The host downloads the file, updates its shared
  files, updates its data structure, and transitions to IDLE.

The five transitions that impact the amount of overhead traffic generated are:

1. The transition from INITIALIZE to IDLE in which a broadcast message per file
   shared entry to all hosts is issued. Broadcast is done as in Gnutella.
2. The transition from IDLE that occurs from a change in the shared files, update
   message is broadcast.
3. The transition from IDLE that occurs when a depart message is received and then
   broadcast.
4. The transition from IDLE that occurs when an update message is received and
   broadcast.
5. The transition that allows hosts from the IDLE state to disconnect by the
   broadcast of a depart message.

Figure 4.3 is the BULLS FSM based on Gnutella version 0.6. The FSM shown in
Figure 4.3 only describes the behavior of an ultrapeer host. Ultraceep hosts, and not leaf
hosts, exchange overhead messages (i.e., query and queryhit messages). The behavior of
a leaf host in BULLS is the same as in Gnutella version 0.4 that is, only generating query
message overhead traffic. The queryhit message response from an ultrapeer to a query
Figure 4.3 BULLS FSM based on Gnutella version 0.6
message from one of its leaves is omitted from the FSM because is does not impact the overhead queryhit traffic.

The data structure used by BULLS is only stored by ultrapeer hosts. Each ultrapeer host stores in the data structure its own share file listing and the shared file listing of the leaf hosts connected to it. Four states (as in Gnutella version 0.6) for ultrapeers are defined, INITIALIZE, IDLE, SEARCH, and SELECT. The states and transitions are very similar to the FSM of Figure 4.2; they are:

- **INITIALIZE**: An ultrapeer host entering the network can be in this state by requesting to receive neighbor addresses of ultrapeers (neighbors) or leaves and downloading the data structure from a specialized bootstrapping host. On the reception of a response with the requested neighbor addresses, the ultrapeer host connects to its neighbors, forwards its own shared file list (one update message per file shared) and the share file listing of the leaves (one update message per file shared) to ultrapeer neighbors (neighbor host that are ultrapeers) only and transitions to IDLE.

- **IDLE**: In this state an ultrapeer host can 1) make a file search by a local look-up in the data structure and transition to SEARCH, 2) detect a change in the data structure, repeat to ultrapeer neighbors via an update message the changes in data structure (one update per change) and remain in IDLE, 3) receive an update message, modify the data structure with the update received, store it in the cache, repeat it (send update message to all ultrapeers neighbors except the one from which the message was received), and remain in IDLE, 4) receive a query message from a leaf host, repeat the query to all of its ultrapeer neighbors,
and remain in \textbf{IDLE}, 5) receive a depart message, update data structure by modifying departing host’s row entry and repeat depart message to ultrapeer neighbors, or 6) disconnect from the network by sending a depart message.

- \textbf{SEARCH}: In this state the ultrapeer host waits for results from a local look-up and it can 1) transition to \textbf{SELECT} if local look-up is successful or 2) transition to \textbf{IDLE} if local look-up does not return results.

- \textbf{SELECT}: In this state a host is selected from which to download a file (the host can be an ultrapeer or a leaf). The set of possible hosts to select from is returned by the successful local look-up executed in the \textbf{SEARCH} state. The ultrapeer host downloads the file, updates its shared files, updates its data structure, and transitions to \textbf{IDLE}.

The transitions that impact the amount of overhead traffic generated are the same five transitions that impact the overhead traffic in the FSM from Figure 4.2. The transitions that impact the amount of overhead traffic cause the broadcast of the shared file list and the broadcast of updates when the shared file list is modified, that is when a file is added, deleted or downloaded.

4.3 Flow Models for Gnutella and BULLS

The flow models developed in this section result in expressions that are statistics for the storage requirement of the data structure of BULLS \((S_{\text{bulls}})\) in bytes and the overhead traffic per node in messages per second for Gnutella \((X_{\text{gnutella}})\) and BULLS \((X_{\text{bulls}})\). Flow models are based upon each of the FSMs for Gnutella and BULLS described. All flow models are developed as a function of the ten independent variables shown in Figure 4.4. The variables are defined for both Gnutella and BULLS.
There are three assumptions to be considered:

1. The first assumption is that the number of hosts in the P2P network \( (N_{\text{hosts}}) \) remains constant. This makes the behavior of both protocols independent of the number of hosts or the order that the hosts connect to the network. Overhead traffic can be analyzed when the P2P network is in a stable state where the same number of hosts enter and depart.

---

### Figure 4.4 Model variables

Independent variables:

- \( D \) = Host degree
- \( M_{\text{files}} \) = Number of files shared per host
- \( P \) = Probability of a host having a given file
- \( N_{\text{filename}} \) = Number of bytes required to store a filename
- \( N_{\text{hops}} \) = Number of hops (hosts) a queryhit travels
- \( N_{\text{hostname}} \) = Number of bytes required to store a host name
- \( N_{\text{hosts}} \) = Number of hosts in the P2P network
- \( R_{\text{search}} \) = Rate of searches per host (messages/sec)
- \( R_{\text{update}} \) = Rate of file list updates per host (messages/sec)
- \( T_{\text{stay}} \) = Time a host stays in the P2P network (sec)

Dependent variables:

- \( S_{\text{bulls}} \) = Storage required per host for BULLS (bytes)
- \( X_{\text{bulls}} \) = BULLS overhead messages rate per host
- \( X_{\text{gnutella}} \) = Gnutella overhead message rate per host
2. The second assumption is that a single message from either BULLS or Gnutella is equivalent to sending one packet in the network. This allows the comparison of the overhead traffic to be based on the flow of messages and not on the specific characteristics of the links and hosts of the network.

3. The third assumption defines each search to be equivalent to one file search (searches are used to locate one file in the network). Multiple files searches can be modeled as multiple single file searches.

In the flow model for version 0.4 of Gnutella and BULLS the total number of files shared in the network is $M_{files}N_{hosts}$ as each host shares $M_{files}$ files. Thus, for version 0.6 in Gnutella and BULLS, at least $M_{files}$ files are shared by each ultrapeer host. In the case of version 0.6 of Gnutella and BULLS, $N_{hosts}$ is the total number of ultrapeer hosts in the network. For simplicity it is assumed that there is an equal number of leaves connected to each ultrapeer and that the total number of files shared by all leaves connected to an ultrapeer is $M_{files}$. The total number of files shared by all the leaves in the network is $M_{files}N_{hosts}$ as each group of leaf hosts connected to an ultrapeer shares the same the number of files as the ultrapeer itself. This is a reasonable assumption given that the ultrapeer capacity must at least be equal to the aggregated capacity of its leaves. The total number of files in the network is $2M_{files}N_{hosts}$. The variable $P$ is defined as the measure of popularity of a file (the percentage of hosts that have a requested file). If $P = 0$ a host (ultrapeer or leaf) does not have a file, otherwise when $P = 1$ then all hosts have the file. Thus, $P$ determines the number of queryhit responses for Gnutella. The host degree $D$ is the number of neighbors maintained by a host for the
flow model of version 0.4. For the flow model of version 0.6, for simplicity it is assumed that ultrapeers have degree $D$ and that leaves also have degree $D$.

The rate of file searches per host (ultrapeer or leaf), $R_{\text{search}}$, corresponds to the total file query search activity initiated by the user at a host (successful and unsuccessful searches). It is assumed that search activity is the same for a leaf and ultrapeer host. Search activity depends on the user and not the host capability. A successful file search response in Gnutella is a Queryhit. Each queryhit message is routed back via the hosts from which the query was received from. The number of hops (hosts) the queryhit message travels in the network is $N_{\text{hops}}$.

All successful file searches are assumed to result in a complete file download (causing an update to the shared file list of the host). In addition to downloads, it is possible for the shared file list of a host to be changed by users removing or adding files from sources other than the P2P network. The rate of shared file list addition and deletions is the rate of updates, $R_{\text{updates}}$ for the flow model version 0.4. In the case of flow model version 0.6 ultrapeers have the rate of updates as $R_{\text{updates}}$ but a leaf update rate is less than that of an ultrapeer. It is a requirement that ultrapeers have more bandwidth available than leaves. Thus, it is reasonable to assume that leaves have half or less the rate of updates of an ultrapeer, that is $0.5R_{\text{updates}}$. Another assumption is that a single message (or packet) is used to send a request to neighbor hosts.

The following five events describe the situations in which BULLS or Gnutella send a single message: 1) file search query (Gnutella), 2) queryhit response message (Gnutella), 3) file update message (BULLS), 4) host departing message (BULLS), and 5)
broadcasting the entire shared file list (BULLS) when a host connects to the network. In the case of version 0.4 for Gnutella and BULLS it is assumed that $M_{files}$ messages are required to broadcast the entire shared file list (i.e., each filename requires one message). For version 0.6 of Gnutella and BULLS $2M_{files}$ messages are required to broadcast the entire shared file list of an ultrapeer ($M_{files}$ files) and the entire shared file list of all the leaves ($M_{files}$ files) of the ultrapeer. This is an extreme assumption. The shared file list for versions 0.4 and 0.6 could be compressed and require far fewer than $M_{files}$ messages or fewer than $2M_{files}$ messages, respectively, for each case.

The traffic overhead for both Gnutella and BULLS is generated by the flooding of messages. Each host that receives a unique (not already received) message repeats the message to all of its neighbors, except the neighbor it received the message from. The actual number of times a host receives a given message is a function of the network topology and message forwarding delay. Figure 4.5 shows two cases where (a) each message sent by Host 1 is received only once by Host 2, and (b) where the message is received four times by Host 2. In this dissertation, we consider the worst case of each host receiving a flooded message $D$ times. In any case, this behavior will be the same between Gnutella and BULLS (both use the same rules to repeat messages and have the same network topologies), so relative comparisons are similar.

4.3.1 Flow Model for Gnutella

Using the FSMs of Gnutella from chapter 2 for versions 0.4 and 0.6 a flow model is developed for the storage requirement and overhead traffic.
4.3.1.1 Storage Requirement

A Gnutella host for version 0.4 or version 0.6 does not require any local storage other than storing the shared files, thus there is no storage requirement for Gnutella.

4.3.1.2 Overhead Traffic

The overhead message rate per host for Gnutella version 0.4 is

\[ X_{\text{gnutella}} \approx R_{\text{search}} D(N_{\text{hosts}} - 1) + R_{\text{search}} N_{\text{hops}} P(N_{\text{hosts}} - 1). \]  

The first term is the rate of query messages seen by each host. Each host receives \( D \) copies of each query message sent by every other host. The second term is an approximation of the rate of queryhit response message seen by each host. This is an estimate of the number of hosts that forward the queryhit message because we do not consider any existing topology. Queryhit messages are returned via the backward path a query was received, thus each queryhit message travels on average \( N_{\text{hops}} \) and thus is received by \( N_{\text{hosts}} \) hosts.

Figure 4.5 Duplicated messages caused by broadcasting

(a) (b)
In Gnutella version 0.6 only ultrapeers repeat messages, thus the leaf hosts only generate query message traffic for the ultrapeers to route. The variable $N_{\text{hosts}}$ is the number of ultrapeer hosts in the network and the rate of file searches per host, $R_{\text{search}}$, is the same for ultrapeer and leaf hosts. The overhead message rate per host for Gnutella version 0.6 is

$$X_{\text{gnutella}} \approx R_{\text{search}}DN_{\text{hosts}} + R_{\text{search}}N_{\text{hosts}}P(N_{\text{hosts}} - 1).$$

(4.2)

The first term is the rate of query messages seen by each ultrapeer host and initiated by an ultrapeer host. Each ultrapeer host receives $D$ copies of each query message sent by every other ultrapeer host. The second term is the rate of query messages seen by each ultrapeer host and initiated by a leaf host. Each ultrapeer receives $D$ copies of each query message. The third term is an approximation of the rate of queryhit response messages seen by each ultrapeer host. Queryhit messages are returned via the backward path a query message was received, thus each queryhit message travels on average $N_{\text{hops}}$ and thus is received by $N_{\text{hosts}}$ ultrapeer hosts.

4.3.2 Flow Model for BULLS

A BULLS flow model as for Gnutella is developed for the storage requirement and overhead traffic for versions 0.4 and 0.6.

4.3.2.1 Storage Requirement

In BULLS version 0.4 each host must store the data structure that contains all of the names of all files stored in the network by all hosts. The size of this data structure (in bytes) is

$$S_{\text{bulls}} = N_{\text{hosts}}(N_{\text{hostname}} + M_{\text{files}}N_{\text{filename}}).$$

(4.3)
The first term is the number of bytes required to store all the *hostnames*. The second term is the total number of bytes necessary to store the *filenames* of all the files shared by each host. For BULLS version 0.6 each host must store the data structure that contains all of the names of all files stored in the network by all hosts (ultrapeers and leaves). The total number of files shared by all the leaves in the network is $MN_{hosts}$. The size of this data structure (in bytes) is

$$S_{bulls} = N_{hosts} \left( N_{hostname} + 2M_{files}N_{filename} \right).$$

(4.4)

### 4.3.2.2 Overhead Traffic

BULLS versions 0.4 and 0.6 use the same rules to repeat messages although in version 0.4 all nodes repeat the messages while in version 0.6 only ultrapeers repeat messages. The overhead message rate per host for BULLS for version 0.4 is

$$X_{bulls} \approx R_{update} D \left( N_{hosts} - 1 \right) + D \left( N_{hosts} / T_{stay} \right) \left( M_{files} + 1 \right).$$

(4.5)

The first term is the rate of flooded directory update messages seen by each host as a result of hosts adding or deleting a shared file. When all searches are successful (i.e., a file is found) and files are not otherwise added or deleted to a host, $R_{update}$ will be the same as $R_{search}$. The second term is the rate of flooded update messages seen by each host as a result of hosts entering the network (flooding their entire directory listing of shared files to all hosts) and from depart messages from departing hosts (by the first assumption, the rate in which hosts enter and depart the network is the same). Clearly, the trade-off in overhead traffic between Gnutella and BULLS version 0.4 is a function of the hosts entering and departing the network ($N_{hosts} / T_{stay}$) and $M_{files}$ (traffic generated by flooding updates). Specifically, BULLS version 0.4 will have lower overhead than Gnutella.
version 0.4 when the values of \( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \) and \( M_{\text{files}} \) are low, that is, \( X_{\text{gnutella}} > X_{\text{bulls}} \).

Using equations (4.1) and (4.5) the inequality for \( X_{\text{gnutella}} > X_{\text{bulls}} \) is

\[
R_{\text{search}} N_{\text{hops}} P(N_{\text{hosts}} - 1) > D\left( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \right) M_{\text{files}} + 1. \tag{4.6}
\]

The overhead message rate per ultrapeer host for BULLS version 0.6 is

\[
X_{\text{bulls}} \equiv R_{\text{update}} D(N_{\text{hosts}} - 1) + 0.5R_{\text{update}} D(N_{\text{hosts}}) + D\left( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \right) (2M_{\text{files}} + 1), \tag{4.7}
\]

and can be simplified to

\[
X_{\text{bulls}} \equiv R_{\text{update}} D(1.5N_{\text{hosts}} - 1) + D\left( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \right) (2M_{\text{files}} + 1). \tag{4.8}
\]

The first term is the rate of flooded directory update messages seen by each ultrapeer host as a result of ultrapeer hosts adding or deleting a shared file. The second term is the rate of flooded directory update messages seen by each ultrapeer host as a result of leaf hosts adding or deleting a shared file. When all searches are successful (i.e., a file is found) and files are not otherwise added or deleted to a host (ultrapeer or leaf), \( R_{\text{updates}} \) will be the same as \( R_{\text{search}} \). The third term is the rate of flooded update messages seen by each ultrapeer host as a result of hosts entering the network (flooding their entire directory listing of shared files and share files list of its leaves to all ultrapeer hosts) and from depart messages from departing ultrapeer hosts (by the first assumption, the rate in which hosts enter and depart the network is the same). The trade-off in overhead traffic between Gnutella and BULLS version 0.6 as before is a function of the ultrapeer hosts entering and departing the network \( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \), and \( M_{\text{files}} \). Using equations (4.3) and (4.8), BULLS version 0.6 will have lower overhead than Gnutella version 0.6 \( (X_{\text{gnutella}} > X_{\text{bulls}}) \) when the values of \( \frac{N_{\text{hosts}}}{T_{\text{stay}}} \), and \( M_{\text{files}} \) are low, that is.
\[ 0.5R_{\text{search}}DN_{\text{hosts}} + R_{\text{search}}N_{\text{hops}}P(N_{\text{hosts}} - 1) > D(N_{\text{hosts}} / T_{\text{stay}}) (2M_{\text{files}} + 1) \]  

(4.9)

4.4 Performance Evaluation of BULLS

The performance evaluation is based on the flow models in section 4.3. The numerical values selected for the variables of both Gnutella and BULLS are described in the next section followed by the numerical results and result discussion. Results are first shown for the flow model for version 0.4 and then for version 0.6

4.4.1 Selection of Parameter Values

The models for the storage requirements of BULLS and the overhead traffic of BULLS and Gnutella need to be parameterized for a performance comparison of Gnutella and BULLS. Figure 4.6 shows the values (and range) for the independent variables. The fixed value for each of the variables is representative and the range for \( T_{\text{stay}} \) is reasonable to study the dynamics of hosts. The values for \( M_{\text{files}} \), \( P \), and \( D \) were selected from the literature, \( M_{\text{files}} \) from [97] and \( P \) , and \( D \) from [70].

The estimates for the other variables are:

- \( N_{\text{hosts}} \) is calculated from \( D \). Given that each host has \( D \) different neighbors and that the maximum number of hops a message travels is 7 hops based on the standard Gnutella time-to-live value of 7 [70], then \( N_{\text{hosts}} = (D - 1)^7 = 78125 \).
- \( N_{\text{hops}} \) is the average path length a message can travel. It can be estimated as half the maximum number of hops a message travels (i.e., 7 hops), so \( N_{\text{hops}} = 3.5 \).
$D = 6 \text{ hosts}$

$M_{\text{files}} = 100 \text{ files}$

$P = 0.00125$

$N_{\text{filename}} = 50 \text{ bytes}$

$N_{\text{hops}} = 3.5 \text{ hosts}$

$N_{\text{hostname}} = 16 \text{ bytes}$

$N_{\text{hosts}} = 78125 \text{ hosts}$

$R_{\text{search}} = 4.17 \times 10^{-3} \text{ messages/sec}$

$R_{\text{update}} = 3.21 \times 10^{-3} \text{ messages/sec}$

$T_{\text{stay}} = 12 \text{ hours to 7 days}$

**Figure 4.6 Numerical values for BULLS model variables**

- $R_{\text{search}}$ is the sum of the average time for a user to search (30 seconds), select the file to download (30 seconds) and download a file (3 minutes). This is an extreme case where a user does not consume (e.g., listen or view) a file before initiating another search and download.

- $R_{\text{update}}$ is the sum of the rate of downloads (successful searches) and the rate that a user adds or deletes a shared file. It has been estimated that 77% of the searches are successful [61]. The rate at which a user adds or deletes a shared file is approximated at one per every few hours [61], which is negligible with respect to the rate of downloads as shown in [61]. The rate of updates is then

$$R_{\text{update}} = 0.77 R_{\text{search}}.$$
• \( T_{\text{stay}} \) is estimated to be in the range of many hours to several days. This models P2P applications as pervasive and “always on” as is currently the case with shared disks in desktop PCs. The value of \( T_{\text{stay}} \) has significant effect on BULLS overhead (in the second term in eq. (3)).

• \( N_{\text{filename}} \) is 50 bytes. Filenames are not usually longer than 50 ASCII characters (1 byte per character).

• \( N_{\text{hostname}} \) is 16 bytes because the IP address of host is used as the host name.

4.4.2 Numerical Results – Representative Parameters

The numerical results for the representative values in Figure 4.6 with \( T_{\text{stay}} = 12 \) hours and equations 4.1, 4.3, and 4.5 are:

• \( X_{\text{gnutella}} = 1956 \) messages/second

• \( S_{\text{bulls}} = 3.92 \times 10^8 \) bytes

• \( X_{\text{bulls}} = 2600 \) messages/second

For equations 4.2, 4.4 and 4.7 are:

• \( X_{\text{gnutella}} = 3908 \) messages/second

• \( S_{\text{bulls}} = 7.83 \times 10^8 \) bytes

• \( X_{\text{bulls}} = 4437 \) messages/second

The data structure size is about 374 MB and 747 MB for version 0.4 and 0.6, respectively. Given that hard drives sizes are usually 100 GB or larger, the BULLS storage requirement can easily be satisfied for both versions. Given that storage costs decrease with time, it is probable that within a few years the amount of storage required
for both versions of BULLS will be entirely negligible with respect to the capacity of a commodity hard drive. The message rate corresponds to less than 200 Kb/sec and 350 Kb/sec for version 0.4 and 0.6 respectively, which is reasonable for broadband connections of several Mb/sec data rate. If $T_{\text{stay}} = 12$ hours then the BULLS overhead traffic rate is 33% greater than Gnutella 0.4 and 12% greater than Gnutella 0.6.

4.4.3 Numerical Results – Ranged Parameters

Figure 4.7 and 4.8 show the overhead traffic rate as a function of the rate of hosts entering (and leaving) the network for flow model version 0.4 and 0.6 respectively. The variables $M_{\text{files}}, P, D, N_{\text{hosts}}, R_{\text{search}},$ and $R_{\text{update}}$ are fixed and $T_{\text{stay}}$ is varied. Figure 4.7 demonstrates that the overhead traffic rate for BULLS decreases as $T_{\text{stay}}$ increases.

4.4.4 Discussion of Results

From the numerical results from Figure 4.7 and 4.8, the difference between BULLS and Gnutella Overhead traffic as a function of $T_{\text{stay}}$ can be studied.
Figure 4.7 shows that BULLS overhead traffic rate is higher than Gnutella when $T_{\text{stay}} < 30$ hours. However, for the current version of BULLS and Gnutella (version 0.6), Figure 4.8 shows a higher traffic rate when $T_{\text{stay}} < 15$ hours. As P2P becomes a pervasive Internet application users are likely to remain connected for longer periods of time. Thus, the cases for which BULLS overhead traffic rate is higher is not a representative case.

From Figure 4.7, it can be determined that when $T_{\text{stay}} > 30$ hours BULLS reduces Gnutella’s overhead traffic by a minimum of 0.6% and a maximum of 19% ($T_{\text{stay}} = 7$ days). Also from Figure 4.8, it can be determined that when $T_{\text{stay}} > 15$ hours, BULLS reduces Gnutella’s overhead traffic by a minimum of 0.4% and a maximum of 38%. It is possible to further reduce the BULLS rate of overhead traffic. The broadcast of updates can be reduced by batching update messages and/or compressing them instead of sending separately.
Chapter 5:
Power Management in P2P File Sharing Networks

In this chapter, new directions in power management to reduce overall energy consumption of P2P file sharing networks are investigated. The hypothesis is that hosts sharing files shared by other hosts are redundant. Therefore, redundant hosts can be powered down to save energy. The challenge is to identify those hosts that are redundant. A redundant host is any host sharing files that are fully shared by other hosts. Remaining non-redundant powered-on hosts maintain availability of all shared files. The problem of P2P power management is studied as an application of the well-known minimum set cover problem. In this chapter a new minimum set cover heuristic, called Random Map Out (RMO), is developed and evaluated.

5.1 Potential Savings from Power Management

P2P file sharing is growing in popularity. In 2006 it was estimated that nine million desktop PCs were P2P hosts [80]. To effectively share files in a P2P network a host must be fully powered-on 24 hours/day, 365 days/year. Thus, sharing files via a P2P network is an application that induces or increases energy use. Many P2P hosts share only very few files and/or share highly popular files that are also shared by many other P2P hosts. A host with redundant content does not need to be sharing its files; it can power down and save energy (and the overall file availability on the P2P network is not affected).
A host that has redundant content may power down after a certain period of user inactivity. User inactivity is usually defined as a lack of mouse or keyboard activity (i.e., an implicit determination that a host is not being actively used) [77]. An inactivity timer is used to determine when the sleep state should be entered. The inactivity timer is always running but is reset whenever activity is detected. The duration of the inactivity time can be set, and it is typically in the range of 15 to 30 minutes. When the inactivity timer expires, a P2P host can determine if it is a redundant host and thus decide if it should power down to a low-power sleep state. The capability of a host to determine if it is redundant and power down is the P2P power management investigated in this chapter. Not all hosts with redundant content should power down; some level of redundancy in file sharing is desirable for load balancing. Load balancing is beyond the scope of this chapter; however the methods developed in this chapter can support redundancy in a P2P network.
Figure 5.1 shows an example P2P network showing redundant hosts. The five hosts shown in Figure 5.1 are sharing files identified with letters. For example, host 4 is sharing the three files named \( u \), \( v \), and \( w \). The files shared by host 3 are shared by host 2, and the files shared by host 5 are shared by hosts 2 and 4. Thus, hosts 3 and 5 are redundant.

### 5.2 Set Cover Model for P2P Power Management

Set cover was defined in chapter 2 section 2.6 of this dissertation as, given a universe \( U \) and a collection \( S \) of subsets of \( U \), a set cover is the sub-collection \( C \) contained in \( S \) whose union is \( U \). Minimum set cover is the \( C \) with the fewest number of subsets. In this model:

- Subsets of \( S \) are the P2P hosts
- Shared files are the elements of \( U \), thus a file is an element of \( U \)

The minimum set cover, \( C \), is the subset of P2P hosts that must be powered-on so that at least one instance of each file (each element in the universe \( U \)) is shared or available.

### 5.3 Random Map Out (RMO) - A Distributed Set Cover Algorithm

The minimum set cover provides the set of hosts that can power down without affecting file availability. To achieve this, each host should have a power management capability. One possible approach is for each host to independently determine if it is a redundant host and if so then power down. Hosts that are not redundant have a unique file (that is, a file that is shared by no other host in the network). These non-redundant hosts cannot power down as file availability would be affected.

The input to the Greedy set cover algorithm requires that all hosts know the files shared by all other hosts. This capability is not implemented in current P2P file sharing networks [84]. A new P2P file sharing protocol that can satisfy this requirement is the
BULLS protocol described in chapter 4 of this dissertation. The BULLS protocol maintains in each P2P host a global directory data structure containing the list of files shared by every P2P host in the network. Therefore, the hosts (locations) sharing a given file are known to all other hosts. This global directory is somewhat similar to the central directory that Napster maintained [6]. With Napster, the central directory is stored in one host. In contrast, all hosts using the BULLS protocol locally store the global directory. Thus, the set cover problem can now be viewed as a distributed set cover problem where all hosts know the files shared by other hosts and all shared files are distributed over a network. With this scenario, many questions arise for the case of distributed set cover. For example, does the Greedy heuristic remain the best?

A new heuristic for set cover that can be distributed with a reduction in needed processing is the new Random Map Out (RMO) heuristic algorithm shown in Figure 5.2. The input to and output from the new RMO algorithm is the same as for the Greedy algorithm described in chapter 2 of this dissertation. The most important step of the RMO algorithm is to determine if a given set \( Z \) belongs to the set cover (\( C \)). A set \( Z \) does not belong to \( C \) if each element in \( Z \) is a member of at least one other set in \( C \). That is, a set \( Z \) belongs to \( C \) if at least one of its elements is not a member of any set beside itself in \( C \). In
if my elements are contained other sets in $C$
remove myself from set cover ($C$)

**Figure 5.3 Distributed RMO heuristic algorithm**

the RMO algorithm all sets are initially assigned to the set cover. Therefore, when the RMO algorithm determines that a set $Z$ does not belong to $C$, it will remove $Z$ from $C$. The sets in $C$ are a minimal cover of $U$ as the RMO algorithm determines for each set $Z$ of $S$, whether it belongs to $C$ or not.

For the case of distributed set cover, the RMO algorithm can be distributed as each host can use the global directory to independently determine if it alone should power down. That is each host can independently determine if its content is redundant. Figure 5.3 shows the distributed RMO heuristic where each host executing the algorithm determines if it is “in” (a member of) or “out” (not a member of) the set cover. Each host independently and randomly executes the distributed RMO heuristic. For example, each host can maintain an independent inactivity timer. Upon the timer expiring (inactivity time-out), the execution of the distributed RMO heuristic will be triggered. Because each host executes the distributed RMO heuristic independently of other hosts, it is possible that a given file could be lost (that is, not included in the set cover). This would occur only if all hosts sharing the same file remove themselves from the set cover at the same time. An analysis of the probability of file loss is described in Section 5.5.

Distributing the Greedy heuristic is possible but does not have any advantage in overall reduced processing time (i.e., speedup). In the RMO heuristic the loop can be “unrolled” (i.e., task in if command of Figure 5.3) and yields the distributed heuristic
shown in Figure 5.3. In the case of the Greedy heuristic, each host (set) needs to compare its results (i.e., number of uncovered elements contained by the set) against the results of all other hosts. If one host is selected as a central coordinator, the loop could be “unrolled”. However, this approach to unroll the loop is not feasible for a fully distributed P2P network.

5.4 Performance Evaluation of RMO

There are two performance metrics of interest for minimum set cover algorithms:

1. The size or cardinality of the resulting minimal set cover.

2. The computational complexity (i.e., the running time).

The set cover size corresponds to the number of hosts that should remain powered-on so file availability is not affected. Computational complexity corresponds to the number of element comparisons required to achieve a minimal set cover. The number of element comparisons for the creation of, or any updates to, the global data structure described in Section 5.3 is assumed to be “free” or disregarded. Previous work in chapter 4 with the BULLS protocol makes this assumption possible as the required data for the global data structure is in the global directory maintained in each host by BULLS.

For the Greedy and RMO algorithms we assume a global data structure that contains for each element the number of sets that the element belongs to. This can be implemented as an integer array. The index of the array corresponds to the position of an element in $U$, and the content of the array is the number of sets the element belongs to. Figure 5.4 shows this global data structure for the example P2P network shown in Figure 5.1. The worst case running time for determining if an element is in a given set $U$ is,

$$O(|U|)$$

(5.1)
where $|U|$ represents the cardinality of $U$. This would be the case where a set contains all elements of $U$. Therefore, the worst case running time for determining if an element is in a set for a collection of sets $S$ occurs when all sets of $S$ contain all elements in $U$. This case is rare or pathological for the set cover problem and would mean that any single set from $S$ is a minimum set cover. The worse case running time for the Greedy algorithm is

$$O(|U| |S| N_{\text{min}})$$  \hspace{1cm} (5.2)

with $N_{\text{min}}$ defined as the minimum between $|U|$ and $|S|$ [28]. For $|U| < |S| \Rightarrow N_{\text{min}} = |U|$ the worst case running time is

$$O(|U|^2).$$  \hspace{1cm} (5.3)

Otherwise, when $|U| \geq |S| \Rightarrow N_{\text{min}} = |S|$ the worst case running time is

$$O(|S|^2).$$  \hspace{1cm} (5.4)

The worse case running time in number of element comparisons for the RMO algorithm with the global data structure describe in Figure 5.4 is

$$O(|U|^2 |S|),$$  \hspace{1cm} (5.5)
and for the case of distributed set cover, the worse case running time is

\[ O(\vert U \vert^2) \]  

(5.6)

5.4.1 Selection of Parameter Values

The performance of Greedy versus RMO is determined by the size of the set cover and the number of comparisons of each algorithm. The experimental performance evaluation of the set cover algorithms is divided in two parts. The first part is the simulation of synthetic input data for set cover, and the second part (in section 5.5) is an estimate for the probability of losing a file. For the first part, the synthetic input data is represented by:

- \( N \) = Size or cardinality of \( U \)
- \( M \) = Number of subsets contained by the collection \( S \)
- \( K_j \) = Size or cardinality of subset \( j \) from \( S \) where \( j = 1, 2, \ldots, M \)

The random variable \( K \) has the following properties,

\[ \sum_{j=1}^{\vert S \vert} K_j \geq N \]  

(5.7)

and

\[ \forall j (K_j \leq N). \]  

(5.8)

Synthetic input data that is a collection of subsets, \( S \), can be generated with these three variables. A random assignment algorithm for a given \( U \) can be used to generate the sets in \( S \). The process to generate the subsets is divided in two separate steps:
set random number generator seed
for $i = 1$ to $M$
  $K_i =$ set size from peaked distribution

**Figure 5.5** Set size creation for $M$ sets

set random number generator seed
for $i = 1$ to $M$
  put $K_i =$ uniformly randomly chosen elements from $U$ into set $i$

**Figure 5.6** Element assignment for $M$ sets

1. Generate the size for each of the sets in $S$ so there will be $M$ sets created. In Figure 5.5 the detailed description of this step is shown. If the properties specified in equations (5.7) and (5.8) do not hold, this step is repeated with a new seed.

2. Assign elements to each of the $M$ sets generated in the previous step. The detailed description of this step is shown in Figure 5.6. This step is repeated with a new seed if not all elements are assigned to the $M$ sets.

In particular, generating the sets of $S$ requires the use of a peaked distribution similar to the one previously defined in chapter 3 of this dissertation. The peaked distribution used in Figure 5.5 is described in chapter 3 in equation (3.8) and defined as

$$
\Pr[X = i] = \begin{cases} 
\frac{\alpha}{M + \alpha - 1} & i = 1 \\
\frac{1}{M + \alpha - 1} & i = 2, 3, \ldots, M.
\end{cases}
$$

(5.9)

For $\alpha > 1$, equation (5.9) describes a peaked distribution of set sizes. That is, $\Pr[X = 1]$ is $\alpha$ times greater than the $\Pr[X > 1]$; or the size of set 1 is $\alpha$ times larger than the size of any
of the other sets \((i = 2, \ldots, M)\). Therefore, \(\Pr[X = i]\) is the set size probability for set \(i\).

For \(\alpha=1\), equation (5.9) describes a uniform distribution of set sizes, therefore all set sizes have the same probability or \(\Pr[X = i] = 1/M\) for \(i = 1, 2, \ldots, M\). Assuming that set 1 is of size \(\alpha x\) such that \(x\) is a positive integer; the size of the other sets is \(x\). Thus, the expected value of \(X\) is

\[
E[X] = \frac{\alpha x + (M - 1)x}{M}.
\]  

(5.10)

Given \(E[X]\), \(\alpha\), and \(M\), it is possible to solve for \(x\),

\[
x = \frac{E[X]M}{(M + \alpha - 1)}.
\]  

(5.11)

The peaked distribution is used to model the set sizes represented by random variable \(K\) if and only if equations (5.7) and (5.8) are both satisfied. That is,

\[
\alpha x + (N_{sets} - 1)x \geq N
\]  

(5.12)

and

\[
\alpha x \leq N.
\]  

(5.13)

The programs that generate the input and implement the set cover algorithms are all implemented in ANSI C and thus can run on Microsoft Windows or Unix. The experimental evaluation of the set cover algorithms consists of varying \(M\), \(N\), and \(E[K]\). In the case of \(E[K]\), the peaked distribution defined in equation (5.9) is used. For each experiment that varies \(M\), \(N\), and/or \(E[K]\) the resulting set cover size and the number of comparisons are measured.
In the case where the input sets are distributed over $M$ P2P hosts, it is assumed that each host is equivalent to a processor. Thus, the experiments for distributed set cover measure the number of comparisons for $M$ processors. These $M$ processors were simulated over a single processor by counting separately the number of comparisons of each processor (set) using an array of counters (one for each set). That is, by sequentially selecting each set and counting the total number of comparisons required for that set by RMO. In contrast, for the single processor case, it is assumed that a single P2P host stores all input sets and all set cover algorithms are executed using the single (one) processor.

The programs that generate the input and evaluate the set cover algorithms are instrumented to count the number of sets in the minimal set cover size and the number of element comparisons made. Figure 5.7 summarizes the complete procedure used to experimentally evaluate the set cover algorithms. Also, Figures 5.5 and 5.6 show the details of the command `create $M$ subsets` in Figure 5.7. The selection of the experiment parameters, $M$, $N$, and $K$ are taken from measurement studies of real P2P networks and from the P2P networks literature.

**Figure 5.7 Set cover evaluation process**

```plaintext
for $i = 1$ to NumReplications
  set random number seed to $i$
  create $M$ subsets
  execute Greedy algorithm
  execute RMO algorithm

calculate mean minimal set cover size for each algorithm
calculate mean number of comparisons for each algorithm
```
Because $N$ changes over time, a range of values were depicted from the studies of Saroiu et al. in [97], Zhao et al. in [120], and Sripandkulchai et al. in [106]. The second parameter $M$ is the total number of P2P hosts. The total number of P2P hosts is estimated to be less than or equal to 78,125. This estimate is based on the following:

1. The maximum number of hops a message travels or time-to-live value is seven. This is taken from the file sharing P2P protocol specification in [111] and [62].
2. Lv et al. in [70] give the formula to calculate the total number of P2P hosts with the average degree of a host (six) and number of hops.

The third parameter $K$ is a random variable modeling the number of shared files by a P2P host. It is known from investigations in [15, 37, 97, 120] that the distribution of files in a P2P network is heavy-tailed, that is most of the files are shared by few hosts while most of the hosts share few files. Therefore, $K$ follows a peaked distribution which has the key properties similar to that of a heavy-tailed distribution. The peaked distribution is used because it can be “tuned” for different degrees of peakedness. Additionally from current P2P measurements, it is estimated that the expected number of files shared by a P2P host ranges between one hundred and one thousand files [97, 120]. A summary of the values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>total files shared</td>
<td>3600 to 7100</td>
</tr>
<tr>
<td>$M$</td>
<td>total number of hosts</td>
<td>$M \leq 78125$</td>
</tr>
<tr>
<td>$K$</td>
<td>heavy-tailed distribution</td>
<td>$100 \leq E(K) \leq 1000$</td>
</tr>
</tbody>
</table>
and distributions used by the three experiment parameters is shown in Figure 5.8 followed by their description.

5.4.2 Description of Experiments

A total of three experiments were designed to study the performance of the set cover algorithms. The first experiment uses the P2P values shown in Figure 5.8 to determine the performance for a typical P2P network. The second experiment studies how an increase in the number of hosts affects the set cover metrics. Also, a decreasing ratio of elements per set \( \frac{N}{M} \) is studied as \( M \) increases for the second experiment. The third experiment investigates how the set cover metrics are affected by increasing the total number of files shared in the network. In contrast, to the second experiment, in the third experiment an increasing ratio of elements per set is studied as \( N \) increased. The experiments investigate values that are representative values, and give insight to the

<table>
<thead>
<tr>
<th>Experiment #1: Representative values</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ( N = 5000 )</td>
</tr>
<tr>
<td>• ( M = 5000 )</td>
</tr>
<tr>
<td>• ( E[K] = 500 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment #2: Range of hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ( N = 5000 )</td>
</tr>
<tr>
<td>• ( M ) is a range from 100 to 1000</td>
</tr>
<tr>
<td>• ( E[K] = 50 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment #3: Range of files shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ( N ) is a range from 5000 to 15000</td>
</tr>
<tr>
<td>• ( M = 1000 )</td>
</tr>
<tr>
<td>• ( E[K] = 50 )</td>
</tr>
</tbody>
</table>

*Figure 5.9 Experiment design summary*
tendencies of the performance of the RMO and Greedy algorithms. Two key questions addressed are, as $K$ increases in peakness which set cover algorithm is better? And, how does the ratio of elements per set affect performance? Figure 5.9 gives the name of each experiment and description. For each of the experiments the set cover algorithms are executed for the case of a single processor (set cover is not distributed) and $M$ simulated processors (set cover is distributed). The parameter $\alpha$ of the peaked distribution is the “degree of peakness” or a multiplier that makes the host with the most files have $\alpha$ times many more files than any other host.

In order to study experimentally how the distribution of $K$ affects the set cover metrics, three cases for experiments #2 and #3 are defined using the peaked distribution. The three cases for the peaked distribution are:

- *Uniform* or not peaked ($\alpha = 1$)
- *Slightly* peaked ($\alpha = 10$)
- *Heavily* peaked ($\alpha = 100$)

In the slightly peaked distribution one host is sharing ten times more files than any one other host. The value of $\alpha$ for the highly peaked case is ten times the value of $\alpha$ in the slightly peaked case, and thus for the highly peaked distribution one host is sharing one hundred times more files than any one other host. Each case of the peaked distribution applies to all experiments in Figure 5.9. Each experiment is repeated for each value of $\alpha$ (three cases per experiment). All experiments are run for 1000 replications for which the mean set cover size and mean number of element comparisons are determined.
5.4.3 Numerical Results

In this section the results and observations for each of the three experiments are explained. A summary of the key results and conclusions are briefly discussed at the end.

5.4.3.1 Experiment #1 – Representative Values

Results for the representative values are shown in Tables 5.10 and 5.11 for one processor and for \( M \) simulated processors respectively. These results are the measures for the performance metrics of the two set cover algorithms (Greedy and RMO). The three rows in Tables 5.10 and 5.11 are classified according to the distribution used for \( K \).

In the first row, \( K \) is distributed uniformly, in the second row \( K \) is slightly peaked, and in the third row \( K \) is highly peaked.

It can be seen that for \( K \) uniform:

- The number of comparisons of RMO is 98% fewer than Greedy.
- The cover size of RMO is 43% larger than set cover size of Greedy.
- With respect to \( M \), RMO has 2% more sets in the set cover than Greedy.

For \( K \) slightly peaked:

- The number of comparisons of RMO is 98% fewer than Greedy.
- The cover size of RMO is 49% larger than set cover size of Greedy.
- With respect to \( M \), RMO has 2% more sets in the set cover than Greedy.

For \( K \) highly peaked:

- The number of comparisons of RMO is 27% fewer than Greedy.
- The cover size of RMO is 784% larger than set cover size of Greedy.
- With respect to \( M \), RMO has 5% more sets in the set cover than Greedy.
Overall, for the one processor case RMO is suitable when $K$ is uniform and slightly peaked. For the highly peaked case, Greedy results in both fewer comparisons (by about a factor of 3.5) and a smaller set cover size (by about a factor of 9) than RMO. For this case, RMO is not suitable. The explanation for this case is discussed later in section 5.4.4.
For the case of distributed RMO and Greedy with $M$ hosts (that is, $M$ processors), the set cover size results for one processor (Table 5.10) and $M$ processors (Table 5.11) are the same. However, the number of comparisons for one and $M$ processors are significantly different. This is as expected from the explanation at the end of section 5.3 of this chapter. For $M$ processors, the number of comparisons for RMO is reduced by a factor of $M$ with respect to the results for one processor. The number of comparisons required by Greedy does not change (that is, it is the same for one or $M$ processors).

### 5.4.3.2 Experiment #2 – Number of Hosts

The results for Experiment #2 show how an increase in the number of hosts and peakness of $K$ affect set cover performance metrics. The three results of the one processor case for which $K$ is distributed uniformly, slightly peaked, and highly peaked are shown in Figures 5.10, 5.11, and 5.12, respectively.
In Figure 5.10, RMO has a smaller set cover size than Greedy for $N_{sets} \leq 500$, and as $M$ increases the set cover size remains, on average, constant at about 330. Also, the
number of comparisons for RMO is significantly fewer than for Greedy as $M$ is increased.

The minimal set cover size and number of comparisons in Figure 5.11 is the roughly the same as in Figure 5.10. However, for Greedy and RMO the set cover size in Figure 5.12 is smaller (about three times smaller) than the set cover size in Figures 5.10 and 5.11. Set cover size decreases when $K$ is highly peaked. The number of comparisons in Figure 5.12 is significantly reduced for Greedy, but remains roughly the same for RMO.

For the case of $M$ processors Figures 5.13, 5.14, and 5.15 show the results of increasing $M$ and peakness of $K$ for the number of comparisons. The minimal set cover size is not included because it does not change from the results shown in Figures 5.10, 5.11, and 5.12.

Figure 5.13 Experiment #2-M processors with $K$ uniform
From Figures 5.13, 5.14, and 5.15 it can be observed that:

- The number of comparisons for RMO is significantly less than Greedy.
- The number of comparison decreases as $K$ increases in peakness.

![Figure 5.14 Experiment #2-M processors with $K$ slightly peaked](image1)

![Figure 5.15 Experiment #2-M processors with $K$ highly peaked](image2)
In summary, for experiment #2 it can be seen that:

- RMO at most has 1% more sets with respect to $M$ in set cover than Greedy.
- RMO is suitable for $K$ uniform and slightly peaked. RMO has 70% fewer comparisons and 4% larger cover size than Greedy.
- RMO unsuitable for $K$ highly peaked.
- Comparing the results between one processor and $M$ processors yields that RMO has $M$ times fewer comparisons than Greedy. Additionally, Greedy does not have a reduction in the number of comparisons.

**5.4.3.3 Experiment #3 – Number of Files**

In Experiment #3 the effect of increasing the number of files was studied. For one processor, Figures 5.16, 5.17, and 5.18 show results for increasing number of files and peakness of $K$. In Figure 5.16, as the number of files is increased, minimal set cover size increases (Greedy and RMO). Also, RMO has fewer comparisons than Greedy.
The results in Figure 5.16 and 5.17 are similar. That is, for $K$ uniform and slightly peaked, the minimal set cover size and number of comparisons are about the same.
The results in Figure 5.18 vary from Figures 5.16 and 5.17 in that Greedy has significantly fewer comparisons.

Figure 5.19 Experiment #3-M processors with K uniform

Figure 5.20 Experiment #3-M processors with K slightly peaked
For $M$ processors, again the results shown in Figures 5.19, 5.20, and 5.21 are similar to those obtained in Experiment #2. Thus, RMO has significantly fewer comparisons and set cover size is at most 5% more than Greedy.

### 5.4.4 Summary and Discussion of Results

Overall, a set cover algorithm is better, if it has significantly fewer comparisons (i.e., at least 50% fewer comparisons), and roughly the same set cover size. The observations of the results from all experiments are as follows.

- RMO is better as the ratio of elements per set ($N/M$) increases.
- RMO is better for $K$ uniform or slightly peaked.
- RMO has $M$ times fewer comparisons with $M$ processors and Greedy does not have a reduction in comparisons.
- Greedy is better for $K$ highly peaked and the ratio of elements per set decreases.

From the above observations it can be seen the set cover size for RMO and Greedy are roughly the same, but RMO has at least 50% fewer comparisons than Greedy for $K$.
uniform and slightly peaked. These results are not unexpected because for each set in the cover, Greedy is required to determine the set with the largest number of uncovered elements (that is, the maximum set) and finding this maximum set by Greedy requires many more comparisons than by RMO. RMO only needs to compare the elements of each set with the universe to find the set cover. Also, RMO is better as the ratio of elements per set \(\frac{N}{M}\) increases because of the possibility of a set containing one unique element increases. Therefore, the size of the set cover will increase, and Greedy will need more comparisons to locate the set(s) with a unique element.

For \(K\) highly peaked and decreasing ratio of elements per set, the performance of Greedy is better than RMO. This result is expected because the set cover size obtained by Greedy is relatively small (about 1% of 5000 sets) and can be explained by the following two reasons. The first reason is that the first host stores 98% of the files. The second reason is that as \(M\) is increased the possibility of a host having a unique file (that is, the ratio of elements per set decreases) is decreased. There are fewer files to be assigned among the \(M - 1\) hosts (that is, all \(M\) hosts except the first host).

The energy saving obtained by powering down redundant hosts or hosts that do not belong in the set cover for Experiment #1 (Representative values) is based on the following:

- The power consumption of a desktop PC or host is 60W when powered on and 6W when powered down (sleep state) [41].
- All P2P hosts are powered on 24/7.
- There is no user activity two thirds of the time a given host is powered on [23].
Therefore, for a 24 hour day, a redundant host can be powered down for up to 16 hours. With the average cost of electricity in the U.S. being $0.08 dollars/kWh \[34\], the total power consumption the hosts for Experiment #1 when fully powered on is 268 MWh/year or $210,240 dollars/year. By powering down the redundant hosts in Experiments #1 with the results obtained by RMO, the yearly savings achieved on average is 2.5 GWh/year or $202,758 dollars/year. The yearly savings is obtained by subtracting the power consumption of the hosts that are not in the set cover (i.e., powered down) from the total power consumption of all hosts fully powered on. The power consumption of the hosts that are not in the set cover is calculated by multiplying the cost of electricity per kWh, by the total kWh consumption of the hosts powered down. Thus, the total power consumption of 5000 fully powered on hosts can be reduced by up to 96% if RMO is used.

5.5 Probability of File Loss for Distributed RMO

Because each host executes the distributed RMO algorithm independently of other hosts, it is possible that a given file could be lost resulting in an incomplete or partial set cover. This would occur only if all hosts sharing the same file remove themselves from the set cover at the same time. A host in a P2P network removes itself from the set cover by powering down.

The two conditions that must be satisfied so that a file $f$ is lost are:

1. The file $f$ is shared only by non-unique hosts. A host is non-unique if all of its shared files are shared by at least one other host. Thus, file $f$ is shared by two or more hosts such that only one host is needed for the set cover to cover file $f$. 

123
2. All hosts sharing file \( f \) execute the RMO remove (distributed RMO that determines if a host should be removed from the set cover) at the same time. “Same time” means that there is not sufficient time for updates to the host data structure to have propagated between the hosts. Therefore, the hosts erroneously remove themselves from the cover assuming the other hosts in the “collision” are sharing the file \( f \).

An example that illustrates the two conditions necessary for file \( f \) to be lost is as follows. Assume a P2P network with three hosts (1, 2, 3) and three files (a, b, c) as shown in Figure 5.22. Host 1 shares files a and c, host 2 shares b and c, and host 3 shares file b. There are two possible set covers. The first solution has host 1 and host 2 in the set cover. The second solution has host 1 and host 3 in the set cover. If both non-unique hosts 2 and 3 execute the RMO remove at the same time, then both hosts will remove themselves from the network at the same time and file b is lost. If hosts 2 and 3
execute the RMO remove at different times (say, host 2 executes first), updates will cause the other host (host 3 in this case) not to be removed from the cover since it is the only host sharing file $b$ in the P2P network.

When a file is lost, the file is not lost permanently. The file is only lost for the period of time the host sharing the file remains powered down and out of the network. When the host is again active (i.e., due to user interaction with the host) and powered-up, the lost file will again be shared. What needs to be determined is what percentage of files are temporarily “lost” and for what period of time. The following factors need to be considered:

- Distribution of files in hosts to determine unique and non-unique files and hosts.
- Probability of a given host timing-out its inactivity timer and attempting to remove (that is, power-down) in a given time period (or time slot).
- The order that hosts time-out and remove.
- Amount of time needed for updates to be sent and received by all affected hosts.

An analytical model can be developed to give insight into the probability of losing a file. This model can be used to bound the probability of file loss for various realistic cases.

### 5.5.1 The Machine Failure Problem

The probability of losing a given file can be re-stated as a generic “machine failure problem” given in Figure 5.23. In the machine failure problem, the probability that all machines fail by a time $T$ corresponds to the probability that a file is lost by time $T$. Machine failure corresponds to RMO remove. The probability of machine failure ($p$) corresponds to the probability of an RMO remove in a given time slot. The time slot
“Suppose we have $M$ machines each running at time 0. At times 1, 2, … , each machine may fail with probability $p$. These failures are independent except that if exactly one machine is left, it does not fail again.”

Figure 5.23 Machine failure problem definition

duration is the time it takes for updates to be sent and received. Thus, if two or more hosts execute an RMO within this update time of each other they all potentially remove.

In discussion with Allen Roginsky (personal communications, December 2006), the exact solution for the Machine Failure problem was developed. Let $p_{t,k}$ be the probability that at time $t$ exactly $k$ machines are running. We are interested in the probability that all machines will fail by time $T$, that is $P_{T,0}$. That is,

$$P_{T,0} = \sum_{j=1}^{T} \Pr[\text{All machines fail at time } j]. \quad (5.15)$$

The probability of any given machine to be running at time $t$ is $(1 - p)^t$, if it were not for the possibility that this was the only machine running by time $k < t$ then the formula is correct. Using this fact, $P_{T,0}$ can be derived as follows:

$$P_{T,0} = p^M + \sum_{t=2}^{T} \sum_{k=2}^{M} \Pr[\text{Exactly } k \text{ machines running at time } t-1] p^k \quad (5.16)$$

With the definition of $p_{t,k}$, the $P_{T,0}$ in equation (5.16) is

$$P_{T,0} = p^M + \sum_{t=2}^{T} \sum_{k=2}^{M} p_{t,k} p^k. \quad (5.17)$$

Using the binomial distribution for $k > 2$, $p_{t,k}$ is
\[ p_{t,k} = C_M^k \left( (1 - p)^t \right)^k \left( 1 - (1 - p)^T \right)^{M-k}. \] 

Substituting \( p_{t,k} \) in equation (5.18) yields

\[ P_{T,0} = p^M + \sum_{t=2}^{T} \sum_{k=2}^{M} C_M^k \left( (1 - p)^{t-1} \right)^k \left( 1 - (1 - p)^{t-1} \right)^{M-k - k} p^k. \] 

Equation (5.19) is not a computable solution for large values of \( M \) as it contains the combinatorial \( C_M^k \). Further simplification of equation (5.19) is obtained by grouping the terms which have \( K \) as exponent and using the binomial theorem with \( x = (1 - p)^{t-1} p \) and \( y = 1 - (1 - p)^{t-1} \) for the expansion of the power of sums. The probability of a collision from equation (5.19) is then simplified to

\[ P_{T,0} = p^M + \left[ 1 - (1 - p)^T \right]^M - (-1)^M p^M - \left( \frac{M^p}{1-p} \right) \sum_{t=2}^{T} \left[ 1 - (1 - p)^{t-1} \right]^{M-1} (1 - p)^{t-1}. \] 

**5.5.2 Numerical Results**

In this section the probability of a file being lost is computed for a range of parameters. The parameters of interest are:

- \( M \) = number of hosts sharing a file that is vulnerable to being lost.
- \( T \) = number of time slots in a day where a time slot is the period of time needed for updates to be sent and received.
- \( N_p \) = number of times in a 24 hour day a typical host tries to power down due to its inactivity timer having timed out.
- \( D_{slot} \) = length of a time slot.

From these parameters the probability of a given host executing an RMO remove for a given time slot is
In equation (5.21) the denominator is the number of seconds in one day.

Given two hosts ($M = 2$) where each host has an inactivity time-out once per hour ($N_p = 24$) and the length of a time slot is 1 second ($D_{slot} = 1$), we get $P_{T,0} = 0.000139$. That is, with probability 0.000139 both hosts will have removed from the P2P network at least once in 24 hours at the same time. This is a very small probability of file loss. The amount of time that both hosts will be powered down (and thus the file(s) in question are lost) is not determined from this calculation. The parameter values for $D_{slot}$ and $N_p$ can be determined only experimentally. The value of $D_{slot}$ depends on the size and congestion level of the P2P network – a value of 1 second is probably “reasonable”. The value of $N_p$ depends on the level of user activity with the P2P hosts. Figure 5.24 shows a plot of $N_p$ versus probability of a lost file ($P_{T,0}$) for $M = 2$. The value of $N_p$ is ranged from 1

![Figure 5.24 Probability of a lost file ($P_{T,0}$)](image-url)
inactivity time out per hour to 10 per hour (so, 24 per day to 240 per day). The result is strictly linear.

To get a better understanding of file loss a simulation experiment was run. For the simulation experiment, Experiment #1 from Section 5.4.2 was modified so that multiple hosts could remove at the same time. The simulation was run for 100,000 replications with \( N_p \) ranging from 1 to 10 per hour. The resulting probability of one or more files being lost for any period of time was effectively 0.

Further work is needed to develop a better analytical understanding of incomplete set cover in a distributed RMO implementation. However, the above results show that RMO is a practical solution for distributed P2P power management.
Chapter 6:  
Summary and Directions for Future Research

In 2006 the overhead traffic from P2P file sharing networks accounted for about 20% of the traffic on the Internet [56]. In addition, the nine million hosts in the U.S. that participated in P2P file sharing consumed approximately 4.9 TWh/yr or $392 million/year [42, 80, 86]. Much of this electricity may have been wasted due to hosts sharing redundant files (that is, files already shared by other hosts). This dissertation has investigated new methods for reducing P2P overhead traffic and contains a new and novel approach for power management of P2P protocols.

The reduction of overhead traffic was investigated in two directions. In the first direction, statistics of where previous files were downloaded were used to target future searches to hosts with expected high likelihood (i.e., those hosts that have satisfied file searches in previous iterations) to contain a searched-for file. In Targeted Search up to \( M_{top} \) query message are sent unicast and only if these query messages do not result in a hit is the query message fully broadcast to all hosts. Targeted Search was shown to reduce overhead traffic by 63% at an expense of only doubling the search time when compared to a fully broadcast based approach (e.g., such as Gnutella version 0.4). The Targeted Search method was implemented in a P2P host software release named Ditella which is fully compatible with Gnutella.
In the second direction to reduce overhead traffic, the query broadcast paradigm was reversed to one of broadcasting what files are shared, building a local files shared directory, and then doing search locally to a host (i.e., without a broadcast query). This Broadcast Update with Local Lookup Search (BULLS) protocol was designed and evaluated using flow models. It was shown that BULLS reduces Gnutella’s overhead traffic by a maximum of 38% when hosts stay in the network for more than 15 hours. The BULLS protocol was also designed with the goal of enabling global power management.

Global P2P power management was investigated as a set cover problem. Redundant hosts are identified and powered-down. A redundant host is one which shares only files also shared by other hosts. A new set cover heuristic called Random Map Out (RMO) was developed that could be distributed to all the hosts in a P2P network. RMO achieved a reduction in computation time (where distributing the well-known Greedy heuristic would not result in reduced computation time). It was experimentally shown that the RMO with respect to $M$ (total number of sets) has 5% more sets in the cover and significantly less number of comparisons. Because each host in a P2P network executes the distributed RMO independently of other hosts, it is possible that a given file could be “lost” (i.e., lost to sharing) if all the hosts sharing a given file try to remove from the network at exactly the same time. The analysis of this so-called probability of file loss is defined and modeled as a generic problem called the “Machine failure problem”. An exact solution is derived and studied. Numerical results show that for representative values of a P2P network, the probability of file loss is 0. Further work is needed to develop a better analytical understanding of a distributed RMO implementation. However, the set cover approach to P2P networks is a promising direction for power
management and can result in significant energy savings by enabling redundant hosts to power down.

6.1 Future Research

This research has brought to light new questions in reducing P2P overhead traffic and enabling power management. In addition, the set cover approach studied for P2P power management may have applications beyond those studied in this dissertation. New questions and possible directions for studying them are:

1. Topology may play a key role in determining the actual overhead from queries. This has been established in [30, 70, 79, 118]. The effects of topology need to be studied, in particular within the context of the Targeted Search method.

2. BULLS can be improved by limiting the broadcast of file updates. Methods using Bloom filters (e.g., as applied to caches in [36, 75, 91]) need to be explored.

3. Redundant files are often desired for reasons of load balancing and improving availability. Extending RMO to allow for redundant files should be investigated. A possible direction to do this is by allowing a host to be removed from the set cover when there are at least a given number of hosts sharing its files.

4. It may be interesting to better analyze the average running time of RMO to see if the cost of implementing RMO in real P2P networks is feasible.

5. Looking beyond BULLS, can P2P power management be fully distributed without the use of a global data structure? This important question merits further study.
Set cover is a common component of problems in fields ranging from DNA studies to graphics visualization. New applications of the RMO heuristic may be useful in:

- DNA typing for diagnosis and sickness treatment. In particular, given a set $V$ of viruses and a set $E$ of enzymes, finding the minimum subset of enzymes which allow distinguishing all types of viruses in $T$ is a set cover problem [94].
- Scene visualization for virtual world exploration. The set cover problem in scene visualization can be states as: given a family of images $I$, the set cover is the minimum number of images that enable all vertices of the scene to be seen [104].
References


URL: http://standards.ieee.org/getieee802/802.3.html.

[51] Internet World Stats (Usage and Population Statistics), the USA Internet Usage and Population, July 2006.


[114] Website of Aurora an open source Freenet simulator.  


Graciela Perera received a B.S. in Systems Engineering from Metropolitan University and a M.S. in Computer Science from Simón Bolívar University; both universities are located in Caracas, Venezuela. While pursuing the M.S. in Computer Science she worked as a teaching assistant for algorithms and data structures courses. After graduating with an M.S. from Simón Bolívar University in 1997 she worked five years for the same University as an instructor (Profesor Asistente) and was a member of the technical support staff in the foundation of research and development. She also served as a consultant to Venezuela’s Oil company, PDVSA, and the National Library. She joined the Ph.D. program in the Department of Computer Science and Engineering at the University of South Florida in 2003. While in the Ph.D. program she has worked both as a teaching and research assistant. She has served as a community service officer for the Venezuela student organization association at the University of South Florida. Her professional affiliations include ACM, ASEE, and IEEE.