MEASURING AND CHARACTERIZING PROPERTIES OF PEER-TO-PEER SYSTEMS

by

DANIEL STUTZBACH

A DISSERTATION

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

December 2006

"Measuring and Characterizing Properties of Peer-to-Peer Systems," a dissertation prepared by Daniel Stutzbach in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

Prof. Reza Rejaie, Chair of the Examining Committee

Date

Committee in charge:
Prof. Reza Rejaie, Chair
Prof. Andrzej Proskurowski
Prof. Virginia Lo
Prof. David Levin
Dr. Walter Willinger

Accepted by:

Dean of the Graduate School

An Abstract of the Dissertation of

Daniel Stutzbach for the degree of Doctor of Philosophy
in the Department of Computer and Information Science
to be taken December 2006

Title: MEASURING AND CHARACTERIZING PROPERTIES OF
PEER-TO-PEER SYSTEMS

Approved: ______________________________
Prof. Reza Rejaie

Peer-to-peer systems are becoming increasingly popular, with millions of simultaneous users and a wide range of applications. Understanding existing systems and devising new peer-to-peer techniques relies on access to representative models, derived from empirical observations, of user behavior and peer-to-peer system behavior on a real network. However, it is challenging to accurately capture behavior in peer-to-peer systems because they are distributed, large, and rapidly changing. While some prior work does study the properties of peer-to-peer systems, they do not quantify the accuracy of their measurement techniques, sometimes leading to significant error.

This dissertation empirically explores and characterizes a wide variety of properties of peer-to-peer systems. The properties examined fall into four groups, along two axes: properties of peers versus properties of how peers are connected, and static properties versus dynamic properties. To study these properties, this dissertation develops and assesses two measurement techniques: *(i)* a crawler for capturing global state and *(ii)* a Metropolized random walk approach for collecting samples. Using these techniques to conduct empirical studies of widely-deployed peer-to-peer systems, this dissertation presents empirical results to suggest useful models for key properties of peer-to-peer systems. In the end, this dissertation significantly deepens our understanding of peer-to-peer systems and lays the groundwork for the accurate measurement of other properties of peer-to-peer systems in the future.

This dissertation includes my previously published and my co-authored materials.

CURRICULUM VITAE

NAME OF AUTHOR: Daniel Stutzbach

PLACE OF BIRTH: Attleboro, MA, U.S.A.

DATE OF BIRTH: March 28, 1977

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Worcester Polytechnic Institute

DEGREES AWARDED:

Doctor of Philosophy in Computer and Information Science, 2006, University of Oregon
Bachelor of Science in Electrical Engineering, 1998, Worcester Polytechnic Institute

AREAS OF SPECIAL INTEREST:

Peer-to-Peer Networks, Network Measurement

PROFESSIONAL EXPERIENCE:

President, Stutzbach Enterprises, LLC, 2006—
Research Assistant, University of Oregon, 2001–2006
Software Engineer Contractor, ADC, Inc., 2001
Senior Software Engineer, Assured Digital, Inc., 1999–2001
Software Test Engineer, Assured Digital, Inc., 1998–1999
Embedded Systems Programmer, Microwave Radio Communications, 1997–1998

AWARDS AND HONORS:

IMC Travel Grant, 2006
Clarence and Lucille Dunbar Scholarship, 2006–2007
Upsilon Pi Epsilon Membership, 2006
INFOCOM Travel Grant, 2006
First place, UO Programming Competition, 2006
SIGCOMM Travel Grant, 2005
First place, UO Programming Competition, 2005
IMC Travel Grant, 2004
ICNP Travel Grant, 2004
First place, UO Programming Competition, 2004
Honorable Mention, ACM ICPC World Finals Programming Competition, 2003
First place, UO Programming Competition, 2003
First Place, ACM ICPC Pacific Northwest Programming Competition, 2002
First place, UO Programming Competition, 2002
First place, WPI ACM Programming Competition, 1998
First place, WPI Tau Beta Pi Design Competition, 1997
Scholarship Winner, Rhode Island Distinguished Merit Competition in Computer Science, 1995

PUBLICATIONS:

D. Stutzbach and R. Rejaie, “Understanding churn in peer-to-peer networks,” in *Proc. Internet Measurement Conference*, Rio de Janeiro, Brazil, Oct. 2006.

D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, “On unbiased sampling for unstructured peer-to-peer networks,” in *Proc. Internet Measurement Conference*, Rio de Janeiro, Brazil, Oct. 2006.

D. Stutzbach and R. Rejaie, “Improving lookup performance over a widely-deployed DHT,” in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "Sampling techniques for large, dynamic graphs," in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

A. Rasti, D. Stutzbach, and R. Rejaie, "On the long-term evolution of the two-tier Gnutella overlay," in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

J. Li, R. Bush, Z. M. Mao, T. Griffin, M. Roughan, D. Stutzbach, and E. Purpus, "Watching data streams toward a multi-homed sink under routing changes introduced by a BGP beacon," in *Proc. Passive and Active Measurement Workshop*, Adelaide, Australia, Mar. 2006.

S. Zhao, D. Stutzbach, and R. Rejaie, "Characterizing files in the modern Gnutella network: A measurement study," in *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 2006.

D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," in *Proc. Internet Measurement Conference*, Berkeley, CA, Oct. 2005, pp. 49–62.

D. Stutzbach and R. Rejaie, "Characterizing the two-tier Gnutella topology," Extended Abstract in *Proc. SIGMETRICS*, Banff, AB, Canada, June 2005.

D. Stutzbach, D. Zappala, and R. Rejaie, "The scalability of swarming peer-to-peer content delivery," in *Proc. IFIP Networking*, Waterloo, Ontario, Canada, May 2005, pp. 15–26.

D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the Gnutella network," in *Proc. Global Internet Symposium*, Miami, FL, Mar. 2005, pp. 127–132.

D. Stutzbach and R. Rejaie, "Evaluating the accuracy of captured snapshots by peer-to-peer crawlers," Extended Abstract in *Proc. Passive and Active Measurement Workshop*, Boston, MA, Mar. 2005, pp. 353–357.

ACKNOWLEDGMENTS

I would like to thank my parents for providing me with the intellectual curiosity and work ethic required for any dissertation. Alisa Rata deserves special thanks for her emotional support and encouragement that greatly accelerated my progress. Her unending patience while I worked on my papers and this dissertation are particularly appreciated.

My thanks also go out to Prof. Daniel Zappala for his tutelage during my initial years in graduate school, particularly for helping me to understand the difference between science and engineering.

Early in my graduate school career, I had many fruitful discusses on the Gnutella Developer Forum mailing list, particularly with Greg Bildson, Raphael Manfredi, Serguei Osokine, Gordon Mohr, and Vinnie Falco.

I would also like to thank Dr. Subhabrata Sen, Dr. Walter Willinger, Dr. Nick Duffield, Prof. Andrzej Proskurowski, Prof. Virginia Lo, and Prof. David Levin for insightful comments that opened me to new ideas and developing my sense of scientific rigor. I am also grateful for my friends and fellow Ph.D. students, Peter Boothe and Chris GauthierDickey, for their companionship and many stimulating conversations. Amir Rasti, Shanyu Zhao, and John Capehart are particularly thanked for their collaborative work that contributed to Chapters 5 and 6.

The help of Joel Jaeggli, Lauradel Collins, Paul Block, and other members of the systems staff at the University of Oregon are greatly appreciated for assisting with hardware and software support for my experiments and post-processing. I am particularly appreciative of their patience when dealing with security concerns from alarmist security administrators and performance issues when I slowed their file server to a crawl. Robin High provided me with several useful insights and pointers on statistically analysis.

My work on this dissertation was supported in part by by the National Science Foundation (NSF) under Grant No. Nets-NBD-0627202 and an unrestricted gift from

Cisco Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF or Cisco. I am also grateful to the federal Stafford Loan program which provided me loans that greatly improved my quality of life while a graduate student.

BitTorrent tracker logs used in my study of churn in Chapter 7 were provided by Ernst Biersack and others from the Institut Eurecom, the Debian organization, and 3D Gamers. I am thankfully for their generosity in releasing this data.

Finally, I am very thankful for the guidance of my adviser, Prof. Reza Rejaie, who continually pushed me to dig deeper and ask the next question.

To my parents,
Ron and Joan Stutzbach

TABLE OF CONTENTS

Chapter Page

1. INTRODUCTION 1

2. BACKGROUND 6

2.1 History 6
2.2 Modeling and Distributions 9
2.3 Graph Theory 14

3. RELATED WORK 17

3.1 Measurement Techniques 17
3.2 Measurement Results 20
 3.2.1 Static Peer Properties 22
 3.2.2 Dynamic Peer Properties 23
 3.2.3 Static Connectivity Properties 26
 3.2.4 Dynamic Connectivity Properties 27
3.3 Summary 29

4. CAPTURING GLOBAL STATE 31

4.1 The Design of Cruiser 33
4.2 Effect of Unreachable Peers 35
4.3 Other Practical Considerations 37
 4.3.1 Exhausting File Descriptors 37
 4.3.2 Filling Firewall/NAT Tables 38
 4.3.3 Oversensitive Intrusion Detection Systems 38
4.4 Quantifying Snapshot Accuracy 39
4.5 Summary 42

Chapter Page

5. CAPTURING SAMPLES . . . 44

5.1 Related Work . . . 47
5.2 Sampling with Dynamics . . . 49
5.3 Sampling from Static Graphs . . . 51
5.4 Sampling from Dynamic Graphs . . . 56
    5.4.1 Adapting Random Walks for a Dynamic Environment . . 58
    5.4.2 Evaluation Methodology . . . 58
    5.4.3 Evaluation of a Base Case . . . 60
    5.4.4 Exploring Different Dynamics . . . 61
    5.4.5 Exploring Different Topologies . . . 65
5.5 Empirical Results . . . 69
    5.5.1 Ion-Sampler . . . 69
    5.5.2 Empirical Validation . . . 71
5.6 Discussion . . . 76
    5.6.1 How Many Samples are Required? . . . 76
    5.6.2 Unbiased versus Biased Sampling . . . 77
    5.6.3 Sampling from Structured Systems . . . 77
5.7 Summary . . . 78

6. STATIC PEER PROPERTIES . . . 79

6.1 Geography . . . 79
6.2 File Properties . . . 81
    6.2.1 Related Work . . . 83
    6.2.2 Measurement Methodology . . . 84
    6.2.3 Capturing Complete Snapshots . . . 85
    6.2.4 Capturing Unbiased Samples . . . 86
    6.2.5 Dataset . . . 86
    6.2.6 Challenges and Problems . . . 89
    6.2.7 Empirical Observations . . . 90
    6.2.8 Summary . . . 100

7. DYNAMIC PEER PROPERTIES . . . 102

7.1 Related Work . . . 104
7.2 Background . . . 106
7.3 Pitfalls in Characterizing Churn . . . 109

Chapter Page

7.3.1 Missing Data 109
7.3.2 Biased Peer Selection 111
7.3.3 Handling Long Sessions 112
7.3.4 False Negatives 113
7.3.5 Handling Brief Events 114
7.3.6 NAT 116
7.3.7 Dynamic Addresses 117
7.4 Group-Level Characterization 118
7.4.1 Distribution of Inter-Arrival Time 120
7.4.2 Distribution of Session Length 123
7.4.3 Lingering after Download Completion 126
7.4.4 Distribution of Peer Uptime 128
7.4.5 Uptime Predictability 130
7.5 Peer-Level Characterization 132
7.5.1 Distribution of Downtime 134
7.5.2 Correlation in Session Length 136
7.5.3 Correlation in Availability 136
7.6 Design Implications 138
7.7 Summary 141

8. STATIC CONNECTIVITY PROPERTIES 143

8.1 Methodology 145
8.2 Overlay Composition 146
8.3 Node Degree Distributions 147
8.4 Reachability 153
8.5 Small World 157
8.6 Resilience 158
8.7 Summary 159

9. DYNAMIC CONNECTIVITY PROPERTIES 161

9.1 Stable Core 161
9.1.1 Examining Underlying Causes 168
9.2 Query Properties 170
9.2.1 Background 173
9.2.2 Theory versus Practice 176
9.2.3 Analysis of Kademlia's k-Buckets 179

Chapter Page

9.2.4 Accuracy of Routing Tables in Kad 186
9.2.5 Improving Lookup Efficiency 190
9.2.6 Improving Lookup Consistency 196
9.2.7 Related Work 201
9.2.8 Summary and Future Work 202

10. SUMMARY OF CONTRIBUTIONS 204

INDEX 209

BIBLIOGRAPHY 214

LIST OF FIGURES

Figure Page

2.1 Two-tier overlay topology 7
3.1 Relationship between session length, uptime, and remaining uptime . . . 24
4.1 Effects of timeout length 35
4.2 Effect of crawl speed on the accuracy of captured snapshots 40
4.3 Discovered information as a function of contacted peers 41
4.4 Error as a function of crawl duration 42
5.1 Bias of different sampling techniques 55
5.2 Time needed to complete a random walk 60
5.3 Comparison of sampled and expected distributions 62
5.4 Sampling error as a function of session-length 64
5.5 Sampling error as a function of number of connections 67
5.6 Degree distributions using the History mechanism 68
5.7 Sampling error as a function of maximum number of connections 70
5.8 Comparison of sampling versus crawling all peers 72
5.9 Difference between sampling and a crawl as a function of walk length . . 74
6.1 Breakdown of ultrapeers by continent 80
6.2 Population and reachability across captured snapshots 88
6.3 Distributions of files and bytes 94
6.4 Shared bytes as a function of shared files 96
6.5 Distribution of filesize across all files 97
6.6 Breakdown of file types 99
7.1 Inter-event time distributions 110
7.2 Upper-bound on false negatives 115
7.3 Inter-arrival time CCDFs 122
7.4 Session length CCDFs 124
7.5 Interactions with transfer time 127
7.6 Uptime of peers in the system 129
7.7 Remaining uptime as a function of uptime 131
7.8 Remaining uptime for peers already up 133
7.9 Downtime CCDFs 135
7.10 Correlation between consecutive session length 137
7.11 Correlation in availability 139
7.12 Distribution of appearances per day 140
8.1 Degree distributions of a slow and fast crawl 147

Figure Page

8.2 Top-level degree distribution in Gnutella 149
8.3 Mean degree as a function of uptime 150
8.4 Different angles of degree distribution in Gnutella 152
8.5 Search radius as a function of TTL in Gnutella 154
8.6 Different angles on path lengths 156
8.7 Nodes in largest component as a function of nodes removed 159
9.1 Number of stable peers and their external connectivity for different τ . . 164
9.2 Increased clustering among stable nodes 165
9.3 Different angles of connectivity with the stable core 167
9.4 Contribution of user- and protocol-driven dynamics 169
9.5 Routing table structures 181
9.6 Relative performance of different routing table structures 185
9.7 Routing table buckets as a function of the bucket's mask 189
9.8 Distributions of completeness and freshness in Kad's buckets 191
9.9 Effect of parallel lookup on hop count 193
9.10 Strict versus loose parallel lookup 195
9.11 Performance improvement gained by fixing the bugs in eMule 0.45a . . . 197
9.12 Lookup consistency 200

LIST OF TABLES

Table Page

1.1 Groups of properties . . . 4
3.1 File sharing measurement studies . . . 18
3.2 Summary of existing measurement techniques . . . 21
3.3 Observed session lengths in various peer-to-peer file sharing systems . . . 24
5.1 Kolmogorov–Smirnov statistic for techniques over static graphs . . . 54
5.2 Base case configuration . . . 60
5.3 KS statistic (D) between pairs of empirical datasets . . . 73
6.1 Peers that failed during the file crawl . . . 87
6.2 Free riders and number of shared files . . . 91
7.1 Measurement collections . . . 108
7.2 IP addresses eliminated due to NAT . . . 116
7.3 Number of observations for data presented in figures . . . 119
7.4 Number of observations for Fig. 7.7 and 7.8 . . . 120
8.1 Sample crawl statistics . . . 145
8.2 Distribution of implementations . . . 146
8.3 Small world characteristics . . . 158
10.1 Summary of characterization contributions . . . 205

CHAPTER 1

Introduction

The term *peer-to-peer* (P2P) refers to any network technology where the bulk of the hardware resources are supplied by end-users rather than a centralized service [1]. P2P is often contrasted with the traditional client-server paradigm, where the hardware resources are centralized. Traditional client-server applications require the content provider to provide resources proportional to the number of users. In the peer-to-peer paradigm, user systems ("peers") contribute resources. Because the available resources implicitly grow proportionally with the number of users, peer-to-peer systems have the potential to scale more smoothly with the user population than client-server systems.

The P2P paradigm began in earnest with the rise of the Napster file-sharing service, which provided an index for the vast stores of multimedia files on end-user systems. Since then, P2P has continued to increase in popularity, currently with millions of simultaneous users [2] and covering a wide range of applications, from file-sharing programs like LimeWire and eMule to Internet telephony services such as Skype. In particular, today's P2P file-sharing applications (*e.g.*, FastTrack, eDonkey, Gnutella) are extremely popular and contribute a significant portion of total Internet traffic [2–4]. Chapter 2 provides a more in-depth history of widely-deployed peer-to-peer applications and developments in peer-to-peer technology.

In lieu of a centralized service, P2P systems typically make use of user systems (called *peers*) connected together over the Internet. In effect, they form an *overlay* *network* over the physical network. One of the central problems of P2P research is to discover more efficient ways to accomplish a task over these overlay networks, which may include structured the overlay in special ways or using clever algorithms to coordinate peers.

Understanding existing systems and devising new P2P techniques relies on having access to representative models derived from empirical observations of existing systems. However, the large and dynamic nature of P2P systems makes capturing accurate measurements challenging. Because there is no central repository, data must be gathered from the peers who appear and depart as users start and exit the P2P application. Even a simple task such as counting the number of peers can be challenging since each peer can only report its immediate overlay neighbors. This dissertation addresses the following two fundamental questions:

- How do we collect accurate measurements from these systems?
- What are useful models to characterize their properties?

To answer these questions, we take the following steps: *(i)* develop and verify accurate measurements techniques, *(ii)* present empirical results which may be used to evaluate models and deepen our understanding of existing systems, and *(iii)* suggest useful models based on the empirical observations.

While prior studies have attempted to characterize different aspects of P2P systems, they have not taken the first step of critically examining their measurement tools (*i.e.*, answering the first question), leading to conflicting results [5] or conclusions that may be based on measurement artifacts (*e.g.*, power-law degree distributions [6–9] as evidenced in [10]). These measurement studies have been rather ad-hoc, gathering data in the most convenient manner without critically examining their methodology for measurement bias. While an ad-hoc approach is often suitable for first-order approximations (*e.g.*, “file popularity is heavily skewed”), it is generally not appropriate for making precise conclusions (*e.g.*, “session-lengths are power-law”). One of the largest gaps in the prior work is the development and validation of high-fidelity measurement tools for peer-to-peer networks. while Chapter 3 provides a survey of other P2P measurement studies.

Measurement: There are two basic approaches to collecting measurements from P2P systems, each with advantages and disadvantages. The first approach is to *capture global state* by collecting data about every peer in the system. The advantage of this approach is that all the information is made available for analysis. The typical problem with this approach is that the state changes while the measurement tool communicates with the peers, leading to a distorted view of the system. The longer the tool requires to capture the global state, the more distorted the data. Additionally, capturing global state does not scale well. As P2P systems grow larger, capturing global state becomes more time consuming, leading to greater distortion. To be at all practical, capturing global state requires an exceptionally fast tool able to gather data from a large number of peers very quickly. Chapter 4 introduces such a tool, called Cruiser, describes its design, and presents an evaluation of its performance.

The second approach is to *collect local samples*. The advantage of this approach is that it scales well. The Law of Large Numbers from statistics tells us that the average from a large number of samples will closely approximate the true average, regardless of the population size. One disadvantage of sampling is that we cannot easily use samples to examine certain properties which are fundamentally global in nature. For example, we cannot compute the diameter of a graph based on local observations at several peers. More importantly, if the collected samples are biased in some way, the resulting data may lead us to incorrect conclusions. For this reason, capturing samples requires validation of the sampling tool, which must be carefully designed to avoid bias. Chapter 5 examines the different causes of bias that can affect sampling in P2P systems, presents methods for overcoming those difficulties, and demonstrates the effectiveness of the methods by simulating under a wide variety of peer behavior, degree distributions, and overlay construction techniques, culminating in the `ion-sampler` tool.

Properties: Systematically tackling the problem of characterizing P2P systems requires a structured organization of the different components. At the most basic level, a P2P system consists of a set of connected peers. We can view this as a graph with the peers as vertices and the connections as edges. One fundamental way to divide the problem space is into *properties of the peers* versus *properties of the way peers are connected*. Another fundamental division is examining the way the system *is* versus the way the system *evolves*. In some sense, any property may change with time and could be viewed as system evolution. We use the term "static properties" to refer to properties that can be measured at a particular moment in time and modeled with a static model (*e.g.*, peer degree), and the term "dynamic properties" to refer to properties that are fundamentally dynamic in nature (*e.g.*, session length). Table 1.1 presents an overview of several interesting properties categorized by whether they are static or dynamic, and whether they are peer properties or connectivity properties. The properties in these four categories will be examined in detail in Chapters 6, 7, 8, and 9. Chapter 10 concludes the dissertation with a summary of characterized properties.

Inclusion of Published Material: Chapters 4 through 9 of this dissertation are based heavily on my published papers with co-authors [10–19]. Of these, in [15] and [17] the experimental work were performed by fellow graduate students, Shanyu Zhao and Amir Rasti, respectively, building upon my earlier experiments and data collection. In each of my other papers the experimental work is entirely mine, with my co-authors contributing technical guidance, editorial assistance, and small portions of

	Peer Properties	Connectivity Properties
Static Properties	Available resources (*e.g.*, files) Geographic location	Degree distribution Clustering coefficient Shortest path lengths Resiliency
Dynamic Properties	Session length Uptime Remaining uptime Inter-arrival interval Arrival rate	Stable core Search efficiency Search reliability

TABLE 1.1: Groups of properties

writing. Each chapter or major section in this dissertation includes additional details of the nature of my contributions.

Terminology: When a new term is introduced, it will be shown in *italics* and listed in the index. Some common terms used throughout this dissertation are introduced here, borrowing from graph theory and traditional (non-P2P) networking.

CHAPTER 2

Background

This chapter provides background material that the remainder of the dissertation relies on. Section 2.1 provides a brief history of the major peer-to-peer systems and significant steps in the evolution of peer-to-peer systems. Section 2.2 describes the goals of developing models from empirical data, describe the models most commonly used in the related work, and gives a high-level overview of the techniques available for validating the appropriateness of a model. Section 2.3 gives some formal definitions for graph theory concepts such as small-world and power-law graphs.

2.1 History

The term "peer-to-peer" gained widespread usage with the release of Napster in 1999. Unlike the traditional client-server model in which a centralized point (the server) provides resources to a large number of users (the clients), in the peer-to-peer paradigm the majority of the resources are contributed by other users. Napster facilitated the swapping of songs among users by providing a central index for all the songs available from all of Napster's users. While Napster provided the indexing service, the important resource—the songs—were provided by the users.

Facing legal pressure, Napster shut down in July 2001. Meanwhile in early 2000, Nullsoft released Gnutella, a file-sharing application with no central index. Instead, peers connect to one another to form a loose overlay mesh, and searches are performed by flooding the search to other peers nearby in the mesh. However, Nullsoft's parent company, American Online (AOL), shut down the project leaving the official Gnutella application orphaned. The protocol was quickly reverse-engineered and the Gnutella network is now composed of a family of third-party implementations. The Gnutella Developer Forum [20] acts as a loose standards body, allowing the protocol to grow and change while still allowing different implementations to function together.

In 2001, facing scalability problems after the shut down of Napster, Gnutella changed from a flat overlay to a *two-tier* overlay, shown in Figure 2.1. A fraction of well-provisioned peers become *ultrapeers* which act as indexes for the other peers, called *leaf peers*. The ultrapeers connect to one another, forming a loose mesh similar to the original Gnutella network.

Another advancement in Gnutella was the introduction of a new search mechanism called *Dynamic Querying* [21]. The goal in this scheme is to only gather enough results to satisfy the user (typically 50 to 200 results). It is similar in principle to an expanding ring search. Rather than forwarding a query to all neighbors, ultrapeers manage the queries for their leaves. Toward this end, an ultrapeer begins by forwarding a query to a subset of top-level connections using a low search radius (often called the time-to-live, or *TTL*). From that point on, the query is flooded outward until the TTL expires. The ultrapeer collects the results and estimates how rare matching results are. If matches are rare (*i.e.*, there are few or no responses), the query is sent through more connections with a relatively high TTL. If matches are more common but not sufficient, the query is sent down a few more connections with a low TTL. This process is repeated until the desired number of results are collected or the ul-

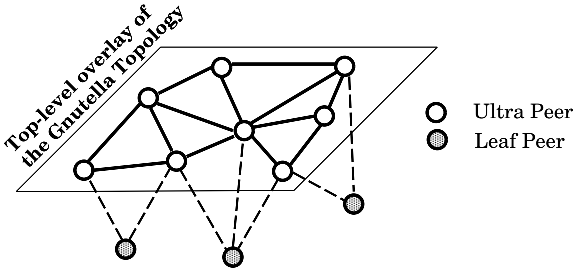


FIGURE 2.1: Two-tier overlay topology

trapeer gives up. Each ultrapeer estimates the number of visited ultrapeers through each neighbor based on the following formula: $\sum_{i=0}^{\mathrm{TTL}-1} (d-1)^i$, where d is the degree of the neighbor. The accuracy of this formula assumes that all peers have the same node degree, d. When Dynamic Querying was introduced, the number of neighbors each ultrapeer attempts to maintain was increased to allow more fine-grained control with Dynamic Querying by giving ultrapeers more neighbors to choose from.

At the time of Napster's shutdown, numerous other file-sharing systems sprang into existence, some of which are still widely popular today. The most prominent examples are Kazaa[1] and eDonkey, which use a two-tier overlay similar to modern Gnutella. Each of these three systems (Kazaa, eDonkey, and Gnutella) regularly has millions of simultaneous users [2].

Swarming: An interesting innovation called *swarming* allows peers to begin relaying a file to other peers before downloading it in full. Swarming allows the file to propagate more quickly through the network, which is an important feature when there is a rapid increase in demand. Although eDonkey and Gnutella also have this feature, it is most commonly associated with BitTorrent.

BitTorrent differs significantly from file-sharing applications such as Gnutella and Kazaa. Instead of offering a distributed search function, BitTorrent provides only a mechanism to transfer a file. Peers neighbor only with other peers downloading the same file, exchanging portions of the file (called *blocks* or *pieces*) until they have assembled the entire file. As a result, BitTorrent forms a separate overlay for each file.

Structured Overlays: In early 2001, the research community proposed a radical new method for search in peer-to-peer systems called Distributed Hash Tables (DHTs), also called structured overlays [22, 23]. In DHTs, each peer has an overlay address and a routing table. When a peer performs a query for an identifier, the query is routed to the peer with the closest overlay address. The DHT enforces rules on the way that peers select neighbors to guarantee performance bounds on the number

---

[1] The Kazaa application uses the FastTrack protocol. The terms are often used interchangeably.

of hops needed to perform a query (typically $O(\log |V|)$ where $|V|$ is the number of peers in the network).

However, initially it was unclear how well DHTs would be able to sustain their strict routing table rules in the face of rapid peer dynamics. Additionally, it was unclear how to efficiently implement a keyword-search over the identifier-query that DHTs provide. The research community continued to explore new variations of the DHT theme [24–26], evaluate DHT performance [5, 27–29], and develop applications making use of DHTs [30–32]. Despite, the excitement from the research community, application developers remained skeptical and there were no large-scale deployments.

That changed in 2003 when the authors of eDonkey created Overnet, based on Kademlia [24]. The authors of eMule, a third-party implementation of the eDonkey protocol, created another Kademlia-based DHT, called Kad. More recently, BitTorrent clients have begun using a Kademlia-based DHT to locate rendezvous points for peers downloading the same file.

2.2 Modeling and Distributions

This dissertation is concerned with empirical measurements and deriving models from them. In most cases, we will observe a series of events $X_1, X_2, X_3, \ldots, X_n$ and aim to model the events with a probability distribution described by a *cumulative distribution function* (CDF):

$$F(x) = Pr[X \leq x].$$

We will also frequently make use of the *complementary cumulative distribution function* (CCDF), given as:

$$Pr[X > x] = 1 - F(x).$$

The noted statistician George E. P. Box wrote: “All models are wrong, but some models are useful” [33]. That is to say, by definition models are imperfect approximations, and any claim that a model is perfectly accurate is highly dubious. Models are useful when they capture the most important properties of some behavior, allowing us to meaningfully describe and reason about the behavior in simplified terms.

When reasoning from a model (or using simulations), the reasoning may be sensitive to certain properties of the model. Any conclusions draw using the model are only valid if the model produces a good approximation for any sensitive properties. As an example, if an analysis depends only on the mean value of some event, then the particular distribution used in the model is of little consequence. On the other hand, if an analysis depends specifically on the events being normally distributed, but the actual events exhibit heavy skew, then the conclusions are of little value. Because the validity of a model depends on the purpose for which we use the model, it is almost always erroneous to make a blanket statement that a model is valid. As an example, Newton's mechanics are an incredibly useful model, but they do not accurately describe behavior near very large masses or when traveling near the speed of light. Einstein's general relativity more accurately describes those cases but does not describe quantum physics, and so on.

Given these considerations, a model that accurately captures many important properties is more powerful than a model that accurately captures fewer properties, because it will remain valid in a wider variety of circumstances. For a particular set of data, we can always construct a very accurate model by using a large number of model parameters. However, such a model is fragile because it is unlikely to fit as accurately if we collect more data. Also, such a model will not be simple, conflicting with one of the goals of using a model in the first place. Therefore, when selecting one model over another, we prefer one that:

- Accurately captures more important properties
- Maintains its accuracy across datasets
- Uses fewer parameters

The goals of simplicity and accuracy are sometimes in conflict and the choice of model sometimes depends on its application. General relativity accurately describes a wider ranger of behavior, but Newtonian mechanics are simpler and accurately describe a wide range of everyday behavior. However, some models are strictly better than others. No one makes use of Aristotle's laws of motions; Newton's are more accurate, explain a wider variety of data, and are no more complex.

In summary, whenever we attempt to fit a model to data, we prefer simpler models and must demonstrate that the model holds for data not used to perform the fit. If we find a useful model, we must specify what aspects of the behavior the model accurately captures.

Classes of Distributions: Statistics provides many classes of probability distributions that can describe a wide range of behavior. A class of distributions is described by a formula for the cumulative distribution function (CDF) that has one or more parameters. For example, the class of exponential distributions have the single positive parameter λ and are described by:

$$F(x) = Pr[X \leq x] = 1 - e^{-\lambda x}.$$

Johnson, Kotz, and Balakrishnan [34, 35] provide an excellent survey of useful classes of distributions and their properties. We briefly survey here the distributions most commonly used by network researchers.

The class of *exponential distributions* given above is a simple one-parameter distribution class, often used to model the time between independent events that occur at a constant rate, λ, such as the number of times you need to roll a die before rolling a 6. It is also used to model how long an entity remains in state 1 if it can change to state 2 with probability $\frac{1}{\lambda}$ per unit time, such as radioactive decay. The exponential distribution class is also called the *memoryless distribution* class because the time until the switch to state 2 is not dependent on the amount of time already spent in state 1. An exponential distribution has mean $\frac{1}{\lambda}$, median $\frac{\log 2}{\lambda}$, and variance $\frac{1}{\lambda^2}$.

The class of *subexponential distributions* are those whose right tail $1 - F(x)$ decays slower than any exponential. Commonly used subexponential distributions include *Weibull distributions* with shape parameter $0 < k < 1$ and the *Log-normal distributions*.

A *heavy-tailed* distribution is one with the following property [36]:

$$Pr[X > x] \propto x^{-k}, \text{ as } x \rightarrow \infty, \ 0 < k < 2.$$

The parameter k is called the *tail index* and is equal to the "slope" of the tail on a log-log plot. As a result, if a distribution is heavy-tailed, on a log-log plot of the CCDF the tail will appear linear with a “slope” between 0 and 2. Heavy-tailed distributions have infinite variance, and for $k < 1$ also have an infinite mean.

The *Pareto distribution* class is the most commonly used heavy-tailed distribution and is given by the CDF:

$$Pr[X \leq x] = 1 - \left(\frac{x_m}{x}\right)^k,$$

where x_m is a positive location parameter and k is a positive shape parameter. Note that a Pareto distribution is only heavy-tailed if $k < 2$, as we define it here.[2] When plotted in log-log scale, the CCDF of the Pareto distribution appears linear, originating at $(x_m, 1.0)$ with slope $-k$. The Pareto distribution is described as *scale-free*, because for any value of x, $\frac{Pr[X>jx]}{Pr[X>x]} = j^{-k}$. In other words, the ratio between x and jx is independent of the scale of x.

The *shifted Pareto distribution* class is alternative form of the Pareto with a scale parameter β instead of the location parameter x_m, and is given by the CDF:

$$Pr[X \leq x] = 1 - \left(1 + \frac{x}{\beta}\right)^{-k}.$$

Asymptotically, it’s equivalent to a regular Pareto distribution with the same k, but it frequently allows for greater flexibility for lower values of x.

The *Zipf distribution* is a related discrete distribution, defined as:

$$Pr[X = x] \propto \text{rank}(x)^{-a},$$

where $\text{rank}(x)$ is the rank of item x when all items are sorted from most common to least common. When plotted in log-log scale with $\text{rank}(x)$ on the x-axis, the Zipf distribution appears linear, much like the Pareto distribution. The Zipf distribution is often used when the X values have no direct numeric interpretation and the $\text{rank}(x)$ transformation is needed before plotting. For example, the first use of Zipf was in examining the frequency of word usage in books [37].

A *power-law* distribution is any distribution such that:

$$Pr[X > x] \propto x^{-k}, \text{ as } x \to \infty.$$

---

[2] Alternative definitions of heavy-tailed are used in some works, which include all Pareto distributions.

Thus, power-law distributions include both Pareto and Zipf distributions. Power-law distributions are also called *scaling* or *scale-free* because their sole response to conditioning is a change in scale.

Fitting: *Fitting* is the act of finding the best distribution within a class by finding the optimum parameters values to minimize error between the distribution and a set of data. Numerous statistical tests exist to validate whether data can be described by a particular distribution. These tests are called *goodness-of-fit tests*. The general idea of these tests is to compute a *test statistic* that summarizes the differences between the observed data and the distribution, then compute the probability, p, of that level of difference in light of the number of samples.[3] If p is below some predetermined threshold (typically 5%), then we can reject the distribution as a likely candidate for generating the data. If p is above the threshold, we cannot claim that the distribution generated the data; we have merely been unable to prove that the distribution did not generate the data. Not all of goodness-of-fit tests are equally useful. The ability of a test to reject a mismatched distribution is the *power* of the test. In general, the power of a test increases with the size of the data set.

One difficulty with goodness-of-fit tests is that if the test is powerful enough, it will almost always reject the distribution. Real events are the composite result of many complex interactions; while we may be able to develop a simple model that describes the general behavior, a sufficiently powerful statistical test will always reject the model due to these simplifications.

Another difficulty is that goodness-of-fit tests typically require that the variables be independently and identically-distributed (*IID*). The IID requirement means that if we collect 100 samples, the distribution sampled from must be the same for all 100 samples. In practice, this is often not the case (and even when true, it is virtually impossible to prove).

---

[3] If the data matches the distribution, the larger the number of samples, the lower the test statistic.

For these reason, goodness-of-fit tests are of limited utility for our purposes.[4] Nevertheless, the test statistic computed by goodness-of-fit tests is often useful for giving a sense of how closely a distribution matches the data. Another rather different, but very useful, statistical test is Leonard Savage's *interocular trauma test* (IOTT): plot the data in such a way as to make any important differences blindingly obvious [38].

2.3 Graph Theory

This dissertation assumes a rudimentary knowledge of graph theory and familiarity with the types of graphs commonly employed by network researchers. Informally a graph is a set of vertices connected by edges. More formally, the graph $G = (V, E)$ consists of the set of edges V and the set of edges E. Each edge is a pair of vertices. Throughout this dissertation we are concerned with *undirected graphs*, where $(A, B) \in E$ implies $(B, A) \in E$. The number of edges incident to a vertex is called the vertex's *degree*.

A natural interpretation of a peer-to-peer network is a graph, where the peers are vertices and network connection between peers (such as via TCP) are edges. Throughout this work, we tend to use *node* or *vertex* when viewing the network as a graph and especially when relying on graph theory, and *peer* in contexts when working outside the graph model (*e.g.*, peers may arrive, depart, and communicate). Likewise, the term *edge* is used in the graph context, while *connection* is used when placing more emphasis on the network aspects. The term *hop* refers to an overlay connection, particularly in the context of traversing the overlay during a search. However, because insights from graph theory are only helpful to the extent they enlighten our knowledge of network operation, we cannot separate the two completely. While the context influences the choice of words, "node", "vertex", and "peer" should be viewed as interchangeable throughout this work, as should "connection", "edge", and "hop".

---

[4] Goodness-of-fit tests are very useful in other fields where rejecting a match is very useful. For example, in medicine, if we can reject that a medication's effect matches a placebo's effect, then we have demonstrated that the medication has a statistically significant effect.

Finally, two connected peers (*i.e.*, two vertices sharing an edge) are called *neighbors* or *adjacent*.

A *random graph model* is a description of a random process for generating graphs. A *random graph* is one such graph generated by a graph model. Depending on the nature of the model, the generated random graphs will have different typical properties. As graphs are good descriptions of networks, a common technique in network research is locate an appropriate graph model whose typical properties match the empirical observations. Using the graph model allows simulation and analytical studies to draw general conclusions about graphs similar to the empirically observed graph, without being tied to particular features of the empirical graph. In particular, studying scalability requires varying the size of the graph, while maintaining its most important properties.

The earliest and simplest graph model is the Erdös–Rényi model [39]. The model has two parameters: $|V|$, the number of vertices, and p, the probability of an edge existing. To generate an Erdös–Rényi graph, we consider every possible edge and instantiate it with probability p. On average, an Erdös–Rényi graph will have $|E| = \frac{|V|(|V|-1)}{2}$ edges. It is well-known that with high probability Erdös–Rényi graphs produce a graph with a single, giant connected component for modest values of p. Additionally, the shortest-path distance between any pair of vertices in the giant component will be short ($O(\log |V|)$).

A *small-world graph* is a graph that has similar shortest-path lengths to Erdös–Rényi graphs, but greater internal structure as measured by the *clustering coefficient* [40]. The clustering coefficient of a vertex, X, is defined as:

$$\frac{|\text{edges between neighbors of } X|}{|\text{possible edges between neighbors of X}|}.$$

For example, if node A has 4 neighbors, they could have at most 6 edges between them. If only two of the neighbors are connected together, that's one edge and the clustering coefficient of A is $\frac{1}{6}$. The clustering coefficient of a graph is defined as the mean clustering coefficient of all the graph's vertices. There are several different models available for generating small-world graphs. Perhaps the most well known is the Watts–Strogatz model, which begins with a *strongly regular graph*[5] and permutes each edge with probability p. Strongly regular graphs are characterized by high clustering coefficients and long shortest-path lengths ($O(|V|)$). For very low p, the Watts–Strogatz graphs retain the properties of the strongly regular graph. For high p, most of the edges in the initial graph are randomly permuted, resulting in a graph similar to Erdös–Rényi graphs (*i.e.*, low diameter and a very low clustering coefficient). However, low to moderate values of p lead to small-world graphs, which have the clustering properties of strongly regular graphs, but the low diameter of Erdös–Rényi graphs.

A *power-law graph* or *scale-free graph* is a graph with a degree distribution that follows a power-law distribution, *i.e.*, a few vertices have very high degree while most vertices have very low degree. For comparison, the degree of vertices in Erdös–Rényi graphs is binomially distributed (asymptotically). Power-law graphs always have a low diameter (short paths are available via the high degree vertices). There are several different models for generating random power-law graphs (*e.g.*, [41, 42]), with somewhat different properties. In some models, they also have a high clustering coefficient (most vertices neighbor with high-degree vertices who neighbor with one another), making them a special case of small-world graphs. However, other models to construct a power-law graph that has a low clustering coefficient (*e.g.*, by making it a tree).

As this dissertation is an empirical study, we are primarily concerned with observing the properties of P2P network topologies in practice to reveal how well commonly employed models approximate the P2P network topologies.

---

[5] In a regular graph, every vertex has the same degree. A strongly regular graph has the additional constraint that every pair of neighboring vertices have the same number of neighbors in common, *i.e.*, every vertex has the same clustering coefficient.

CHAPTER 3

Related Work

Existing empirical studies of peer-to-peer networks can be categorized based on the measurement techniques they employ or based on the properties they examine. In this chapter, we summarize the different measurement techniques employed and discuss their strengths and weaknesses. We then turn our attention to the properties examined and summarize the primary empirical findings of prior work. Closely related work will be discussed in greater detail in later chapters, where appropriate.

3.1 Measurement Techniques

Existing empirical P2P studies employ one of five basic techniques, each offering a different view with certain advantages and disadvantages:

Passive Monitoring: Eavesdrop on P2P sessions passing through a router.

Participate: Instrument peer-to-peer software and allow it to run in its usual manner.

Crawl: Walk the peer-to-peer network, capturing information from each peer.

Probe: Select a subset of the peers in the network and probe them at regular intervals.

Centralize: Rely on logs maintained by a central server.

Table 3.1 summarizes the peer-reviewed studies in each category and lists the particular systems they examine. Studies which intercept data have typically focused on Kazaa, which was one of the most popular peer-to-peer system at the time of the studies. Saroiu*et al.* [43] show that in 2002 Kazaa traffic was between one and two orders of magnitude larger than Gnutella traffic. However, others studies tend to focus on Gnutella, which has several open source implementations available and open protocol specifications. Other popular file-sharing networks such eDonkey 2000, Overnet, and Kad remain largely unstudied. Each of the different measurement techniques has different strengths and weaknesses, explained in detail below.

Passive Monitoring: Monitoring peer-to-peer traffic at a gateway router provides useful information about dynamic peer properties such as the types and sizes of files being transferred. It also provides a limited amount of information about dynamic connectivity properties such as how long peers remain connected. However, passive monitoring suffers from three fundamental limitations.

First, because it looks at only a cross-section of network traffic, usage patterns may not be representative of the overall user populations. For example, two of the

Intercept	Participate	Crawl	Probe	Centralize
[44] (B,D,G,K)	[49] (G)	[7] (G)	[59] (N,G)	[64] (B)
[45] (K)	[50] (G)	[58] (G)	[60] (N,G)	[65] (B)
[46] (K)	[51] (G)	[6] (G)	[61] (O)	[66] (B)
[47] (K)	[52] (G)		[62] (D)	[67] (*)
[43] (K,G)	[53] (G)		[63] (S)	
[48] (K,G,*)	[54] (G)			
[3] (B,D,G,K,N,*)	[55] (B)			
	[56] (K)			
	[57] (K)			

TABLE 3.1: File sharing measurement studies, grouped by technique. The system under study is shown in parenthesis. B=BitTorrent, D=eDonkey 2000, G=Gnutella, K=Kazaa, N=Napster, S=Skype, O=Overnet, *=Miscellaneous

most detailed studies of this type [43, 45] were both conducted at the University of Washington (UW). Because the University has exceptional bandwidth capacity and includes an exceptional number of young people, their measurements may capture different usage characteristics than, for example, a typical home broadband user. This limitation may be somewhat overcome by comparing studies taken from different vantage points. One study [48] overcomes the single-viewpoint limitation by capturing data at several routers within a Tier-1 ISP.

The second limitation of passive monitoring is that it only provides information about peers that are actively sending or receiving data during the measurement window. Monitoring traffic cannot reveal any information about peers which are up but idle, and it is not possible to tell with certainty when the user has opened or closed the application. These caveats aside, passive monitoring is quite useful for providing insight in file sharing usage patterns.

The third limitation is the difficulty in classifying P2P traffic. Karagiannis *et al.* [3] show that the most common method of identifying P2P traffic, by port number, is increasingly inaccurate.

The passive monitoring technique is predominantly used to study bulk data movement such as HTTP-like file transfers and streaming, where it is relatively easy to identify a flow at it beginning and count the bytes transferred.

Participate: Instrumenting open-source clients to log information on disk for later analysis facilities the study of dynamic connectivity properties, such as the length of time connections remain open, bandwidth usage, and the frequency with which search requests are received. However, there is no guarantee that observations made at one vantage point are representative. Some studies employ multiple vantage points, but the vantage points still typically share common characteristics (*e.g.*, exceptionally high bandwidth Internet connections) and still may not be representative.

Crawl: A crawler is a program which walks a peer-to-peer network, asking each node for a list of its neighbors, similar to the way a web-spider operates. Crawling is the only technique for capturing a full snapshot of the topology, needed for graph analysis and trace-driven simulation. However, capturing the whole topology is tricky, particularly for large networks that have a rapidly changing population of millions of peers. All crawlers capture a distorted picture of the topology because the topology changes as the crawler runs.

Probe: Several studies gather data by probing a set of peers in order to study static peer properties, such as link bandwidth and shared files. By probing the set of peers at regular intervals, studies may also examine dynamic peer properties such as the session length distribution. To locate the initial set of peers, researchers have used techniques such as a partial crawl [60–63], issuing search queries for common search terms [59, 60], and instrumenting a participating peer [59]. One drawback of probing is that there is no guarantee that the initial set of peers are representative. Additionally, when studying dynamic properties, probing implicitly gathers more data from peers who are present for a larger portion of the measurement window.

Centralize: The final measurement technique is to use logs from a centralized source. Due to the decentralized nature of peer-to-peer networks, there typically is no centralized source. However, BitTorrent uses a centralized rendezvous point called a *tracker* that records peer arrives, peer departures, and limited information about their download progress.

Summary: Existing measurement techniques for gathering data about the operation of peer-to-peer systems, summarized in Table 3.2, are limited in their accuracy and introduce unknown, and potentially large, amounts of bias. The only high-fidelity technique is relying on centralized logs, which is of limited utility. Chapters 5 and 4 introduce new tools for gathering highly accurate measurements.

3.2 Measurement Results

The following subsections summarize other empirical studies of peer-to-peer systems, discuss their main findings, and identify important areas which remain unstudied.

Technique	Advantages	Disadvantages
Passive monitoring	Provides information about traffic	May be biased Omits idle peers Omits traffic on non-standard ports
Participate	Provides information about dynamic connectivity	May be biased
Crawl	Captures the entire topology Unbiased	Doesn't scale May have significant distortion
Probe	Captures peer properties	May be biased Dynamic properties inherently biased
Centralize	Unbiased	Only available if system has a centralized component

TABLE 3.2: Summary of existing measurement techniques

3.2.1 Static Peer Properties

Saroiu, Gummadi, and Gribble provide an extensive and informative study, primarily of static peer properties [60]. While earlier work conceived of peers as equal participants, their landmark study demonstrates that in practice not all peers contribute equally to peer-to-peer systems. Using data collected from Gnutella and Napster in May 2001, their observations show a heavy skew in the distributions of bottleneck bandwidth, latency, availability, and the number of shared files for each host, with each of these qualities varying by many orders of magnitude.

Additionally, they found correlations between several of the properties. Bottleneck bandwidth and the number of uploads have a positive correlation, while bottleneck bandwidth and the number of downloads have a negative correlation. In other words, peers with high bandwidth tend to be uploading many files, while peers with low bandwidth have to spend more time downloading. Interestingly, no significant correlation exists between bottleneck bandwidth and the number of files stored on a peer.

In addition to the sweeping work of Saroiu *et al.* [60], several studies focus on examining the files shared by peers [54, 59, 62]. A few results have consistently appeared in these studies. First, peers vary dramatically in the number of files that they share, with a relatively small percentage of peers offering the majority of available files. In addition, a large fraction of peers share no files at all (25% in [60], two-thirds in [54, 62]). Second, the popularity of stored files in file-sharing systems is heavily skewed; a few files are enormously popular, while for most files only a few peers have a copy. Fessant *et al.* [62] found that it may be described by a Zipf distribution. However, Chu, Labonte, and Levine [59] found that the most popular files were relatively equal in popularity, although less popular files still had a Zipf-like distribution.

Studies also agree that the vast majority of files and bytes shared are in audio or video files, leading to the distribution of file sizes exhibiting a multi-modal behavior. Each studies shows that a plurality of files are audio files (48% in [62], 76% in [59]).

However, video files make up a disproportionately large portion of the bytes stored by peers (67% in [62], 21% in [59]).

Fessant *et al.* [62] took the additional step of examining correlations in the files shared by peers. Their results show that users have noticeable interests, with 30% of files having a correlation of at least 60% with at least one other file. Of peers with at least 10 files in common, they found that 80% have at least one more file in common. Likewise, of peers with at least 50 files in common, in their data nearly 100% have at least one more file in common.

Adar and Huberman [54] explore the relationship between sharing many files and responding to many queries, showing that there is no clear correlation. In other words, if a peer has many files, it may or may not have highly popular items; it may have a large collection of rarely-sought files while other peers have small collections of highly popular items.

3.2.2 Dynamic Peer Properties

Most dynamic peer properties are tied to how long and how frequently peers are active. *Session length* is the length of time a peer is continuously connected to a given peer-to-peer network, from when it arrives until it departs. *Uptime* is the length of time a peer that is still present has been connected. *Remaining uptime* is how much longer until an active peer departs. *Lifetime* is the duration from the first time a peer connects to a peer-to-peer network—ever—to the very last time it disconnects. *Availability* is the percentage of time that a peer and its resources are connected to the peer-to-peer network within some window. *Downtime* is the duration between two successive sessions. Finally, an *inter-arrival interval* is the duration from the arrival of one peer until the arrival of the next peer. The session length, uptime, and remaining uptime are closely related, as shown in Figure 3.1. The popularity of file transfers is another dynamic peer property, which we examine separately.

Generally, the most important distribution for simulation and analysis is the session-length distribution, as it fully determines the uptime and remaining uptime distributions and strongly influences the availability. The median session length spec-

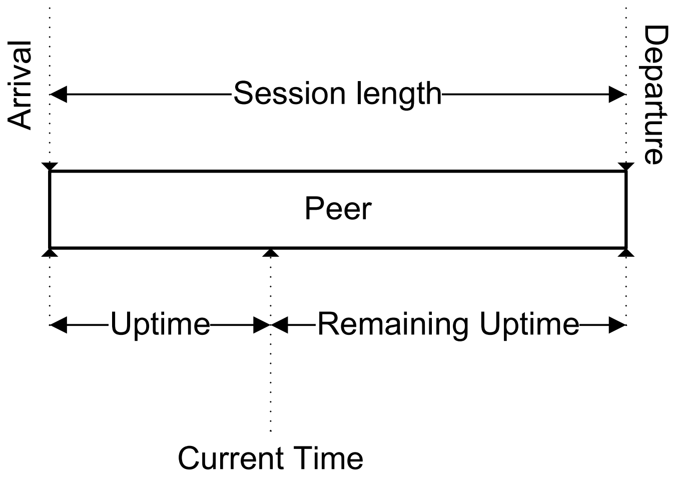


FIGURE 3.1: Illustration of the relationship between session length, uptime, and remaining uptime

ifies how much churn a protocol must cope with, and the shape of the distribution determines whether some peers are dramatically more stable than others.

Due to its importance, several studies examine the session length distribution. Rhea, Geels, and Kubiatowicz [5] summarize these studies, as shown in Table 3.3 which is adapted from their paper. The results for the median session time are conflicting, ranging from as low as one minute to as long as one hour. This may be due to genuine differences in user behavior, or it may be due differing methodology.

While the median differs dramatically, all the studies agree that the session lengths are heavily skewed: many sessions are short, while some session are very long. Several studies draw the conclusion that session lengths can be modeled with a power-law (or Pareto) distribution [63, 68, 69] or exhibit heavy-tailed behavior [45, 48, 63]. Chu,

Citation	Systems Observed	Session Time
[60]	Gnutella, Napster	50% $\leq$ 60 min.
[59]	Gnutella, Napster	31% $\leq$ 10 min.
[48]	Kazaa	50% $\leq$ 1 min.
[61]	Overnet	50% $\leq$ 60 min.
[45]	Kazaa	50% $\leq$ 2.4 min.

TABLE 3.3: Observed session lengths in various peer-to-peer file sharing systems. Adapted from [5].

Labonte, and Levine [59] fit session lengths to a log-quadratic distribution, which can be viewed as a second-order variation of the Pareto distribution. Only one of the studies [68] provide an analysis and fit to support their conclusion. However, Leonard, Rai, and Loguinov [70, pg. 8] suggest that the fit given in [68] seems implausible and point out some possible methodological errors.

In BitTorrent, the availability of high-quality tracker logs facilitates the study of peer dynamics. Prior studies of BitTorrent [64, 65] show that session lengths are heavily skewed. However, they do not attempt to create a model based on the data. A surprising discovery shown by Izal *et al.* [64] is that many peers (81% in their trace) depart before downloading the entire file, while peers who do complete the download linger for more than six hours on average.

While most studies of peer dynamics focus on session length, Bhagwan, Savage, and Voelker [61] provide a study of peer availability in Overnet during January 2003. However, they find that the distribution of availability varies dramatically with the size of the measurement window, due to the significant fraction of hosts who appear briefly and only once.

Files Transfers: Another class of dynamic peer properties are related to the files that peers are actively transferring, which in some sense is the derivative of the files being stored on each peer (discussed above under Static Peer Properties). The properties of files being transferred are most often studied using passive monitoring at gateway routers. Two of the most detailed studies of files being transfered [43, 45] were both conducted at the University of Washington (UW). The first study, [43], focuses on comparing HTTP requests with P2P requests, demonstrating that P2P uses more than twice as much bandwidth as the web on their network. Although they found a smaller number of hosts are involved in the P2P traffic, each object is orders of magnitude larger. Furthermore, they show that a majority of the P2P traffic came from a few large video files. Their second study, [45], more closely examines the popularity and properties of different P2P objects. The popularity of different objects did not match a Zipf distribution, in contrast to the Zipf distribution of popularity observed for Web objects. The authors suggest this may be due to the fact that in

P2P systems, users typically download an object at most once, while Web users may return to a website many times. Instead, they found that unpopular objects appear Zipf-like, while popular objects are relatively equal in popularity, matching the results for stored files seen in [59].

Leibowitz *et al.* [46, 47] provide measurements from an Israeli Internet Service Provider (ISP), and compare their findings with the UW studies. Interestingly, they find that while the UW is an overall provider of P2P content, the ISP they study is an overall consumer of P2P content. Their studies give particular focus to the idea of caching P2P content. In [46], they implement a transparent 300 GB cache yielding a 67% bandwidth savings.

3.2.3 Static Connectivity Properties

In 2000, a company called "Clip2" developed a Gnutella crawler and published their results on the web. Although not validated by peer-review, their analysis and topology captures have been widely used in simulation studies of improvements for Gnutella-like networks [8, 71–74]. In [7], Clip2 presents analysis of snapshots they captured between June and August of 2000. Using their crawler which took a bit less than an hour to survey the entire topology, they gathered snapshots containing between 1,000 and 8,000 peers.

Their work suggests that the Gnutella network has a power-law degree distribution, based on plotting the degree distribution of their snapshots on a log-log scale and demonstrating a linear fit. Adamic *et al.* [8] repeat this analysis on similar data provided by Clip2. However, neither study considers alternative models of the degree distribution. Several later studies [71, 75–78] rely on the power-law model, simulating Gnutella using random power-law topologies. Lv *et al.* [9] show that power-law networks exhibit poorer performance than other types of random graphs.

Ripeanu, Foster, and Iamnitchi [6] implemented a crawler and use it to examine properties of the Gnutella overlay topology. Their crawler uses a client-server architecture running on roughly 50 computers to crawl a 30,000 node network in a few hours. Their crawls were conducted in November 2000 through May 2001. The size of the network grew from 2,063 to 48,195 peers over that time. They performed all-pairs shortest-path computations and plotted the distribution of path lengths. 95% of shortest-paths are 7 hops or less, with most shortest-paths being 4 or 5 hops long. They repeat the analysis of [7] and [8] by plotting the degree distribution in log-log scale. In their November snapshot, the degree distribution appears linear on the log-log plot, suggestion a power-law distribution. Their March snapshot is different. Low-degree nodes are approximately equally common, though among high-degree nodes the distribution still appears linear on the log-log plot. They also show that the Gnutella topology is poorly matched to the underlying Internet topology.

None of these studies quantify the accuracy of their crawlers, making it difficult to determine how accurate the captured topologies are. Given that their crawls take a few hours, and peer uptimes may be just a few minutes [45, 48, 59], it is very possible that these topologies are highly inaccurate, leading to a drastically distorted picture of the network. Also, all of the captured snapshots are relatively small, making it difficult to study scalability.

3.2.4 Dynamic Connectivity Properties

No prior studies have examined the effects of dynamics on topological structure nor properties of structured P2P systems. However, existing work examines different bootstrapping mechanisms, flow control mechanisms, and characteristics of queries.

Bootstrapping: When a user starts their P2P application, the application must discover other peers to form connections with. This initial discovery process is called *bootstrapping*. Karbhari *et al.* [50] provide a comparative study of the bootstrapping mechanisms in LimeWire 2.8, Gtk-Gnutella 0.91, Mutella 9.4.3, and Gnucleus 1.8.6.0. Each implementation employs a local cache of nodes discovered in previous sessions and resorts to contacting special rendezvous points, called GWebCaches [79], if the local cache proves unhelpful. However, the details of each implementation differs substantially. LimeWire's performance is the best; it differentiates between ultrapeers and leaf peers in its cache, prioritizes the cache by age, and includes the hard-coded addresses of two orders of magnitude more GWebCaches than the other implementations. Not only does LimeWire connect faster, but it begins receiving query replies sooner once connected. Unfortunately, GWebCaches are not performing well; most of the load is concentrated on just a few and the caches have many entries for hosts that are no longer present in the system.

Flow Control: Qi He and Mostafa Ammar [49] bring to light the issue of congestion in the overlay and compares how different applications cope with the problem. Each peer connection uses TCP, a reliable transport protocol with congestion control. When an overlay connection is congested, TCP will slow down the transfer rate. In turn, the application must slow the rate it feeds data to TCP, eventually leading to dropping packets at the application layer.[1] Their study examines the approaches taken by three different Gnutella implementations: LimeWire, Mutella, and Gtk-Gnutella. Their study includes a comprehensive explanation of the algorithms employed by these implementations, measurements comparing their performance, and simulations exploring alternative algorithms. LimeWire shows the best overall performance, using a flow-control scheme that prevents neighbors from sending packets that a peer will not have the capacity to forward.

Search Characteristics: Sripanidkulchai presented one of the first studies of search terms in a P2P network [53], demonstrating that queries follow a Zipf distribution, except for the most popular queries which are of roughly equal popularity (similar to the distributions of files stored and file transfers). The fact that popular queries are much more common than unpopular queries suggests caching query results may be beneficial [52, 53].

Klemm *et al.* [51] provide a comprehensive analysis of queries, breaking down the number of queries observed by time of day and geographical region. It includes distributions for the number of sessions that generate queries, the time until the first queries, the query inter-arrival time, and the length of the session. In short, it

---

[1] Application-layer buffers can make dropping packets less frequent, but at the cost of increasing latency through that peer. If a peer continually receives more packets than it can forward, it must eventually drop some packets because the peer cannot have an infinitely large buffer.

provides a framework for generating a synthetic query workload as seen from a single peer.

Annexstein, Berman, and Jovanovic [58] point out an interesting quirk of flooding over a real network, that does not occur in a simple graph model. In simulation and analysis of scoped flooding networks such as Gnutella, it is normally assumed that all hops take equal time to traverse. A message spreads out through the network, and any duplicates along longer paths are dropped. However, due to unequal latencies, it is possible for the message along the "longer" overlay path to arrive first. When this occurs, the message will reach fewer peers, since the message arrives with a lower TTL than in the ideal model. They call this behavior the *short-circuit effect*. In simulation, they show that for a TTL of 5, it reduces the search horizon by 58% on average.

3.3 Summary

Peer-to-peer systems have been a popular topic for empirical studies. However, these measurement studies have been rather ad-hoc, gathering data in the most convenient manner without critically examining their methodology for measurement bias. While an ad-hoc approach is often suitable for first-order approximations (*e.g.*, "file popularity is heavily skewed"), it is generally not appropriate for making precise conclusions (*e.g.*, "session-lengths are power-law"). One of the largest gaps in the prior work is the development and validation of high-fidelity measurement tools for peer-to-peer networks.

Chapters 5 and 4 introduce new tools for gathering highly accurate measurements. While an argument could be made for repeating prior studies with more accurate tools, this dissertation focuses on relatively unstudied areas. Existing studies cover properties of stored files, file transfers, and search terms in great detail. Additionally, Saroiu, Gummadi, and Gribble [60] provide a comprehensive study of static peer properties. Chapter 6 reexamines several static peer properties. While several studies have touched on peer dynamics, they offer conflicting reports. Chapter 7 explores peer dynamics in detail.

Turning to connectivity properties, Chapter 8 examines two-tier topologies, which have not previously been studied empirically. Chapter 9 focuses on dynamic connectivity properties, such as how peer dynamics influence overlay structure and examining search behavior over a widely deployed DHT.

CHAPTER 4

Capturing Global State

Portions © 2005 IEEE. Reprinted, with permission, from D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the Gnutella network," in *Proc. Global Internet Symposium*, Miami, FL, Mar. 2005, pp. 127–132.

The global state of a peer-to-peer system is distributed among all the peers. Exploring the graph and capturing the state of each peer captures the global state. A *crawler* is a tool that captures global state in this way. Since the system changes as the crawler explores, the picture is *distorted*, much like a photograph capturing rapid motion. Therefore, crawl speed is important. The faster the crawler runs, the less distortion. Furthermore, the duration of the crawl determines the granularity of using back-to-back snapshots to explore dynamic properties. In other words, we cannot explore how the system has changed over a period of 10 minutes if it takes an hour to capture a snapshot. Crawlers have most often been used for capturing snapshots of the overlay topology as a graph, needed for studying many static connectivity properties. Prior studies [6, 7, 57, 58] take 30 to 120 minutes to crawl the network and capture a snapshot of the graph. However, many peers are not even present for that long [5, 45, 48]. This suggests existing crawlers are much too slow and may be capturing very distorted snapshots. The accuracy of these snapshots most likely has not been previously addressed because it is challenging to measure the distortion without a perfect reference snapshot for comparison.

This chapter documents a fast crawler, called *Cruiser*, that can capture complete topologies in just a few minutes, introduces techniques for assessing the accuracy of snapshots, and shows that Cruiser may be used to capture accurate snapshots. We focus on capturing snapshots of the Gnutella topology, as Gnutella is one of the largest P2P systems and has been the target of most prior P2P crawlers, allowing us to make meaningful comparisons. Although we focus on Gnutella, Cruiser uses a plug-in architecture, allowing it to crawl other P2P systems with the addition of an appropriate plug-in.

In order to crawl quickly, the design of Cruiser must overcome several challenges:

- It must make heavy use of parallelism to contact many peers simultaneously. Managing so many connections in parallel can lead to CPU bottlenecks requiring a distributed architecture.
- If the load is too great, Cruiser may lose data.[1] Therefore, Cruiser must carefully control the load by appropriately limiting the number of connections. Since each connection uses a variable amount of resources, this limit must be dynamic.
- Cruiser cannot afford to wait the minutes for a TCP connection attempt to timeout. Instead, the proper trade-off between timing out too quickly (which increases distortion by losing data) and timing out too slowly (which increases distortion by making the crawl slower) must be found empirically.

Material in this chapter was adapted from material under submission to a journal, which includes material previously presented at conferences and workshops [10–13]. The material was co-authored with Prof. Reza Rejaie and Dr. Subhabrata Sen. The experimental work is entirely mine. The writing is primarily mine, with contributions by Reza Rejaie, who also provided technical guidance. Subhabrata Sen provided additional editorial assistance.

---

[1] For example, connections may appear to time out if the CPU load is so great that received packets spend too long in a queue before being processed.

4.1 The Design of Cruiser

Our primary goal in the design of Cruiser is to minimize distortion in captured snapshots by maximizing the speed at which Cruiser explores the overlay topology. We employ several techniques and features to achieve this design goal, as described below.

Two-Tier Networks: Cruiser leverages the two-tier structure of modern P2P networks by only crawling ultrapeers. Since each leaf must be connected to an ultrapeer, this approach enables us to capture all the nodes and links of the overlay by contacting a relatively small fraction of all peers. This strategy leads to a major reduction (around 85%) in the duration of a crawl without any loss of information.

Distributed Architecture: Cruiser employs a master-slave architecture in order to achieve a high degree of concurrency and to effectively utilize available resources on multiple computers. A master process coordinates multiple slave processes that act as independent crawlers and crawl disjoint portions of the network in parallel. The slaves communicate with the master using loose synchronization as follows. Each slave has an independent 2000-element queue of addresses to contact, which the master fills. Each slave drains its queue by querying peers for their neighbors and reporting back with the data they've gathered. The master extracts new addresses from the data and uses this to fill the queues. The crawl terminates when all the queues are empty.

Asynchronous Communications: Each slave process crawls hundreds of peers in parallel using asynchronous communications. Cruiser implements an adaptive load management mechanism to ensure that slave processes remain busy but do not become overwhelmed. This is important for the steady progress of the crawl especially when different slave nodes have heterogeneous processing capabilities. Toward this end, each slave monitors its CPU load and adjusts its maximum number of parallels connections using an additive-increase multiplicative-decrease (AIMD) algorithm similar to TCP's congestion control mechanism. In practice, each PC typically runs with close to 1,000 parallel connections, contributing an additional speed-up of nearly three orders of magnitude, compared to using synchronous communications (as in [6]).

Each slave maintains a parameter, call *max_concur*, that limits the maximum number of open connections, similar to a TCP congestion window. No new connections will be attempted if doing so would exceed this threshold. Because there is a high delay between opening connections and the increase in CPU load,[2] *max_concur* should not be adjusted too frequently. To measure CPU load, each slave sets up a timer to fire every half-second. A timer more than 50% late is interpreted as a signal of high CPU load: *max_concur* is multiplicatively decreased to 90% of its value and is not changed until the timer is on time. When the timer is on time, *max_concur* is linearly increased by one. Similar to TCP Slow Start, *max_concur* is multiplicatively increased by 1.2 during the initial phase of crawling to quickly reach an appropriate value. These parameters were set empirically through experimentation with our own systems. A similar adaptation mechanism could be incorporated to adjust the total number of parallel connections in order to avoid congestion on the access link of the crawler, though thus far this has not been a bottleneck for our systems.

We have experienced other system issues in the development of Cruiser that are worth mentioning. In particular, we needed to increase the limit on the number of open file descriptors per process on each Linux box. Otherwise, many connection attempts return immediately with an automatic "Connection Refused" error. In a similar vein, we increased the number of connections that our lab firewall could track to prevent the firewall from dropping packets due to this constraint.

Appropriate Timeouts: When peers are unresponsive, waiting for TCP to timeout and give up attempting to connect takes a long time. On our systems, a full TCP timeout to an unresponsive address takes more than 3 minutes. While this is suitable for many interactive and automated applications, one of our primary design goals it to make crawls as quick as possible. We conducted an evaluation of the cost-versus-benefit tradeoff of different timeout values for crawling. As a function of the timeout

---

[2] This delay is the time from when the crawler sends the first TCP SYN packet until the connection is established and data is returned by the peer, *i.e.*, at least two round-trip times.

length, Figure 4.1 shows the duration of the crawl and the percentage of peers that were unreachable. We see that while very low timeouts (less than 10 seconds) result in a dramatic increase in the number of unreachable peers, there are diminishing returns for using longer timeout values, while the crawl length (and thus distortion) continues to increase. In other words, if a peer has not responded after 10 seconds, it is unlikely to ever respond. Therefore, we use a timeout of 10 seconds, providing an additional speedup of more than a factor of two, compared to using full TCP timeouts.

4.2 Effect of Unreachable Peers

During the crawl, some peers may not be reachable by the crawler due to a variety of temporary (route failure, network congestion) or permanent (departed, firewalled) causes. In this section, we carefully examine the effect of unreachable peers on the accuracy of captured snapshots. Unreachable ultrapeers can introduce the following errors in a captured snapshot: *(i)* including unreachable peers that departed (which should not be included), *(ii)* missing links between unreachable ultrapeers and their

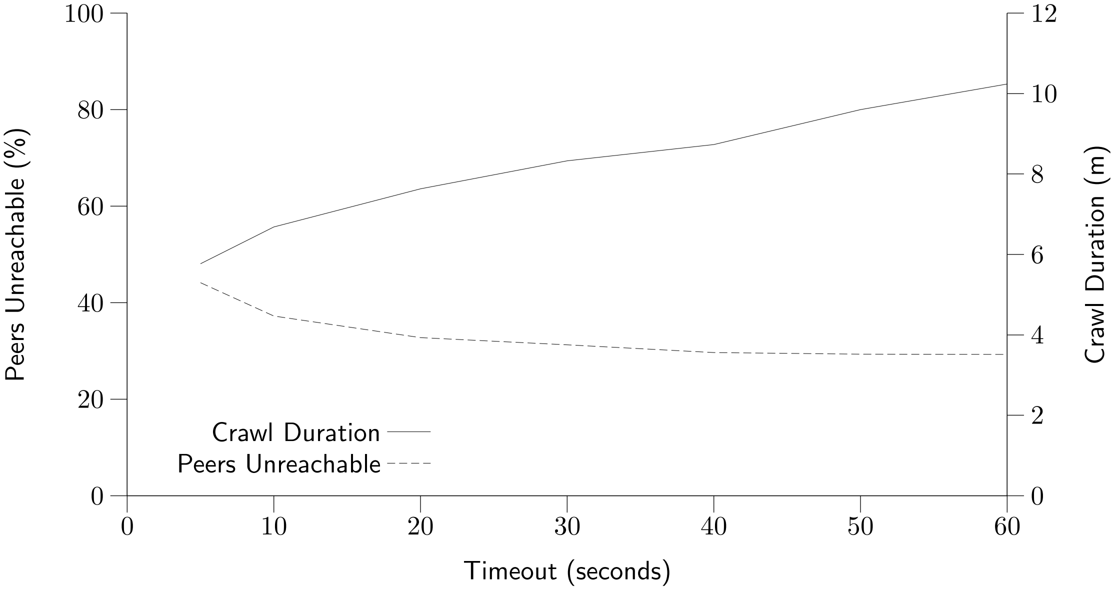


FIGURE 4.1: Effects of the timeout length on crawl duration and snapshot completeness

leaves (which should be included), and *(iii)* missing links between two unreachable ultrapeers (which should be included).

Interestingly, our measurements revealed that some of the unreachable peers are actually overwhelmed ultrapeers that sporadically accept TCP connections and can be contacted after several attempts. This transport-layer refusal means that the application is not able to call *accept()* sufficiently fast, leading to a TCP listen buffer overflow. We also noticed that connections to most of these overwhelmed ultrapeers exhibit long RTT ($>$ 1sec) and little to no loss. Since latency due to a long queue in a router is typically accompanied by packet loss, this suggests the peer's CPU may be the bottleneck and the operating system is taking a long time to process the packets. We briefly experimented with a multiple-attempt strategy, but ultimately dropped it from Cruiser for two reasons: *(i)* it only marginally increased the number of reachable peers at the cost of significantly increasing the duration of each crawl which in turn increases distortion in captured snapshots, and *(ii)* it may exacerbate the existing problem.

It is important to determine what portion of unreachable peers are departed versus firewalled or overloaded, because each group introduces a different error on the snapshot. However, there is no reliable test to distinguish the three cases, because firewalls can time out or refuse connections depending on their configuration. Previous studies assume that these unreachable peers are either departed or firewalled and exclude them from their snapshots.

To determine the status of unreachable peers, we devise the following technique to identify the fraction of peers that are unreachable because they have departed. We perform back-to-back crawls to capture two snapshots. We can then conclude that the unreachable peers in the first snapshot that are missing entirely from the second snapshot have departed. This approach reveals that departed peers constitute only 2–3% of all peers in each snapshot.

Finally, we examine those unreachable peers that time out. Since overwhelmed ultrapeers will refuse connections, we hypothesized that this group of peers is firewalled. To verify this hypothesis, we randomly selected 1000 (about 3% of) peers that were unreachable due to time out, and re-contacted them every 5 minutes.[3] Interestingly, more than 92% of these peers were never reachable at all. This implies that timeout is a good indicator for firewalled peers. *In summary, our investigation reveals that in each crawl, 30%–38% of discovered peers are unreachable. In this group, the breakdown is as follows: 2%–3% departed, 15%–24% firewalled, and the remaining unreachable peers (3%–21%) are either also firewalled or overwhelmed ultrapeers.* Since Cruiser only needs to contact *either* end of an edge, it is able to discover at least 85%–91% of edges. Since firewalled peers cannot directly connect together (*i.e.*, cannot be located at both ends of a missing edge) and they constitute more than half of the unreachable peers, the actual portion of missing edges is considerably smaller.

4.3 Other Practical Considerations

We encountered several other technical difficulties:

4.3.1 Exhausting File Descriptors

Unix-like operating systems limit the number of file descriptors each process can concurrently use. A "hard" limit, set by the system administrator, absolutely prevents the user's process from gaining more descriptors. A "soft" limit imposes a lower default setting which the user can raise, up to the hard limit, with the shell command `ulimit -n hard`. A common default soft limit is 256 file descriptors, which is often smaller than the level of parallelism Cruiser will attempt to employ. Additionally, because many TCP connections linger for a few minutes in the `TIME_WAIT` state [81], a crawler may have many more file descriptors in use than the level of parallelism seen by Cruiser.

When software attempts to open more file descriptors than the limit, the open operation fails. If the crawler software isn't careful, this can easily be confused with

---

[3] Note that each attempt translates into several attempts by TCP to establish a connection by sending SYN packets.

failure for another reason (such as a connection refusal from the computer at the target address). Because these "connections" consume few resources and complete quickly, Cruiser may incorrectly conclude that it can increase the level of parallelism, exacerbating the problem.

To avoid this problem, whenever Cruiser is about to open a new connection, it first attempts to open `/dev/null`. If the attempt fails, Cruiser postpones opening the new connection. If the attempt succeeds, Cruiser closes the file descriptor on `/dev/null` and opens the new connection. Because the process is single-threaded, there is no danger of another thread taking the available file descriptor after `/dev/null` is closed.

In our use of Cruiser, we work with our system administrators to set the hard ulimit to a high value, preferably several thousand. Cruiser executes `ulimit -n hard` during its start-up to make full use of the hard limit value.

4.3.2 Filling Firewall/NAT Tables

Connection-tracking firewalls and NAT devices also impose a limit on the number of concurrent connections. These devices track every flow passing through them. If a crawler opens more concurrent flows than the device can track, the extra flows won't reach their destination. Some devices may generate a connection failure message while others will simply drop the packets and allow the connection to time out.

We encountered this problem in our initial experiments with Cruiser when it was running behind a Linux-based connection-tracking firewall. By increasing the number of flows the firewall can handle,[4] we eliminated the problem.

4.3.3 Oversensitive Intrusion Detection Systems

Because a P2P crawler necessarily opens connections to a large number of addresses on a variety of ports,[5] we sometimes receive inquires from system administrators with intrusion detection systems that report our machines as probing their

---

[4] via `/proc/sys/net/ipv4/netfilter/ip_conntrack_max`

[5] Many peers do not run on a default port [3].

systems. Frequently, this is an indication that one of their users is running P2P behind their firewall. In other cases, it occurs when a user with a dynamic IP address assignment receives an address recently used by a P2P user. In any case, it's essential for a P2P crawler operator to contact their system support staff about the crawler and its purpose before running it. Otherwise, they may easily mistake the crawler process for a port-scanning worm.

4.4 Quantifying Snapshot Accuracy

One obvious metric to evaluate the performance of Cruiser is the time it takes to perform a crawl. However, the crawl duration doesn't reveal how accurate the crawl is; it only informs us if the crawl is more accurate than another crawl performed under similar conditions. Snapshot accuracy can not be directly measured since there is no reference snapshot for comparison. Therefore, we indirectly quantify the effect of crawling speed and duration on two dimensions of snapshot accuracy: completeness and distortion.

Impact of Crawling Speed: To examine the impact of crawling speed on the accuracy of captured snapshots, we adjust the crawling speed (and thus the crawl duration) of Cruiser by changing the number of parallel connections that each slave process can open. Using this technique, Cruiser can effectively emulate the behavior of previously reported crawlers which have a lower degree of concurrency.

We introduce the following two metrics for evaluating a crawler. The first metric, *edge distortion*, examines the edges in the captured snapshot. For each contacted peer A, with neighbors N_A, we examine each of its neighbors $B \in N_A$ to see if they likewise reported A as their neighbor. If not, we have an inconsistency in the graph caused by the fact that the edge changed sometime between crawling node A and crawling node B. The edge distortion, then, is the fraction of edges that are inconsistent.

The second metric, *node distortion*, examines the peers present in two consecutive snapshots captured back-to-back. We denote the peers as the sets V_1 and V_2. Comparing these two back-to-back snapshots provides insight into how distorted our picture of the network is. If Cruiser were instantly fast and captured perfect snapshots, V_1 and V_2 would be identical. The greater the change that occurs while Cruiser runs, the greater the difference between V_1 and V_2. We define the node distortion as $\frac{|V_1 \Delta V_2|}{|V_1|+|V_2|}$, where $V_1 \Delta V_2$ is the symmetric difference of V_1 and V_2 (*i.e.*, peers in one set or the other, but not both). Note that when $V_1 = V_2$, the node distortion is 0%, and when V_1 and V_2 are completely disjoint the node distortion is 100%.

Figure 4.2 depicts peer and edge distortion as a function of crawl duration. This figure demonstrates that the accuracy of snapshots decreases with the duration of the crawl, because the increased distortion reflects changes in the topology that occur *while the crawler is running.* Crawlers that take 1–2 hours (comparable to those in earlier works) have a peer distortion of 9%–15% and an edge distortion of 31%–48%, while at full speed Cruiser exhibits a peer distortion of only 4% and an edge distortion of only 13%.

Completeness of Snapshots: To examine the completeness of snapshots captured by Cruiser, we keep track of the following variables during each crawl: the number of discovered top-level peers, the number of leaves, the number of links between ultrapeers, and the number of links to leaves. Figure 4.3 presents variations of these

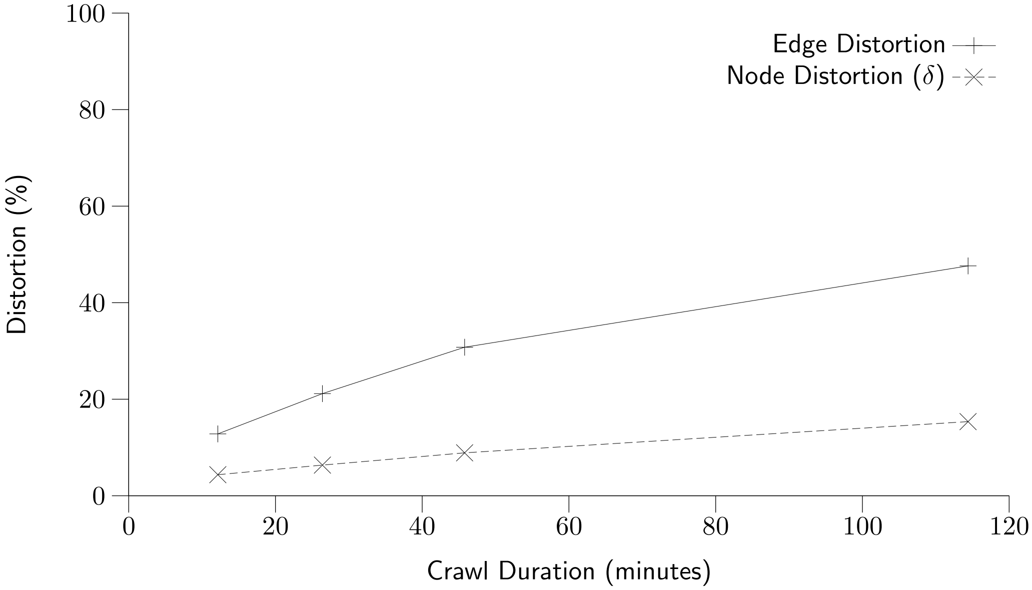


FIGURE 4.2: Effect of crawl speed on the accuracy of captured snapshots

four variables as a function of the number of top-level contacted peers in a sample crawl. Note that the number of discovered top-level peers as well as leaves curve off which is evidence that Cruiser has captured nearly all the participating peers. Links between top-level peers somewhat curves off. Finally, links to leaves is necessarily linearly increasing with the number of top-level peers because each top-level peers provide a unique set of links between itself and its leaves.

Completeness-Duration Tradeoff: To examine the completeness-duration tradeoff for captured snapshots, we modified Cruiser to stop the crawl after a specified period. Then, we performed two back-to-back crawls of the same duration and repeated this process for different durations. Figure 4.4 demonstrates the completeness-duration tradeoff. During short crawls (on the left side of the graph), node distortion (δ) is high because the captured snapshot is incomplete, and each crawl captures a different subset. As the duration of the crawl increases, δ decreases which indicates that the captured snapshot becomes more complete. Increasing the crawl length beyond four minutes does not decrease δ any further, and achieves only a marginal increase in the number of discovered peers (*i.e.*, completeness). This figure reveals a

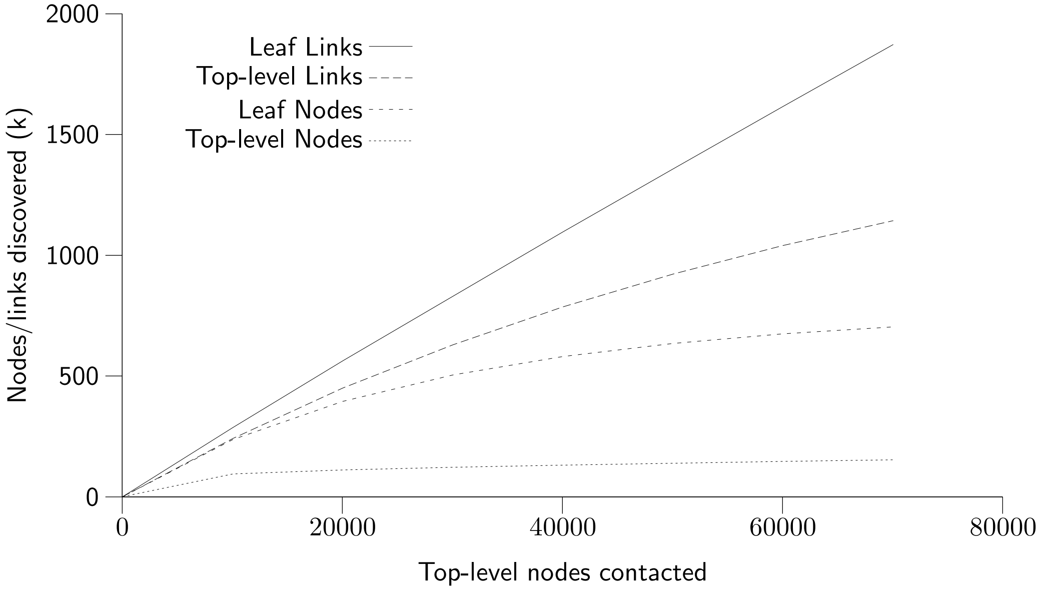


FIGURE 4.3: Cumulative dicovered information about overlay nodes and links as a function of number of contacted peers

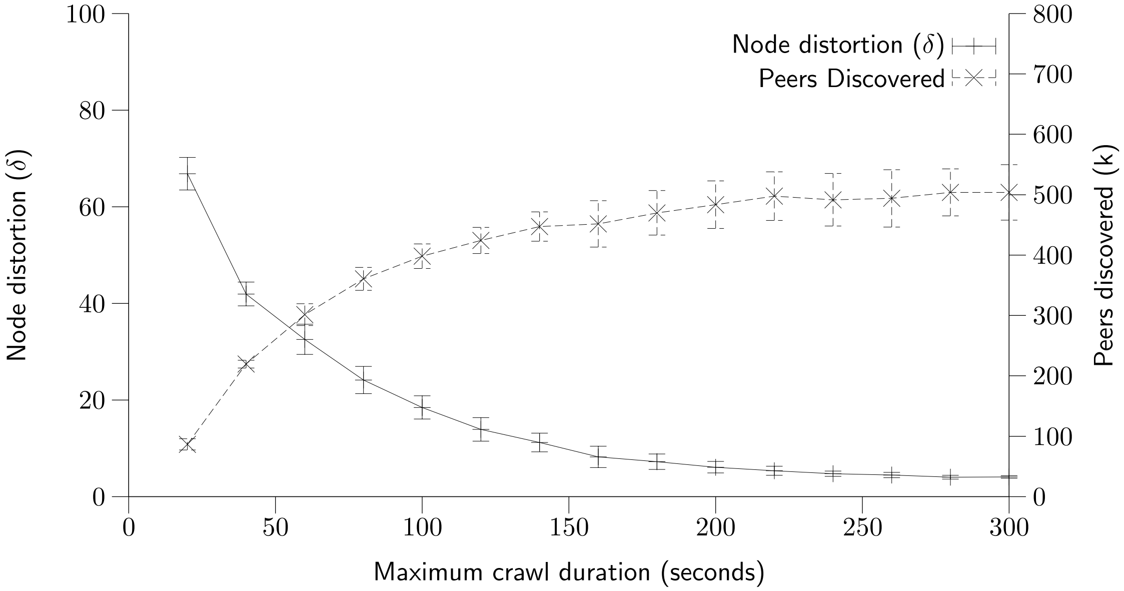


FIGURE 4.4: Error as a function of maximum crawl duration, from running two crawls back-to-back for each x-value and computing δ. Averaged over 8 runs with standard deviation shown.

few important points. First, there exists a "sweet spot" for crawl duration beyond which crawling has diminishing returns if the goal is simply to capture the population. Second, for sufficiently long crawls, Cruiser can capture a relatively un-stretched snapshot. Third, the change of $\delta = 4\%$ is an upper-bound on the distortion due to the passage of time as Cruiser runs. The relatively flat delta on the right suggest that around 4% of the network is unstable and turns over quickly.

In summary, our evaluations reveal that *(i)* Cruiser captures nearly all ultrapeers and the pair-wise connections between them and the majority of connections to leaves; *(ii)* both node distortion and edge distortion in captured snapshots increases linearly with the crawl duration; and *(iii)* snapshots captured by Cruiser have little distortion. In particular, we found that two back-to-back snapshots of unlimited duration differed only 4% in their peer populations.

4.5 Summary

This chapter presents Cruiser, a fast crawler for two-tier peer-to-peer systems such as Gnutella. We present the different techniques used in Cruiser to achieve its high speed, including leveraging the two-tier structure, a distributed architecture, asynchronous communications, and choosing appropriate timeout values. We also present techniques for quantifying the measurement inaccuracy introduced by crawl speed and present evidence that the error in Cruiser's snapshots is reasonably small.

Cruiser can capture the Gnutella network with one million peers in around 7 minutes using six off-the-shelf 1 GHz GNU/Linux boxes in our lab. Cruiser's crawling speed is about 140k peers/minute which is orders of magnitude faster than previously reported crawlers (*i.e.*, 2 hours for 30K peers (250/minute) in [6], and 2 minutes for 5K peer (2.5k/minute) in [60]).

Cruiser is a useful tool for capturing global snapshots of system state. Chapters 8 and 9 rely heavily on snapshots of the Gnutella overlay topology captured with Cruiser. While fast, Cruiser unavoidably takes $O(|V|)$ time, which means it may still be too slow to capture accurate snapshots of system with a very large population (V) or when the per-peer state is time-consuming to collect. For such cases, we need a mechanism to collect unbiased samples, which is the topic of the next chapter.

CHAPTER 5

Capturing Samples

Portions © 2006 IEEE. Reprinted, with permission, from D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "Sampling techniques for large, dynamic graphs," in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

While capturing global state is certainly ideal, sometimes it is just not practical. For very large systems, it is often prohibitively expensive to purchase the bandwidth and computers needed to capture quick, accurate snapshots. The crawl is even more expensive when capturing additional information about each peer, beyond its neighbor information. For example, we may want to know the geographic distribution across peers, or learn about the most popular files being shared. Sampling is a natural approach for learning about these systems using light-weight data collection.

A basic objective is to devise an unbiased sampling method, *i.e.*, one which selects any of the present peers with equal probability. Fulfilling this objective becomes highly non-trivial when the structure of the peer-to-peer system changes underneath the measurements. First-generation measurement studies of P2P systems typically relied on ad-hoc sampling techniques (*e.g.*, [60, 61]) and provided valuable information concerning basic system behavior. However, lacking any critical assessment of the quality of these sampling techniques, the measurements resulting from these studies may be biased and consequently our understanding of P2P systems may be based on incorrect or misleading conclusions.

In this chapter we primarily consider the challenging problem of obtaining unbiased samples from unstructured P2P systems, where peers select neighbors through a predominantly random process. Most popular P2P systems in use today belong to this unstructured category. For structured P2P systems such as Chord [82] and CAN [23], knowledge of the structure significantly facilitates unbiased sampling as we discuss in Section 5.6.

The main contributions of this chapter are *(i)* a detailed examination of the ways that the topological and temporal qualities of peer-to-peer systems can introduce bias and *(ii)* an in-depth exploration of the applicability of a sampling technique called the *Metropolized Random Walk with Backtracking (MRWB)*, representing a variation of the Metropolis–Hastings method [83–85]. Our study indicates that MRWB results in nearly unbiased samples under a wide variety of commonly encountered peer-to-peer network conditions. The technique assumes that the P2P system provides some mechanism to query a peer for a list of its neighbors: a capability provided by most widely deployed P2P systems. We also implemented the proposed sampling technique in a tool called `ion-sampler`. Our evaluations show that MRWB yields more accurate samples than previously considered sampling techniques. We quantify the observed differences, explore underlying causes, and discuss the implications on accurate inference of P2P properties and high-fidelity modeling of P2P systems. While our focus is on P2P networks, many of our results apply to any large, dynamic, undirected graph where nodes may be queried for a list of their neighbors.

Building on our earlier formulation in [16], the basic problem in sampling P2P networks concerns the selection of representative samples of *peer properties* such as peer degree, link bandwidth, or the number of files shared. To measure peer properties, any sampling technique needs to locate a set of peers in the overlay and then gather data about the desired properties.

An unbiased sampling technique will select any of the peers present with equal probability. While relatively straightforward in a static and known environment, this objective poses considerable problems in a highly dynamic setting like P2P systems, which can easily lead to significant measurement bias for two reasons. The first cause of sampling bias derives from the temporal dynamics of these systems, whereby new peers can arrive and existing peers can depart at any time. Locating a set of peers and measuring their properties takes time, and during that time the peer constituency is likely to change. In Section 5.2, we show how this often leads to bias towards short-lived peers and explain how to overcome this difficulty. The second significant cause of bias has to do with the connectivity structure of P2P systems. As a sampling program explores a given topological structure, each traversed link is in general more likely to lead to a high-degree peer than a low-degree peer, significantly biasing peer selection towards high-degree peers. We describe and evaluate different techniques for traversing static overlays to select peers in Section 5.3 and find that the Metropolized Random Walk (MRW) collects unbiased samples. In Section 5.4, we adapt MRW for dynamic overlays by adding backtracking and demonstrate its viability and effectiveness when the causes for both temporal and topological bias are present. We show via simulations that the MRWB technique works well and produces nearly unbiased samples under a variety of circumstances commonly encountered in actual P2P systems.

Finally, in Section 5.5 we describe the implementation of the `ion-sampler` tool based on MRWB and empirically evaluate its accuracy through comparison with complete snapshots of Gnutella taken with Cruiser [10], as well as compare it with results from previously used, more ad-hoc, sampling techniques. Section 5.6 discusses some important questions such as how many samples to collect, when sampling with a *known* bias may be desirable, and outlines a practical solution to obtaining unbiased samples for structured P2P systems. Section 5.7 concludes the chapter with a summary of our findings.

Material in this chapter was adapted from material previously published in proceedings [16, 18]. The material was co-authored with Prof. Reza Rejaie, Dr. Nick Duffield, Dr. Subhabrata Sen, and Dr. Walter Willinger. The experimental work is entirely mine. The writing is primarily mine, with small contributions from the each of the co-authors, who also provided technical guidance. Prof. David Levin provided the suggestion for using the Metropolis–Hastings method. The dynamic overlay simulator used in Section 5.4 was initially developed for another project by Amir Rasti and improved by John Capehart. I made additional improvements and added the code for performing random walks.

5.1 Related Work

The phrase "graph sampling" means different things in different contexts. We provide an overview of some of the different meanings of graph sampling to place our work in the context of other research on sampling graphs. Sampling from a class of graphs has been well studied in the graph theory literature [86, 87], where the main objective is to prove that for a class of graphs sharing some property (*e.g.*, same node degree distribution), a given random algorithm is capable of generating all graphs in the class. Cooper *et al.* [88] used this approach to show that their algorithm for overlay construction generates graphs with good properties. Our objective is quite different; instead of *sampling a graph from a class of graphs* our concern is *sampling peers (*i.e.*, vertices) from a largely unknown and dynamically changing graph.*

Others have used sampling to extract information about graphs (*e.g.*, selecting representative subgraphs from a large, intractable graph) while maintaining properties of the original structure [89–91]. Sampling is also frequently used as a component of efficient, randomized algorithms [92]. However, these studies assume complete knowledge of the graphs in question. Our problem is quite different in that we do not know the graphs in advance.

A closely related problem to ours is sampling Internet routers by running traceroute from a few hosts to many destinations for the purpose of discovering the Internet's router-level topology. Using simulation [93] and analysis [94], research has shown that traceroute measurements can result in measurement bias in the sense that the obtained samples support the inference of power law-type degree distributions irrespective of the true nature of the underlying degree distribution. A common feature of our work and the study of the traceroute technique [93, 94] is that both efforts require an evaluation of sampling techniques without complete knowledge of the true nature of the underlying connectivity structure. However, exploring the router topology and P2P topologies differ in their basic operations for graph-exploration. In the case of traceroute, the basic operation is “What is the path to this destination?”. In P2P, the basic operation is “What are the neighbors of this peer?”. In addition, the Internet’s router-level topology changes at a much slower rate than the overlay topology of P2P networks.

Another closely related problem is selecting Web pages uniformly at random from the set of all Web pages [95–97]. Web pages naturally form a graph, with hyperlinks forming edges between pages. Unlike peer-to-peer networks, the Web graph is *directed* and only outgoing links are easily discovered. Much of the work on sampling Web pages therefore focuses on estimating the number of incoming links, to facilitate degree correction. Unlike peers in peer-to-peer systems, not much is known about the temporal stability of Web pages, and temporal causes of sampling bias have received little attention in past measurement studies of the Web.

Several properties of random walks on graphs have been extensively studied analytically [98], such as the access time, cover time, and mixing time. While these properties have many useful applications, they are only well-defined for static graphs. To our knowledge the application of random walks as a method of selecting nodes uniformly at random from a *dynamically changing* graph has not been studied.

A number of papers [9, 76, 99, 100] have made use of random walks as a basis for searching unstructured P2P networks. However, searching simply requires locating a certain piece of data *anywhere* along the walk, and is not particularly concerned if some nodes are preferred over others. Some studies [76, 100] additionally use random walks as a component of their overlay-construction algorithm.

Recent work by Leskovec *et al.* [101] discusses the evolution of graphs over time and focuses on empirically observed properties such as densification (*i.e.*, networks become denser over time) and shrinking diameter (*i.e.*, as networks grow, their diameter decreases) and on new graph generators that account for these properties. However, the changes they observe occur on the time-scale of years, while we are concerned with short time-scale events (*e.g.*, minutes). Additionally, the graphs they examine are not P2P networks and their properties are by and large inconsistent with the design and usage of actual P2P networks. Hence, the graph models proposed in [101] are not appropriate for our purpose.

Awan *et al.* [102] also address the problem of gathering uniform samples from peer-to-peer networks. They examine several techniques, including the Metropolis–Hastings method, but only evaluate the techniques over static power-law graphs. Their formulation of the Metropolis–Hastings method, as well as the Random Weight Distribution method which they advocate, require special underlying support from the peer-to-peer application. We implement Metropolis–Hastings in such a way that it relies only on being able to discover a peer's neighbors, a simple primitive operation commonly found in existing peer-to-peer networks, and introduce backtracking to cope with departed peers. We also conduct a much more extensive evaluation of the proposed MRWB method. In particular, we evaluate MRWB over dynamically changing graphs with a variety of topological properties. We also perform empirical validations over an actual P2P network.

5.2 Sampling with Dynamics

We develop a formal and general model of a P2P system as follows. If we take an instantaneous snapshot of the system at time t, we can view the overlay as a graph $G(V, E)$ with the peers as vertices and connections between the peers as edges. Extending this notion, we incorporate the dynamic aspect by viewing the system as an infinite set of time-indexed graphs, $G_t = G(V_t, E_t)$. The most common approach for sampling from this set of graphs is to define a measurement window, $[t_0, t_0 + \Delta]$, and select peers uniformly at random from the set: $V_{t_0,t_0+\Delta} = \bigcup_{t=t_0}^{t_0+\Delta} V_t$. Thus, we do not distinguish between occurrences of the same peer at different times.

This formulation is appropriate if peer session lengths are exponentially distributed (*i.e.*, memoryless). However, existing measurement studies [19, 60, 64, 65] show session lengths are heavily skewed, with many peers being present for just a short time (a few minutes) while other peers remain in the system for a very long time (*i.e.*, longer than Δ). As a consequence, as Δ increases, the set $V_{t_0,t_0+\Delta}$ includes an increasingly large fraction of short-lived peers.

A simple example may be illustrative. Suppose we wish to observe the number of files shared by peers. In this example system, half the peers are up all the time and have many files, while the other peers remain for around 1 minute and are immediately replaced by new short-lived peers who have few files. The technique used by most studies would observe the system for a long time (Δ) and incorrectly conclude that most of the peers in the system have very few files. Moreover, their results will depend on how long they observe the system. The longer the measurement window, the larger the fraction of observed peers with few files.

One fundamental problem of this approach is that it focuses on sampling *peers* instead of *peer properties*. It selects each sampled vertex at most once. However, the property at the vertex may change with time. Our goal should not be to select a vertex $v_i \in \bigcup_{t=t_0}^{t_0+\Delta} V_t$, but rather to sample the property at v_i at a particular instant t. Thus, we distinguish between occurrences of the same peer at different times: samples $v_{i,t}$ and $v_{i,t'}$ gathered at distinct times $t \neq t'$ are viewed as distinct, even when they come from the same peer. *The key difference is that it must be possible to sample from the same peer more than once, at different points in time.* Using the formulation $v_{i,t} \in V_t,\ t \in [t_0, t_0 + \Delta]$, the sampling technique will not be biased by the dynamics of peer behavior, because the sample set is decoupled from peer session lengths. To our knowledge, no prior P2P measurement studies relying on sampling make this distinction.

Returning to our simple example, our approach will correctly select long-lived peers half the time and short-lived peers half the time. When the samples are examined, they will show that half of the peers in the system at any given moment have many files while half of the peers have few files, which is exactly correct.

Another problem is that using a large Δ captures the average behavior of the system, which may not reflect the true state of the system at any particular moment. In particular, it precludes study of how the system as a whole evolves, such as due to the time-of-day effect or a flash crowd. A better approach would be to gather several series of measurements, each over some short Δ, then compare them. If Δ is sufficiently small, such that the distribution of the property under consideration does not change significantly during the measurement window, then we may relax the constraint of choosing t uniformly at random from $[t_0, t_0 + \Delta]$.

We still have the significant problem of selecting a peer uniformly at random from those present at a particular time. We begin to address this problem in the next section.

5.3 Sampling from Static Graphs

We now turn our attention to topological causes of bias. Towards this end, we momentarily set aside the temporal issues by assuming a static, unchanging graph. The selection process begins with knowledge of one peer (vertex) and progressively queries peers for a list of neighbors. The goal is to select peers uniformly at random. In any graph-exploration problem, we have a set of visited peers (vertices) and a front of unexplored neighboring peers. There are two ways in which algorithms differ: *(i)* how to chose the next peer to explore, and *(ii)* which subset of the explored peers to select as samples. Prior studies use simple breadth-first or depth-first approaches to explore the graph and select all explored peers. These approaches suffer from several problems:

- The discovered peers are correlated by their neighbor relationship.
- Peers with higher degree are more likely to be selected.
- Because they never visit the same peer twice, they will introduce bias when used in a dynamic setting as described in Section 5.2.

Random Walks: A better candidate solution is the random walk, which has been extensively studied in the graph theory literature (for an excellent survey see [98]). We briefly summarize the key terminology and results relevant to sampling. The transition matrix $P(x, y)$ describes the probability of transitioning to peer y if the walk is currently at peer x:

$$P(x,y) = \begin{cases} \frac{1}{\text{degree}(x)} & y \text{ is a neighbor of x,} \\ 0 & \text{otherwise} \end{cases}$$

If the vector v describes the probability of currently being at each peer, then the vector $v' = vP$ describes the probability after taking one additional step. Likewise, vP^r describes the probability after taking r steps. As long as the graph is connected and not bipartite, the probability of being at any particular node, x, converges to a *stationary distribution*:

$$\pi(x) = \lim_{r\to\infty}(vP^r)(x) = \frac{\text{degree}(x)}{2\cdot|E|}$$

In other words, if we select a peer as a sample every r steps, for sufficiently large r, we have the following good properties:

- The information stored in the starting vector, v, is lost, through the repeated selection of random neighbors. Therefore, there is no correlation between selected peers. Alternately, we may start many walks in parallel. In either cases, after r steps, the selection is independent of the origin.
- While the stationary distribution, $\pi(x)$, is biased towards peers with high degree, the bias is precisely known, allowing us to correct it.
- Random walks may visit the same peer twice, which lends itself better to a dynamic setting as described in Section 5.2.

In practice, r need not be exceptionally large. For graphs where the edges have a strong random component (such as in peer-to-peer networks), it is sufficient that the number of steps exceed the log of the population size, *i.e.*, $r \geq O(\log|V|)$.

Adjusting for degree bias: To correct for the bias towards high degree peers, we make use of the *Metropolis–Hastings method* [83–85] for Markov Chains. Random walks on a graph are a special case of Markov Chains. In a regular random walk, the transition matrix $P(x, y)$ leads to the stationary distribution $\pi(x)$, as described above. The Metropolis–Hastings method provides us with a way to build a modified transition matrix, $Q(x, y)$, leading to a target stationary distribution $\mu(x)$, as follows:

$$Q(x,y) = \begin{cases} P(x,y)\min\left(\frac{\mu(y)P(y,x)}{\mu(x)P(x,y)}, 1\right) & \text{if } x \neq y, \\ 1 - \sum_{x\neq y} Q(x,y) & \text{if } x = y \end{cases}$$

Equivalently, to take a step from peer x, select a neighbor y of x as normal (*i.e.*, with probability $P(x, y)$). Then, with probability $\min\left(\frac{\mu(y)P(y,x)}{\mu(x)P(x,y)}, 1\right)$, accept the move.

Otherwise, return to x. For a proof this definition of $Q(x, y)$ leads to sampling peer x with probability $\mu(x)$, see [83].

To collect uniform samples, we have $\frac{\mu(y)}{\mu(x)} = 1$, so the move-acceptance probability becomes:

$$\min\left(\frac{\mu(y)P(y,x)}{\mu(x)P(x,y)}, 1\right) = \min\left(\frac{\text{degree}(x)}{\text{degree}(y)}, 1\right)$$

Therefore, our algorithm for selecting the next step from some peer x is as follows:

- Select a neighbor y of x uniformly at random.
- Query y for a list of its neighbors, to determine its degree.
- Generate a random number, p, uniformly between 0 and 1.
- If $p \leq \frac{\text{degree}(x)}{\text{degree}(y)}$, y is the next step.
- Otherwise, remain at x as the next step.

We call this the Metropolized Random Walk (MRW). Qualitatively, the effect is to suppress the rate of transition to peers of higher degree, resulting in selecting each peer with equal probability.

Evaluation: Although [83] provides a proof of correctness for the Metropolis–Hastings method, to ensure the correctness of our implementation we conduct evaluations through simulation over static graphs. This additionally provides the opportunity to compare MRW with conventional techniques such as Breadth-First Search (BFS) or naive random walks (RW) with no adjustments for degree bias.

To evaluate a technique, we use it to collect a large number of sample vertices from a graph, then perform a goodness-of-fit test against the uniform distribution. For Breadth-First Search, we simulate typical usage by running it to gather a batch of 1,000 peers. When one batch of samples is collected, the process is reset and begins anew at a different starting point. To ensure robustness with respect to different kinds of connectivity structures, we examine each technique over several types of graphs as follows:

Erdös–Rényi: The simplest variety of random graphs

Watts–Strogatz: “Small world” graphs with high clustering and low path lengths

Barabási–Albert: Graphs with extreme degree distributions, also known as power-law or scale-free graphs

Gnutella: Snapshots of the Gnutella ultrapeer topology, captured by Cruiser (Chapter 4)

To make the results more comparable, the number of vertices ($|V| = 161,680$) and edges ($|E| = 1,946,596$) in each graph are approximately the same.[1] Table 5.1 presents the results of the goodness-of-fit tests after collecting $1000 \cdot |V|$ samples, showing that Metropolis–Hastings appears to generate uniform samples over each type of graph, while the other techniques fail to do so by a wide margin.

Figure 5.1 explores the results visually, by plotting the number of times each peer is selected. If we select $k \cdot |V|$ samples, the typical node should be selected k times, with other nodes being selected close to k times approximately following a normal distribution with variance k.[2] We used $k = 1,000$ samples. We also include an “Oracle” technique, which selects peers uniformly at random using global information. The Metropolis–Hastings results are virtually identical to the Oracle, while the other

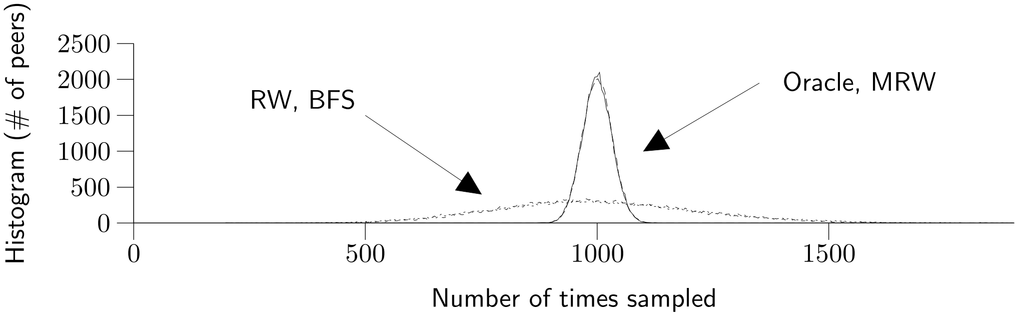
	Erdös–Rényi	Gnutella	Watts–Strogatz	Barabási–Albert
Breadth-First Search	$4.54 \cdot 10^{-4}$	$2.73 \cdot 10^{-3}$	4.73^{-3}	$2.77 \cdot 10^{-3}$
Random Walk	$3.18 \cdot 10^{-4}$	$1.57 \cdot 10^{-3}$	7.64^{-5}	$2.84 \cdot 10^{-3}$
Metropolis–Hastings	$5.97 \cdot 10^{-5}$	$5.79 \cdot 10^{-5}$	6.08^{-5}	$5.22 \cdot 10^{-5}$

TABLE 5.1: Kolmogorov–Smirnov test statistic for techniques over static graphs. Values above $1.07 \cdot 10^{-4}$ lie in the rejection region at the 5% level.

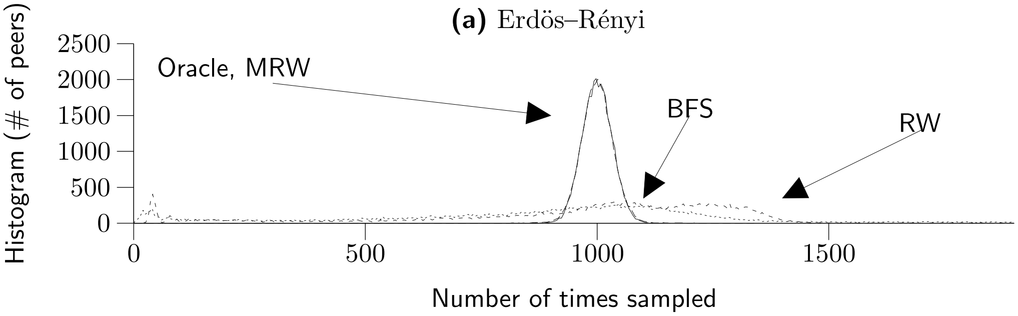
---

[1] Erdös–Rényi graphs are generated based on some probability p that any edge may exist. We set $p = \frac{2|E|}{|V| \cdot (|V|-1)}$ so that there will be close to $|E|$ edges, though the exact value may vary slightly. The Watts–Strogatz model require that $|E|$ be evenly divisible by $|V|$, so in that model we use $|E| = 1,940,160$.

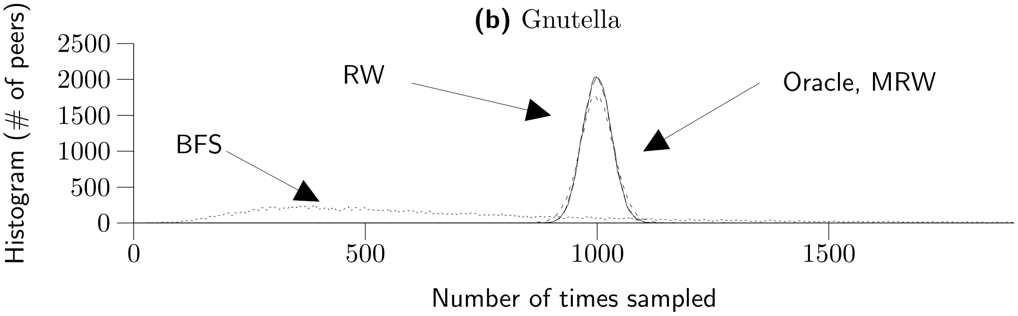
[2] Based on the normal approximation of a binomial distribution with $p = \frac{1}{|V|}$ and $n = k|V|$.



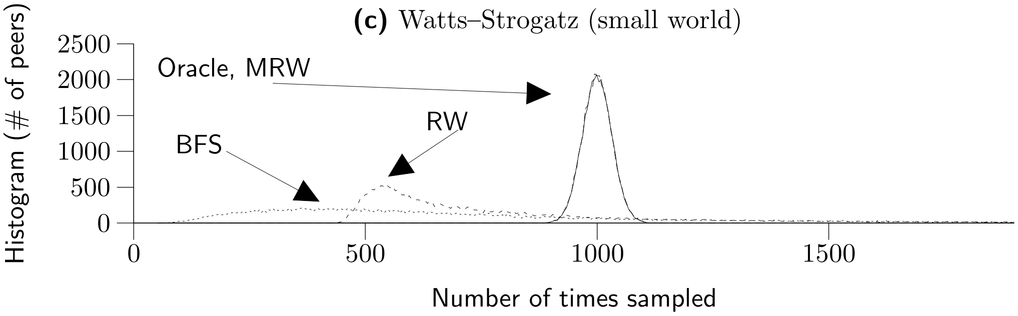
(a) Erdös–Rényi



(b) Gnutella



(c) Watts–Strogatz (small world)



(d) Barabási–Albert (power-law)

FIGURE 5.1: Bias of different sampling techniques; after collecting $k \cdot |V|$ samples. The figures show how many peers (y-axis) were selected x times.

techniques select many peers much more and much less than k times. In the Gnutella, Watts–Strogatz, and Barabási–Albert graphs, Breadth-First Search exhibits a few vertices that are selected a large number of times ($> 10,000$). The (not-adjusted) Random Walk (RW) method has similarly selected a few vertices an exceptionally large number of times in the Gnutella and Barabási–Albert models. The Oracle and MRW, by contrast, did not select any vertex more than around 1,300 times.

In summary, the Metropolis–Hastings method selects peers uniformly at random from a static graph. The next section examines the additional complexities when selecting from a dynamic graph, introduces appropriate modifications, and evaluates the algorithm's performance.

5.4 Sampling from Dynamic Graphs

Section 5.2 set aside topological issues and examined the dynamic aspects of sampling. Section 5.3 set aside temporal issues and examined the topological aspects of sampling. This section examines the unique problems that arise when both temporal and topological difficulties are present.

Our hypothesis is that a Metropolis–Hastings random walk will yield approximately unbiased samples even in a dynamic environment. The fundamental assumption of Metropolis–Hastings is that the frequency of visiting a peer is proportional to the peer's degree. We argue that this assumption will be approximately correct if peer relationships change only slightly during the walk. On one extreme, if the entire walk completes before any graph changes occur, then the problem reduces to the static case. If a single edge is removed mid-walk, the probability of selecting the two affected peers is not significantly affected, unless those peers have very few edges. If many edges are added and removed during a random walk, but the degree of each peer does not change significantly, we would also expect that the probability of selecting each peer will not change significantly. In peer-to-peer systems, each peer actively tries to maintain a number of connections within a certain range, so we have reason to believe that the degree of each peer will be relatively stable in practice. On the other hand, it is quite possible that in a highly dynamic environment, or for certain degree distributions, the assumptions of Metropolis–Hastings are grossly violated and it fails to gather approximately unbiased samples.

The fundamental question we attempt to answer in this section is: *Under what conditions does the Metropolis–Hastings random walk fail to gather approximately unbiased samples?* Intuitively, if there is any bias in the samples, the bias will be tied to some property that interacts with the walk. We identify the following three fundamental properties that interact with the walk:

Degree: The Metropolis–Hastings method is a modification of a regular random walk in order to correct for degree-bias as described in Section 5.3. It assumes a fixed relationship between degree and the probability of visiting a peer. If the Metropolis–Hastings assumptions are invalid, the degree-correction may not operate correctly, introducing a bias correlated with degree.

Session lengths: Section 5.2 showed how sampling may result in a bias based on session length. If the walk is more likely to select either short-lived or long-lived peers, there will be a bias correlated with session length.

Query latency: In a static environment the only notion of time is the number of steps taken by the walk. In a dynamic environment, each step requires querying a peer, and some peers will respond more quickly than others. This could lead to a bias correlated with the query latency. In our simulations, we model the query latency as twice the round-trip time between the sampling node and the peer being queried.[3]

For other peer properties, sampling bias can only arise if the desired property is correlated with a fundamental property and that fundamental property exhibits bias. For example, when sampling the number of files shared by each peer, there may be sampling bias if the number of files is correlated with session length *and* sampling is

---

[3] $\frac{1}{2}$ RTT for the SYN, $\frac{1}{2}$ RTT for the SYN-ACK, $\frac{1}{2}$ RTT for the ACK and the request, and $\frac{1}{2}$ RTT for the reply.

biased with respect to session length. One could also imagine the number of files being correlated with query latency (which is very loosely related to the peer bandwidth). However, sampling the number of shared files cannot be biased independently, as it does not interact with the walk. To show that sampling is unbiased for any property, it is sufficient to show that it is unbiased for the fundamental properties that interact with the sampling technique.

5.4.1 Adapting Random Walks for a Dynamic Environment

Departing peers introduce an additional practical consideration. The walk may try to query a peer that is no longer present–a case where the behavior of the ordinary random walk algorithm is undefined. We make an adaptation by maintaining a stack of visited peers. When the walk chooses a new peer to query, we push the peer's address on the stack. If the query times out, we pop the address off the stack, and choose a new neighbor of the peer that is now on top of the stack. If all of a peer's neighbors time out, we re-query that peer to get a fresh list of its neighbors. If the re-query also times out, we pop that peer from the stack as well, and so on. If the stack underflows, we consider the walk a failure. We do not count timed-out peers as a hop for the purposes of measuring the length of the walk. We call this adaptation of the MRW sampling technique the *Metropolized Random Walk with Backtracking (MRWB)* method for sampling from dynamic graphs. Note that when applied in a static environment, this method reduces to MRW.

5.4.2 Evaluation Methodology

In the static case, we can rely on graph theory to prove the accuracy of the MRW technique. Unfortunately, graph theory is not well-suited to the problem of dynamically changing graphs. Therefore, we rely on simulation rather than analysis. We have developed a session-level dynamic overlay simulator that models peer arrivals, departures, latencies, and neighbor connections. We now describe our simulation environment.

The latencies between peers are modeled using values from the King data set [103]. Peers learn about one another using one of several *peer discovery* mechanisms described below. Peers have a target minimum number of connections (*i.e.*, degree) that they attempt to maintain at all times. Whenever they have fewer connections, they open additional connections. We assume connections are TCP and require a 3-way handshake before the connection is fully established, and that peers will time out an attempted connection to a departed peer after 10 seconds. A new peer generates its session length from one of several different session length distributions described below and departs when the session length expires. New peers arrive according to a Poisson process, where we select the mean peer arrival rate based on the session length distribution to achieve a target population size of 100,000 peers.

To query a peer for a list of neighbors, the sampling node must set up a TCP connection, submit its query, and receive a response. The query times out if no response is received after 10 seconds.[4] We run the simulator for a warm-up period to reach steady-state conditions before performing any random walks.

Our goal is to discover if random walks started under identical conditions will select a peer uniformly at random. To evaluate this, we start 100,000 concurrent random walks from a single location. Although started at the same time, the walks will not all complete at the same time.[5] We chose to use 100,000 walks as we believe this is a much larger number of samples than most researchers will use in practice. If there is no discernible bias with 100,000 samples, we can conclude that the tool is unbiased for the purposes of gathering fewer samples (*i.e.*, we cannot get more accuracy by using less precision). Figure 5.2 shows the distribution of how long walks take to complete in one simulation using 50 hops per walk, illustrating that most walks take 10–20 seconds to complete. In the simulator the walks do not interact or interfere with one another in any way. Each walk ends and collects an independent sample.

---

[4] The value of 10 seconds was selected based on our experiments in developing a crawler for the Gnutella network in Chapter 4.

[5] Each walk ends after the same number of *hops*, but not every hop takes the same amount of *time* due to differences in latencies and due to the occasional timeout.

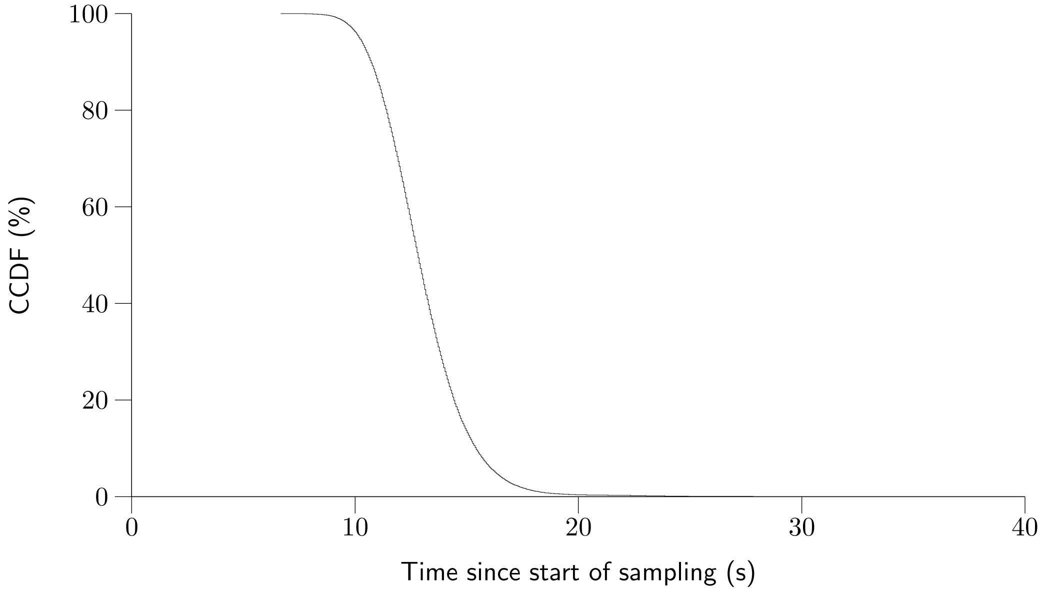


FIGURE 5.2: Distribution of time needed to complete a random walk (simulated)

As an expected distribution, we capture a perfect snapshot (*i.e.*, using an oracle) at the median walk-completion time, *i.e.*, when 50% of the walks have completed.

5.4.3 Evaluation of a Base Case

Because the potential number of simulation parameters is unbounded, we need a systematic method to intelligently explore the most interesting portion of this parameter space. Towards this end, we begin with a base case of parameters as a starting point and examine the behavior of MRWB under those conditions. In the following subsections, we vary the parameters and explore how the amount of bias varies as a function of each of the parameters. As a base case, we use the following configuration:

Session length distribution:	Weibull($k = 0.59$, $\lambda = 40$)
Target degree:	15
Maximum degree:	30
Peer discovery mechanism:	FIFO

TABLE 5.2: Base case configuration

Figure 5.3 presents the sampled and expected distributions for the three fundamental properties: degree, session length, and query latency. The fact that the sampled and expected distributions are visually indistinguishable demonstrates that the samples are not significantly biased in the base case.

To efficiently examine other cases, we introduce a *summary statistic* to quickly capture the difference between the sampled and expected distributions, and to provide more rigor than a purely visual inspection. For this purpose, we use the Kolmogorov–Smirnov (KS) statistic, D, formally defined as follows. Where $S(x)$ is the sampled cumulative distribution function and $E(x)$ is the expected cumulative distribution function from the perfect snapshot, the KS statistic is:

$$D = \max\left(|S(x) - E(x)|\right)$$

In other words, if we plot the sampled and expected CDFs, D is the maximum vertical distance between them and has a possible range of $[0, 1]$. For Figures 5.3a, 5.3b, and 5.3c, the values of D were 0.0019, 0.0023, and 0.0037, respectively. For comparison, at the $p = 0.05$ significance level, D is 0.0061, for the two-sample KS statistic with 100,000 data points each. However, in practice we do not expect most researchers to gather hundreds of thousands of samples. After all, the initial motivation for sampling is to gather reasonably accurate data at relatively low cost. As a rough rule of thumb, a value of $D \geq 0.1$ is quite bad, corresponding to at least a 10 percentage point difference on a CDF. A value of $D \leq 0.01$ is excellent for most purposes when studying a peer property, corresponding to no more than a 1 percentage point difference on a CDF.

5.4.4 Exploring Different Dynamics

In this section, we examine how the amount of bias changes as we vary the type and rate of dynamics in the system. We examine different settings of the simulation parameters that affect dynamics, while continuing to use the topological characteristics from our base case (Table 5.2). We would expect that as the rate of peer dynamics increases, the sampling error also increases. The key question is: *How fast can the churn rate be before it causes significant error, and is that likely to occur in practice?*

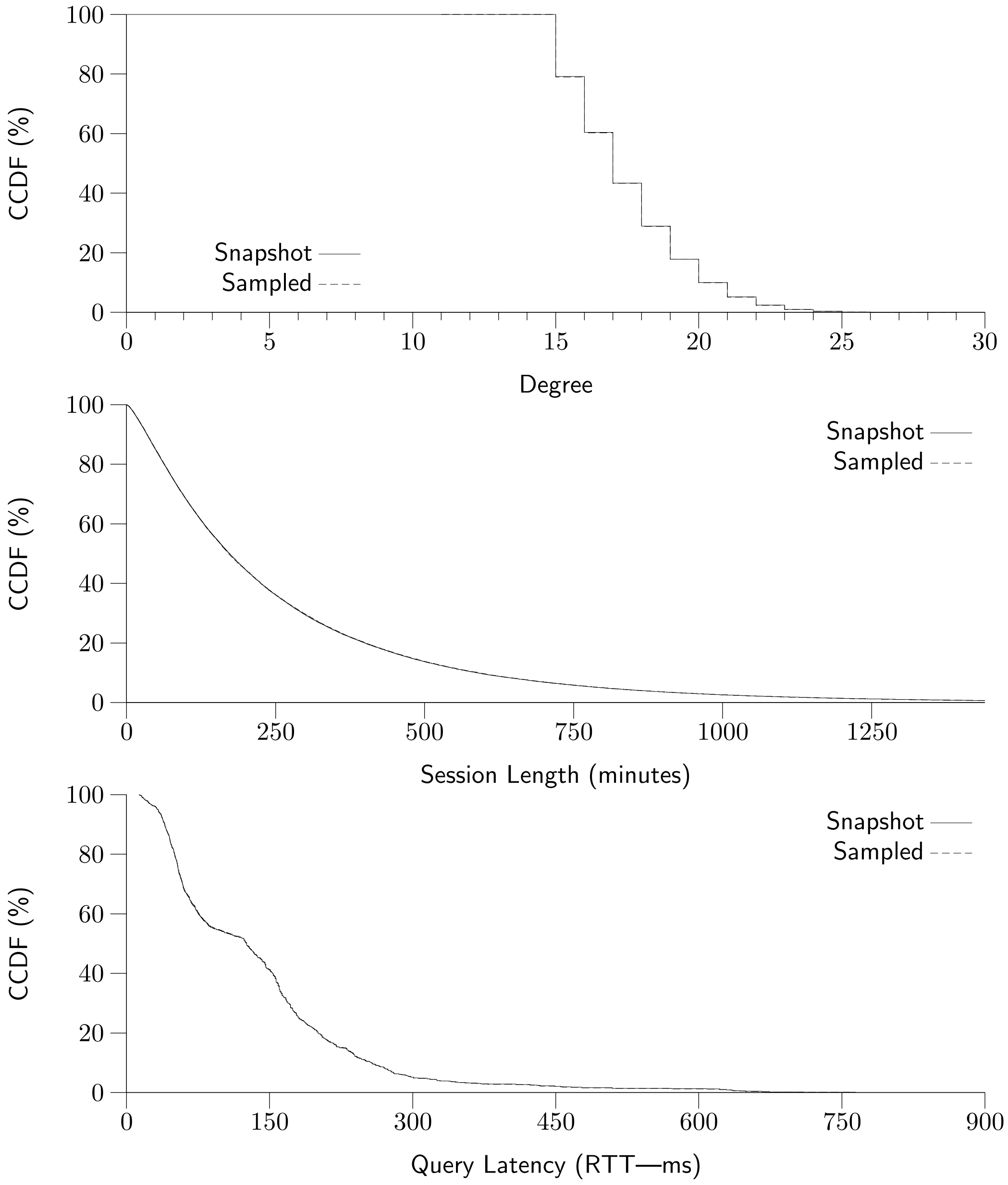


FIGURE 5.3: Comparison of sampled and expected distributions. They are visually indistinguishable.

In this subsection, we present the results of simulations with a wide variety of rates using three different models for session length, as follows:

Exponential: The exponential distribution is a one-parameter distribution (rate λ) that features sessions relatively close together in length. It has been used in many prior simulation and analysis studies of peer-to-peer systems [5, 29, 104].

Pareto: The Pareto (or power-law) distribution is a two-parameter distribution (shape α, location x_m) that features many short sessions coupled with a few very long sessions. Some prior measurement studies of peer-to-peer systems have suggested that session lengths follow a Pareto distribution [45, 48, 68]. One difficulty with this model is that x_m is a lower-bound on the session length, and fits of x_m to empirical data are often unreasonably high (*i.e.*, placing a lower bound significantly higher than the median session length reported by other measurement studies). In their insightful analytical study of churn in peer-to-peer systems, Leonard, Rai, and Loguinov [70] instead suggest using a shifted Pareto distribution (shape α, scale β) with $\alpha \approx 2$. We use this shifted Pareto distribution, holding α fixed and varying the scale parameter β. We examine two different α values: $\alpha = 1.9$ (infinite variance) and $\alpha = 2.1$ (finite variance).

Weibull: Our own empirical observations (Chapter 7) suggest the Weibull distribution (shape k, scale λ) provides a good model of peer session lengths, representing a compromise between the exponential and Pareto distributions. We fix $k = 0.59$ (based on our empirical data) and vary the scale parameter λ.

Figure 5.4 presents the amount of sampling error (D) as a function of median session length, for the three fundamental properties, with a logarithmic x-axis scale. The figure shows that error is low over a wide range of session lengths but begins

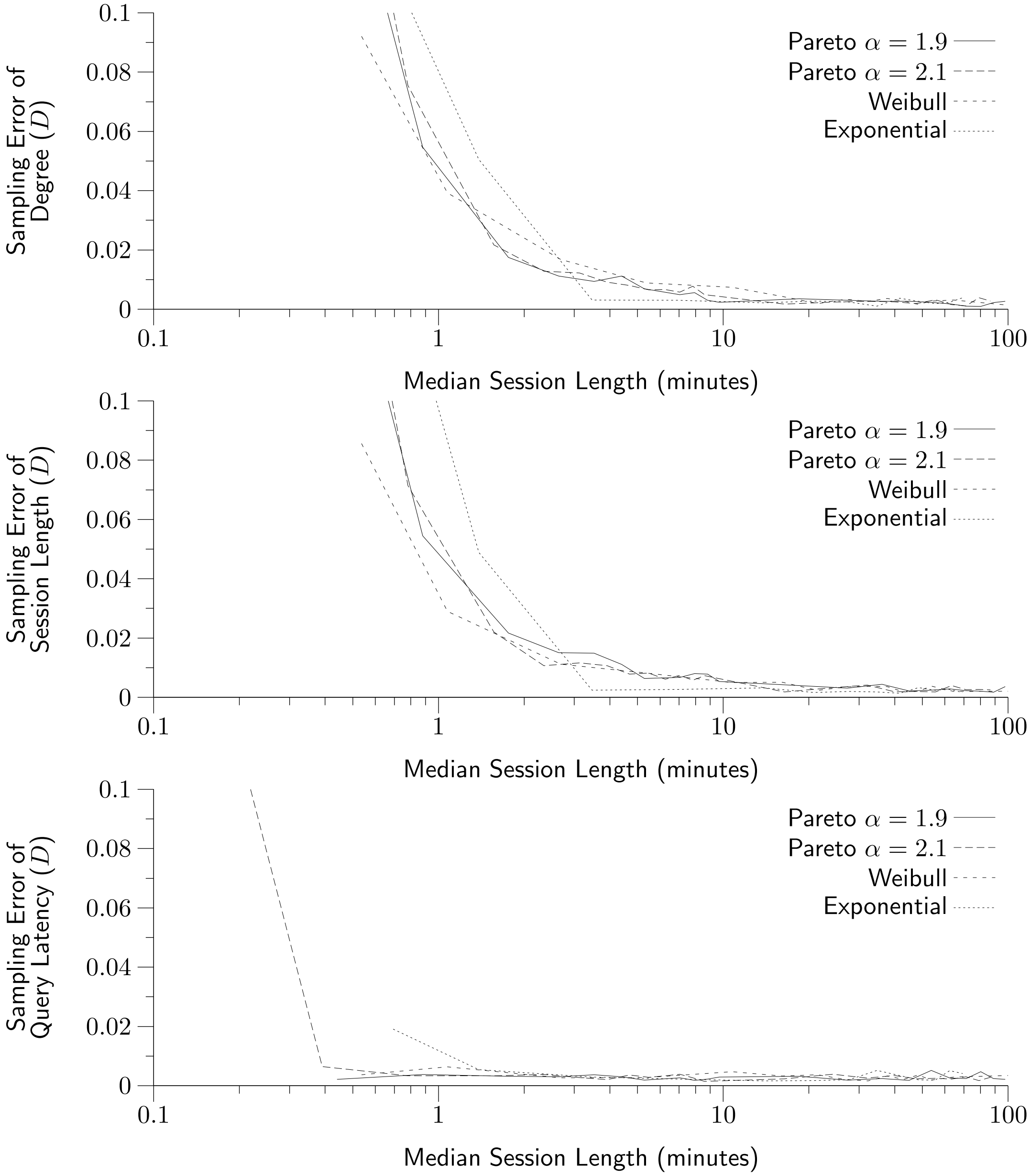


FIGURE 5.4: Sampling error of the three fundamental properties as a function of session-length distribution. Exceptionally heavy churn (median < 1min) introduces error into the sampling process.

to become significant when the median session length drops below 2 minutes, and exceeds $D = 0.1$ when the median drops below 30 seconds. The type of distribution varies the threshold slightly, but overall does not appear to have a significant impact. To investigate whether the critical threshold is a function of the length of the walk, we ran some simulations using walks of 10,000 hops (which take around one simulated hour to complete). Despite the long duration of these walks, they remained unbiased with $D < 0.003$ for each of the three fundamental properties. *This suggests that the accuracy of MRWB is affected primarily by the rate of local variation in the ratio* $\frac{degree(x)}{degree(y)}$ *relative to the time required to query peers, rather than the speed of global variations relative to the length of the walk.*

While the median session length reported by measurement studies varies considerably (see Table 3.3 on page 24 for a summary), none report a median below 1 minute and two studies report a median session length of one hour [60, 61]. In summary, these results demonstrate that MRWB can gracefully tolerate peer dynamics. In particular, it performs well over the rate of churn reported in real systems.

5.4.5 Exploring Different Topologies

In this section, we examine different settings of the simulation parameters that directly affect topological structure, while using the dynamic characteristics from our base case (Table 5.2). The Metropolis–Hastings method makes use of the ratio between the degrees of neighboring peers. If this ratio fluctuates dramatically while the walk is conducted, it may introduce significant bias. If peers often have only a few connections, any change in their degree will result in a large percentage-wise change. One key question is therefore: *Does a low target degree lead to sampling bias, and, if so, when is significant bias introduced?*

The degree of peers is controlled by three factors. First, each peer has a *peer discovery mechanism* that enables it to learn the addresses of potential neighbors. The peer discovery mechanism will influence the structure of the topology and, if performing poorly, will limit the ability of peers to establish connections. Second, peers have a *target degree* which they actively try to maintain. If they have fewer neighbors than the target, they open additional connections until they have reached the target. If necessary, they make use of the peer discovery mechanism to locate additional potential neighbors. Finally, peers have a *maximum degree*, which limits the number of neighbors they are willing to accept. If they are at the maximum and another peer contacts them, they refuse the connection. Each of these three factors influences the graph structure, and therefore may affect the walk.

We model four different types of peer discovery mechanisms:

- **Oracle:** This is the simplest and most idealistic approach. Peers learn about one another by contacting a rendezvous point that has perfect global knowledge of the system and returns a random set of peers for them to connect to.
- **FIFO:** In this scheme, inspired by the GWebCaches of Gnutella [79], peers contact a rendezvous point which returns a list of the last n peers that contacted the rendezvous, where n is the maximum peer degree.
- **Soft state:** Inspired by the approach of BitTorrent's "trackers", peers contact a rendezvous point that has *imperfect* global knowledge of the system. In addition to contacting the rendezvous point to learn about more peers, every peer periodically (every half hour) contacts the rendezvous point to refresh its state. If a peer fails to make contact for 45 minutes, the rendezvous point removes it from the list of known peers.
- **History:** Many P2P applications connect to the network using addresses they learned during a previous session [50]. A large fraction of these addresses will timeout, but typically enough of the peers will still be active to avoid the need to contact a centralized rendezvous point. As tracking the re-appearance of peers greatly complicates our simulator (as well as greatly increasing the memory requirements), we use a coarse model of the History mechanism. We assume that 90% of connections automatically timeout. The 10% that are given valid addresses are skewed towards peers that have been present for a long time (more than one hour) and represent regular users who might have been present during the peer's last session. While this might be overly pessimistic, it reveals the behavior of MRWB under harsh conditions.

Figure 5.5 presents the amount of sampling error (D) for the three fundamental properties as a function of the target degree, for each of the peer discovery methods,

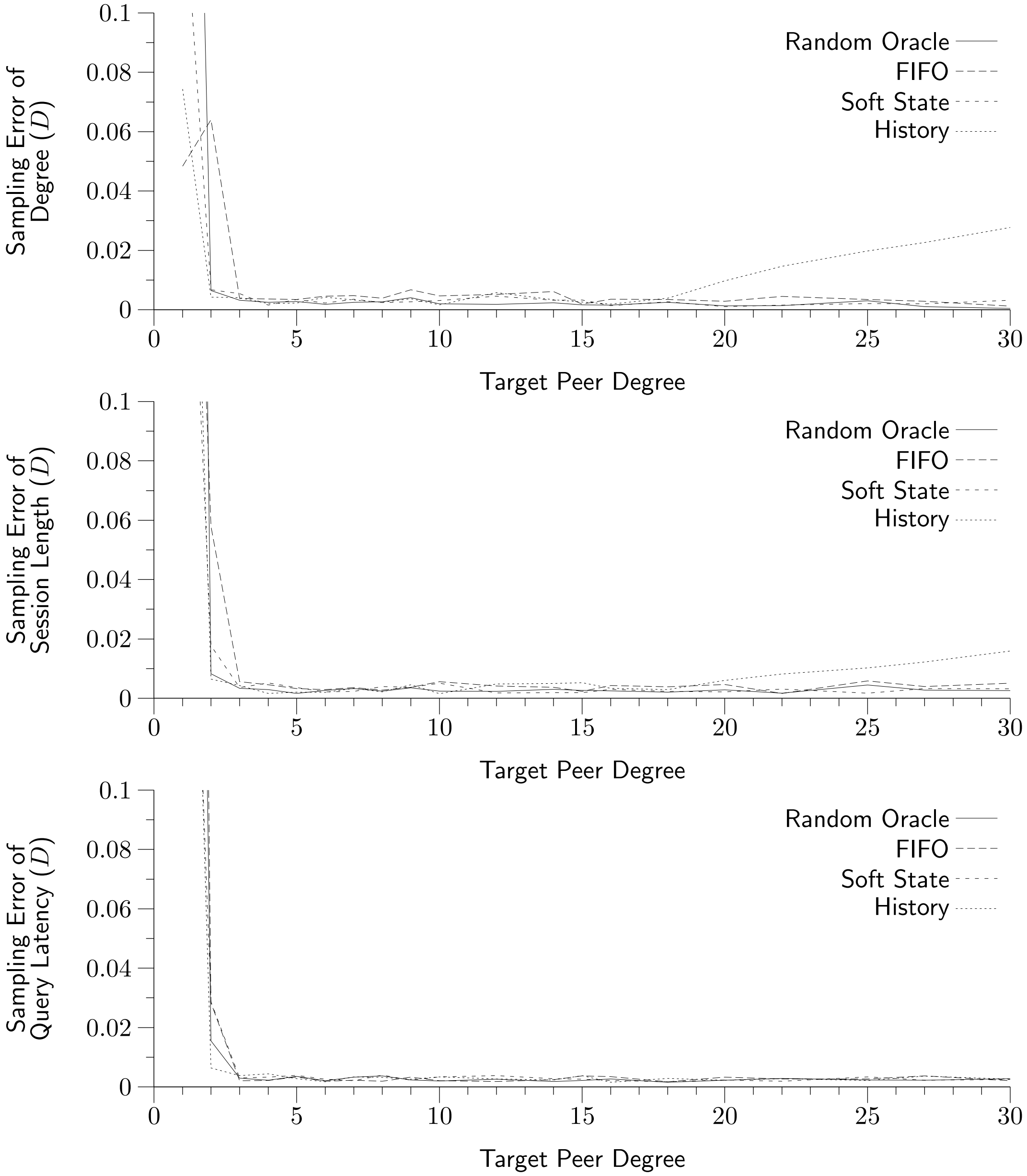


FIGURE 5.5: Sampling error of the three fundamental properties as a function of the number of connections each peer actively attempts to maintain. Low target degree (≤ 2) introduces significant sampling error.

holding the maximum peer degree fixed at 30 neighbors. It shows that sampling is not significantly biased in any of the three fundamental properties as long as peers attempt to maintain at least three connections. Widely deployed peer-to-peer systems typically maintain dozens of neighbors. Moreover, maintaining fewer than three neighbors per peer almost certainly leads to network fragmentation, and is therefore not a reasonable operating point for peer-to-peer systems.

The results for the different peer-discovery mechanisms were similar to one another, except for a small amount of bias observed when using the History mechanism as the target degree approaches the maximum degree (30). To investigate this issue, Figure 5.6 presents the sampled and expected degree distribution when using the History mechanism with a target degree of 30. The difference between the sampled and expected distributions is due to the 2.4% of peers with a degree of zero. These isolated peers arise in this scenario because the History mechanism has a high failure rate (returning addresses primarily of departed peers), and when a valid address is found, it frequently points to a peer that is already at its connection limit. The zero-degree peers are visible in the snapshot (which uses an oracle to obtain global

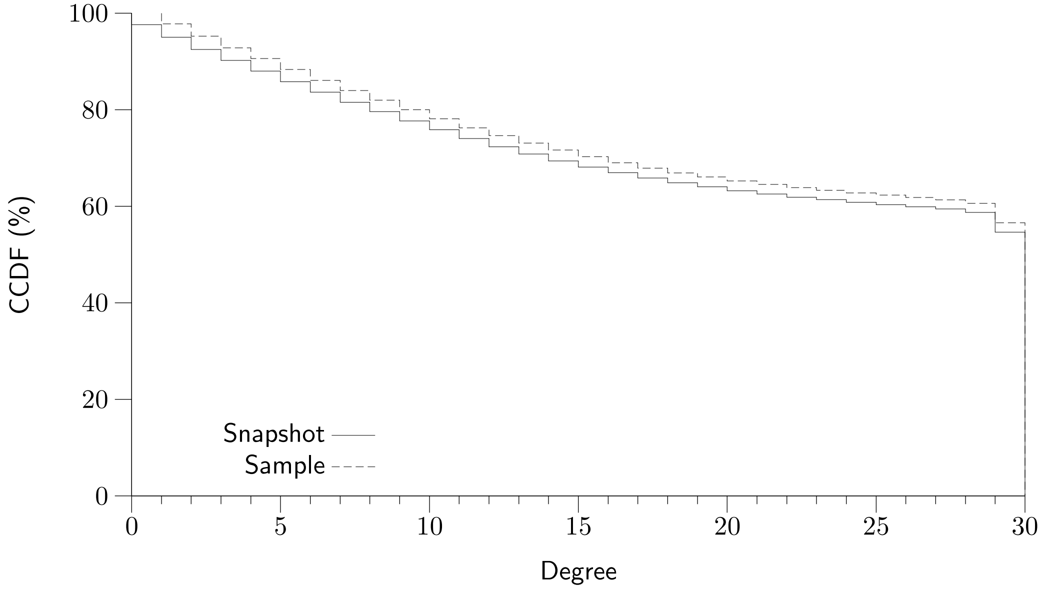


FIGURE 5.6: Comparison of degree distributions using the History mechanism with a target degree of 30. Sampling cannot capture the unconnected peers (degree $= 0$), causing the sampling error observed in Figure 5.5.

information), but not to the sampler (since peers with a degree of zero have no neighbors and can never be reached). We do not regard omitting disconnected peers as a serious limitation.

Having explored the effects of lowering the degree, we now explore the effects of increasing it. In Figure 5.7, we examine sampling error as a function of the maximum degree, with the target degree always set to 15 less than the maximum. There is little error for any setting of the maximum degree.

In summary, the proposed MRWB technique for sampling from dynamic graphs appears unbiased for a range of different topologies (with reasonable degree distributions; *e.g.*, degree ≥ 3), operates correctly for a number of different mechanisms for peer discovery, and is largely insensitive to a wide range of peer dynamics, with the churn rates reported for real systems safely within this range.

5.5 Empirical Results

In addition to the simulator version, we have implemented the MRWB algorithm for sampling from real peer-to-peer networks into a tool called `ion-sampler`. The following subsections briefly describe the implementation and usage of `ion-sampler` and present empirical experiments to validate its accuracy.

5.5.1 Ion-Sampler

The `ion-sampler` tool uses a modular design that accepts plug-ins for new peer-to-peer systems.[6] As long as the peer-to-peer system allows querying peers for a list of their neighbors, a plug-in can be written. The `ion-sampler` tool hands IP-address:port pairs to the plug-in, which later returns a list of neighbors or signals that a timeout occurred. The `ion-sampler` tool is responsible for managing the walks. It outputs the samples to standard output, where they may be easily read by another tool that collects the actual measurements. For example, `ion-sampler`

---

[6] In fact, it uses the same plug-in architecture as our earlier, heavy-weight tool, Cruiser, which exhaustively crawls peer-to-peer systems to capture topology snapshots.

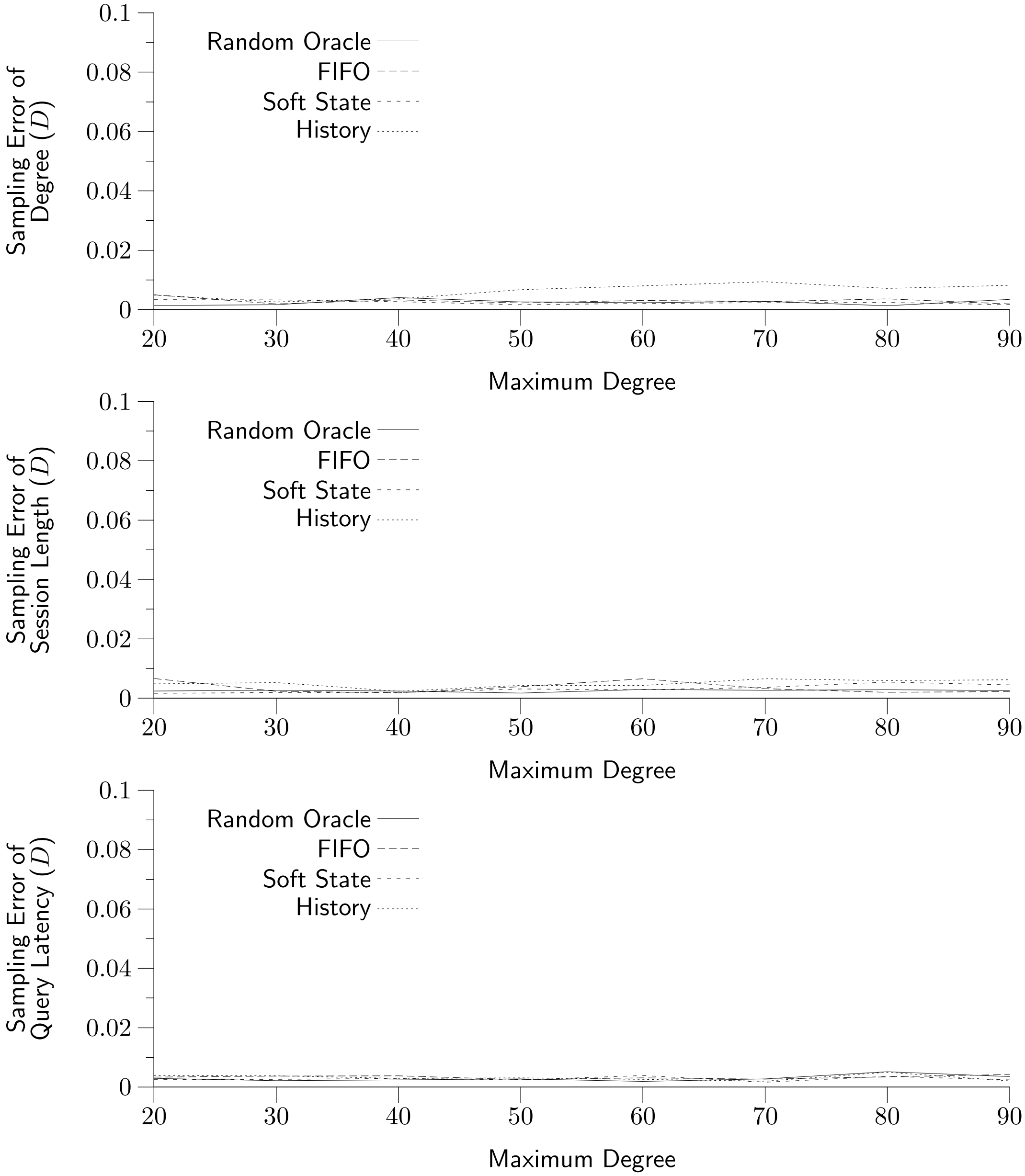


FIGURE 5.7: Sampling error of the three fundamental properties as a function of the maximum number of connections each peer will accept. Each peer actively attempts to maintain $x - 15$ connections.

could be used with existing measurement tools for measuring bandwidth to estimate the distribution of access link bandwidth in a peer-to-peer system. Listing 1 shows an example of using `ion-sampler` to sample peers from Gnutella.

5.5.2 Empirical Validation

Empirical validation is challenging due to the absence of high-quality reference data to compare against. In our earlier work [10, 11], we developed a peer-to-peer crawler called Cruiser that captures the complete overlay topology through exhaustive exploration. We can use these topology snapshots as a point of reference for the degree distribution. Unfortunately, we do not have reliably accurate empirical reference data for session lengths or query latency.

By capturing every peer, Cruiser is immune to sampling difficulties. However, because the network changes as Cruiser operates, its snapshots are slightly distorted [10]. In particular, peers arriving near the start of the crawl are likely to have found additional neighbors by the time Cruiser contacts them. Therefore, we intuitively expect a slight upward bias in Cruiser's observed degree distribution. For this reason, we would not expect a perfect match between Cruiser and sampling, but if the sampling

```
bash$ ./ion-sampler gnutella --hops 25 -n 10
10.8.65.171:6348
10.199.20.183:5260
10.8.45.103:34717
10.21.0.29:6346
10.32.170.200:6346
10.201.162.49:30274
10.222.183.129:47272
10.245.64.85:6348
10.79.198.44:36520
10.216.54.169:44380
bash$
```

Listing 1: Example usage of the `ion-sampler` tool. We specify that we want to use the Gnutella plug-in, each walk should take 25 hops, and we would like 10 samples. The tool then prints out 10 IP-address:port pairs. We have changed the first octet of each result to "10" for privacy reasons.

is unbiased we still expect them to be very close. We can view the CCDF version of the degree distribution captured by Cruiser as a close upper-bound on the true degree distribution.

Figure 5.8 presents a comparison of the degree distribution of reachable ultrapeers in Gnutella, as seen by Cruiser and by the sampling tool (capturing approximately 1,000 samples with $r = 25$ hops). It also includes the results of a short crawl,[7] a sampling technique commonly used in earlier studies (*e.g.*, [60]). We interleaved running these measurement tools to minimize the change in the system between measurements of different tools, in order to make their results comparable.

Examining Figure 5.8, we see that the full crawl and sampling distributions are quite similar. The sampling tool finds slightly more peers with lower degree, compared to the full crawl, in accordance with our expectations described above. We examined several such pairs of crawling and sampling data and found the same pattern in each pair. By comparison, the short crawl exhibits a substantial bias towards high degree

---

[7] A "short crawl" is a general term for a progressive exploration of a portion of the graph, such as by using a breadth-first or depth-first search. In this case, we randomly select the next peer to explore.

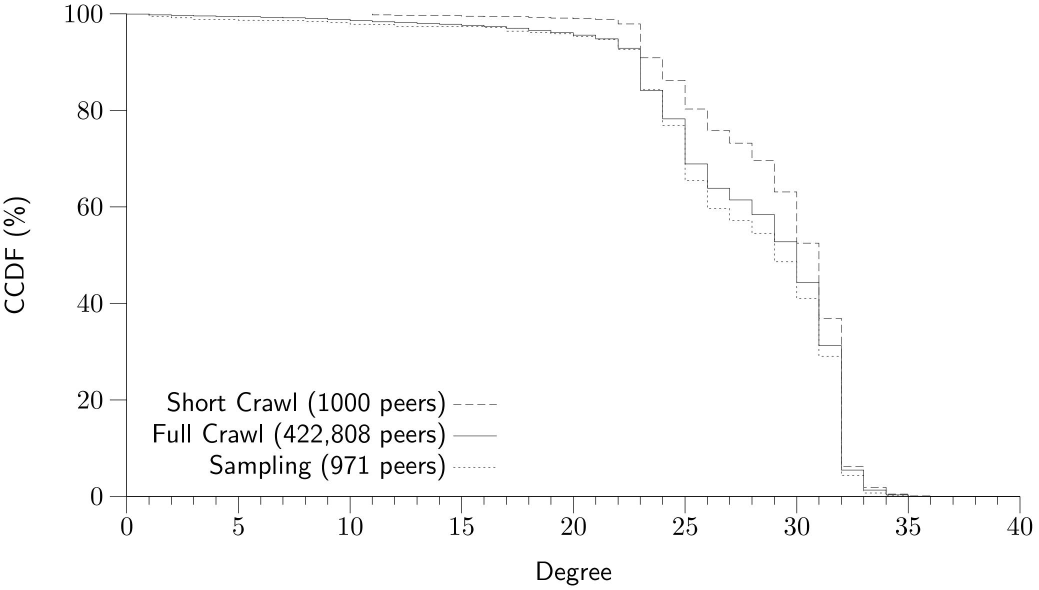


FIGURE 5.8: Comparison of degree distributions observed from sampling versus exhaustively crawling all peers

peers relative to both the full crawl and sampling. We computed the KS statistic (D) between each pair of datasets, presented in Table 5.3. Since the full crawl is a close *upper-bound* of the true degree distribution, and since sampling's distribution is *lower*, the error in the sampling distribution relative to the true distribution is $D \leq 4.3$. On the other hand, because the short crawl data *exceeds* the full crawl distribution, its error relative to the true distribution is $D \geq 12$. In other words, the true D for the sampling data is *at most* 4.3, while the true D for the short crawl data is *at least* 12. It is possible that sampling with MRWB produces more accurate results than a full crawl (which suffers from distortion), but this is difficult to prove conclusively.

Having demonstrated the validity of the MRWB technique, we now turn our attention to its efficiency. Performing the walk requires $n \cdot r$ queries, where n is the desired number of samples and r is the length of the walk in hops. If r is too low, significant bias may be introduced. If r is too high, it should not introduce bias, but is less efficient. From graph theory, we expect to require $r \geq O(\log |V|)$ for an ordinary random walk.

To empirically explore the selection of r for Gnutella, we conducted many sets of sampling experiments using different values of r, with full crawls interspersed between the sampling experiments. For each sampling experiment, we compute the KS statistic, D, between the sampled degree distribution and that captured by the most recent crawl. Figure 5.9 presents the mean and standard deviation of D as a function of r across different experiments. The figure shows that low values of r (≤ 10) can lead to enormous bias ($D \geq 40$). The amount of bias decreases rapidly with r, and low bias is observed for $r \geq 25$ hops. However, in a single experiment with $r = 30$

	Short Crawl	Full Crawl	Sampling
Short Crawl	—	12.0	16.1
Full Crawl	12.0	—	4.30
Sampling	16.1	4.30	—

TABLE 5.3: KS statistic (D) between pairs of empirical datasets

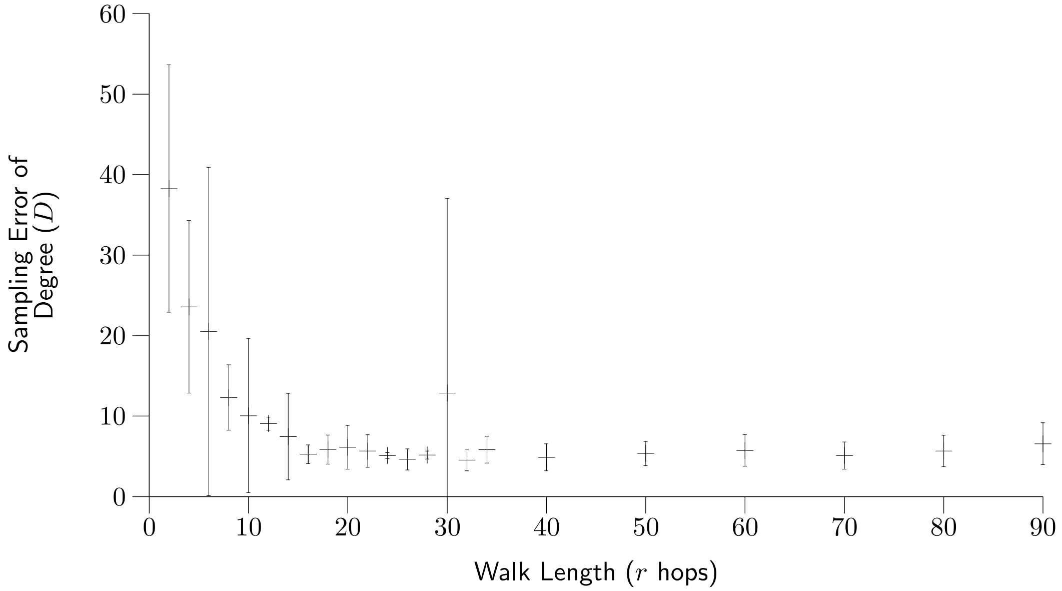


FIGURE 5.9: Difference between sampling and a crawl as a function of walk length. Each experiment was repeated several times. Error bars show the sample standard deviation.

hops, we observed $D > 30$, while all other experiments at that length showed $D < 9$. Investigating the anomalous dataset, we found that a single peer had been selected 309 out of 999 times.

Further examining the trace of this walk, we found that the walk happened to start at a peer with only a single neighbor. In such a case, the walk gets stuck at that peer due to the way Metropolis–Hastings transitions to a new peer y with probability only $\frac{\text{degree}(x)}{\text{degree}(y)}$. When this "stuck" event occurs late in the walk, it is just part of the normal re-weighting to correct for a regular random walk's bias towards high degree peers. However, when it occurs during the first step of the walk, a large fraction of the walks will end at the unusual low-degree peer, resulting in an anomalous set of selections where the same peer is chosen many times.

One way to address this problem is to increase the walk length by requiring

$$r \geq \frac{\text{maximum degree}}{\text{minimum degree}} \cdot \log |V|.$$

However, this reduces the efficiency of the walk. More importantly, we typically do not accurately know the maximum degree, *i.e.*, while increasing r decreases the probability of an anomalous event, it does not preclude it. Therefore, we suggest the following heuristic to prevent such problems from occurring. During the first few steps of the walk, always transition to the next peer as in a regular random walk; after the first few steps, use the Metropolis–Hastings method for deciding whether to transition to the next peer or remain at the current one. This modification eliminates the correlations induced by sharing a single starting location, while keeping the walk length relatively short. We are experimenting with this modification in our ongoing work. In any case, such anomalous data sets can easily be detected (and discarded) by the presence of the same IP address being selected a statistically improbable number of times. In light of these considerations, we regard a choice of $r = 25$ as a safe walk length for Gnutella. Choosing $r = 25$, we can collect 1,000 samples by querying 25,000 peers, over an order of magnitude in savings compared with performing a full crawl which must contact more than 400,000.

With respect to execution time, preliminary results show than an implementation of `ion-sampler` with $r = 25$ hops has execution time comparable using Cruiser to capture the entire network, or around 10 minutes. While `ion-sampler` contacts significantly fewer peers, walks are sequential in nature which limits the amount of parallelism that `ion-sampler` can exploit. As discussed earlier, while longer execution time has a negative impact on the accuracy of Cruiser's results, `ion-sampler`'s results are not significantly impacted by the time required to perform the walk (as demonstrated in Section 5.4.4 where we simulate walks of 10,000 hops). In our initial implementation of `ion-sampler`, a small fraction of walks would get "stuck" in a corner of the network, repeatedly trying to contact a set of departed peers. While the walks eventually recover, this corner-case significantly and needlessly delayed the overall execution time. We added a small cache to remember the addresses of unresponsive peers to address this issue.

In summary, these empirical results support the conclusion that a Metropolized Random Walk with Backtracking is an appropriate method of collecting measurements from peer-to-peer systems, and demonstrate that it is significantly more accurate than other common sampling techniques. They also illustrate the dramatic improvement in efficiency and scalability of MRWB compared to performing a full crawl. As network size increases, the cost of a full crawl grows linearly and takes longer to complete, introducing greater distortion into the captured snapshots. For MRWB, the cost increases logarithmically, and no additional bias is introduced.

5.6 Discussion

5.6.1 How Many Samples are Required?

An important consideration when collecting samples is to know how many samples are needed for statistically significant results. This is principally a property of the distribution being sampled. Consider the problem of estimating the underlying frequency f of an event, *e.g.*, that the peer degree takes a particular value. Given N unbiased samples, an unbiased estimate of f is $\widehat{f} = M/N$ where M is the number of samples for which the event occurs. $\widehat{f}$ has root mean square (RMS) relative error

$$\sigma = \sqrt{\mathsf{Var}(\widehat{f})}/f = \sqrt{(1-f)/fN}.$$

From this expression, we derive the following observations:

- Estimation error does not depend on the population size; in particular the estimation properties of unbiased sampling scale independently of the size of the system under study.
- The above expression can be inverted to derive the number of samples $N_{f,\sigma}$ required to estimate an outcome of frequency f up to an error σ. A simple bound is $N_{f,\sigma} \leq 1/(f\sigma^2)$.
- Unsurprisingly, smaller frequency outcomes have a larger relative error. For example, gathering 1,000 unbiased samples gives us very little useful information about events which only occur one time in 10,000; the associated σ value is approximately 3: the likely error dominates the value to be estimated. This motivates using *biased sampling* in circumstances that we discuss in the next subsection.

The presence of sampling bias complicates the picture. If an event with underlying frequency f is actually sampled with frequency f_0, then the RMS relative error acquires an additional term $(1 - f_0/f)^2$ which *does not reduce* as the number of samples N grows. In other words, when sampling from a biased distribution, increasing the number of samples only increases the accuracy with which we estimate the *biased* distribution.

5.6.2 Unbiased versus Biased Sampling

At the beginning of this chapter, we set the goal of collecting unbiased samples. However, there are circumstances where unbiased samples are inefficient. For example, while unbiased samples provide accurate information about the body of a distribution, they provide very little information about the tails: the pitfall of estimating rare events we discussed in the previous subsection.

In circumstances such as studying infrequent events, it may be desirable to gather samples with a *known sampling bias*, *i.e.*, with non-uniform sampling probabilities. By deliberately introducing a sampling bias towards the area of interest, more relevant samples can be gathered. During analysis of the data, each sample is weighted inversely to the probability that it is sampled. This yields unbiased estimates of the quantities of interest, even though the selection of the samples is biased.

A known bias can be introduced by choosing an appropriate definition of $\mu(x)$ in the Metropolis–Hastings equations presented in Section 5.3 and altering the walk accordingly. Because the desired type of known bias depends on the focus of the research, we cannot exhaustively demonstrate through simulation that Metropolis–Hastings will operate correctly in a dynamic environment for any $\mu(x)$. Our results show that it works well in the common case where unbiased samples are desired (*i.e.*, $\mu(x) = \mu(y)$ for all x and y).

5.6.3 Sampling from Structured Systems

Throughout this chapter, we have assumed an unstructured peer-to-peer network. Structured systems (also known as Distributed Hash Tables or DHTs) should work just as well with random walks, provided links are still bidirectional. However, the structure of these systems often allows a more efficient technique.

In a typical DHT scheme, each peer has a randomly generated identifier. Peers form an overlay that actively maintains certain properties such that messages are efficiently routed to the peer "closest" to a target identifier. The exact properties and the definition of "closest" vary, but the theme remains the same. In these systems, to select a peer at random, we may simply generate an identifier uniformly at random and find the peer closest to the identifier. Because peer identifiers are generated uniformly at random, we know they are uncorrelated with any other property. This technique is simple and effective, as long as there is little variation in the amount of identifier space that each peer is responsible for. We made use of this sampling technique in our study of the widely-deployed Kad DHT, in Chapter 9.

5.7 Summary

This paper explores the problem of sampling representative peer properties in large and dynamic unstructured P2P systems. We show that the topological and temporal properties of P2P systems can lead to significant bias in collected samples. To collect unbiased samples, we present the Metropolized Random Walk with Backtracking (MRWB), a modification of the Metropolis–Hastings technique, which we developed into the `ion-sampler` tool. Using both simulation and empirical evaluation, we show that MRWB can collect approximately unbiased samples of peer properties over a wide range of realistic peer dynamics and topological structures.

CHAPTER 6

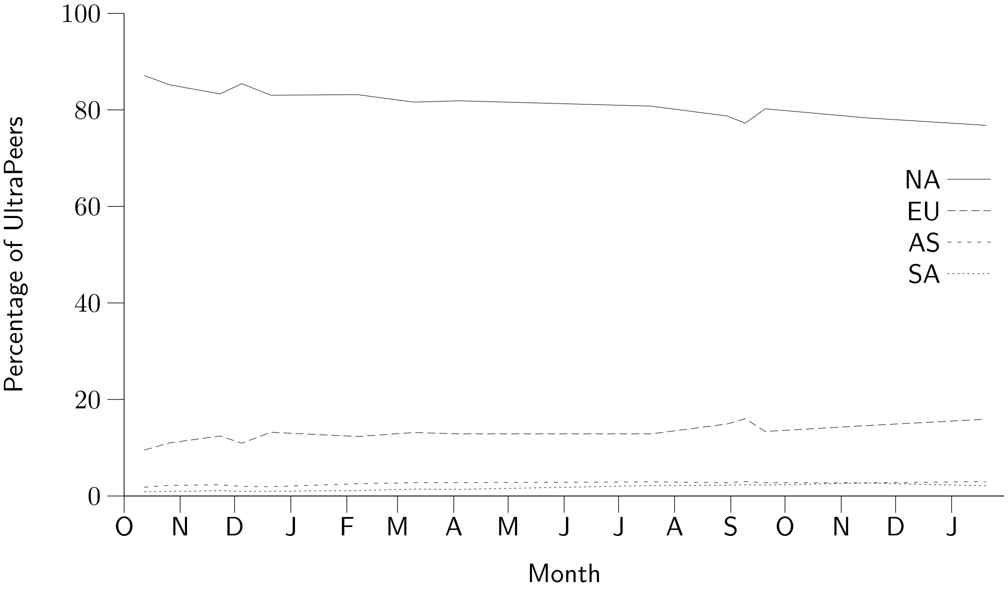
Static Peer Properties

Portions © 2006 IEEE. Reprinted, with permission, from A. Rasti, D. Stutzbach, and R. Rejaie, "On the long-term evolution of the two-tier Gnutella overlay," in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

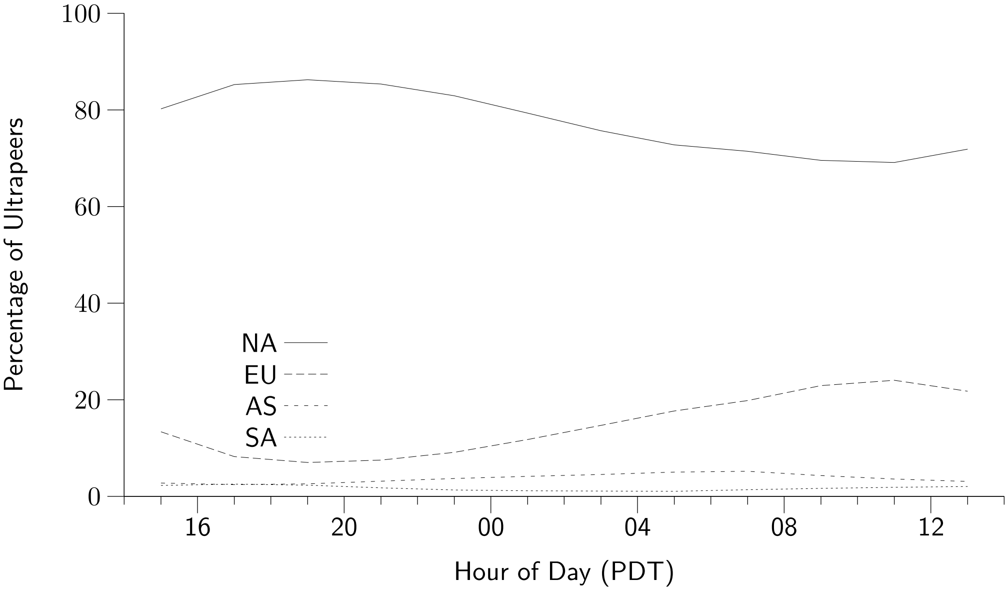
Static peer properties are characteristics of the peer independent of how they are connected to the rest of the overlay. This chapter will briefly examine the geographic distribution of peers, then focus on the file-sharing habits of peers.

6.1 Geography

We examine the geographic location of ultrapeers in Gnutella using snapshots captured by Cruiser. Using a large selection of captured snapshots, we examine how the geographic composition of ultrapeers changes with time-of-day (in Figure 6.1b), as well as how it varied over the course of one year (in Figure 6.1a). When examining changes over the course of a year, we selected snapshots taken from approximately the same time of day, 3pm Pacific Standard Time, to reduce noise caused by time-of-day effects.



(a) Oct. 2004–Jan. 2006



(b) Over 24 hours, Sep. 20–21, 2005

FIGURE 6.1: Breakdown of ultrapeers by continent

We examine the breakdown of ultrapeers across different regions and countries using GeoIP 1.3.14 from MaxMind, LLC. Figure 6.1a shows the distribution of Gnutella clients across four regions, namely North America (NA), South America (SA), Europe (EU), and Asia (AS) that collectively make up 98.5% of the total ultrapeer population. This figure reveals that a majority of Gnutella ultrapeers are in North American (80%) with a significant fraction (13%) in Europe. Furthermore, the user population of different regions have grown proportionally over time. The distribution of user populations across different countries has also grown proportionally, except for China where client population has dropped significantly (94%). Clients in US, Canada, and UK make up 65%, 14%, and 5% of the total population, respectively[1]. The remaining countries made up less than 2% each, but make up 16% in total. Thus, while the Gnutella network is dominated by predominately English-speaking countries, around one-fifth is composed of users from other countries.

Figure 6.1b shows how the geographic distribution changes over a 24-hour period,[2] demonstrating the effect of different time zones on the composition of user population. While clearly discernible, overall the effect is weak, with North American users forming the majority regardless of the time of day.

This section on the geographic location of peers is based on co-authored material with Amir Rasti and Prof. Reza Rejaie. The snapshots were collected by me, while Amir Rasti determined the geographic position of the peers and produced the graphs. The text was written jointly by myself, Amir Rasti, and Prof. Reja Rejaie. Portions of this material appeared previously in [17].

6.2 File Properties

During the past few years, a handful of previous studies have characterized the distribution of shared files in various P2P file sharing applications [57, 59, 60, 62].

---

[1] These values are from the snapshot taken on 9/20/05 and are similar to the other values observed during the study period, as shown in Figure 6.1a.

[2] We noticed that, the population of North American and European clients peak at around 7pm and 11am PDT with 86% and 24%, respectively. This figure indicates that the 3pm snapshots used in Figure 6.1a capture roughly average daily population, *i.e.*, not at any of the peaks.

While these studies shed an insightful light on the characteristics of files in file-sharing applications, they have several limitations. First, almost all the previous studies have focused on a subset of peer population in their target file-sharing applications (*i.e.*, less than 20k peers). To our knowledge, none of these studies have verified whether the captured peers are indeed representative of the entire population. Second, many of the previous studies (except [57, 62]) are more than three years old and thus rather outdated. During the past few years, P2P file-sharing applications have significantly grown in size and have incorporated new features. In particular, most popular file sharing applications have adopted a two-tier architecture along with new search mechanisms (*e.g.*, dynamics querying in Gnutella) to improve their scalability.

In this section, we empirically characterize available files across reachable peers in the modern Gnutella network. We present two new measurement methodologies to capture an accurate snapshot of the system at a particular point of time. Using Cruiser, we have captured more than 50 snapshots of the files available in Gnutella that span over one year and each snapshot contains more than 100 million distinct files. Using these snapshots, we examine properties of contributed resources (*i.e.*, files and storage space) by participating peers. We also leverage our unbiased sampling tool, `ion-sampler`, for P2P systems developed in Chapter 5 to select a representative subset of peers. We use `ion-sampler` to validate the characteristics derived from complete snapshots captured by Cruiser.

Our main findings can be summarized as follows:

- Free riding has significantly decreased among Gnutella users during the past few years and is significantly lower than in other P2P file-sharing applications such as eDonkey.
- The number of shared files and contributed storage space by individual peers are heavily skewed.
- The popularity of individual files is also heavily skewed, with a handful of files being extremely popular while a majority of files are very unpopular.
- The most popular file type is the MP3 file, which accounts for over two-thirds of all files and one-third of all bytes. Both the popularity and occupied space by video files has tripled over the past few years. Furthermore, the number

of video files are less than one-tenth of audio files but they occupy 25% more bytes. Overall, 93% of bytes in the system are occupied by audio and video files.

This section is based on co-authored material under submission to a journal, which includes material previously presented at a conference [15]. The experimental work regarding was performed by Shanyu Zhao under the direction of Reza Rejaie and myself. The associated text was written by myself and Reza Rejaie.

6.2.1 Related Work

We are aware of only two other studies that focus on the characteristics of files shared by users. First, Fessant *et al.* [62] examined characteristics of available files, using data collected from 12,000 eDonkey clients over a three day period in 2003. They show that the popularity of files stored in file-sharing systems is heavily skewed, following a Zipf distribution. They also examine correlations in file preferences. When two peers have 10 files in common, there's an 80% chance they have at least one more file in common. The probability is close to 100% if they have at least 50 files in common. Second, Liang *et al.* [57] recently analyzed the nature and magnitude of deliberately corrupted files ("pollution") in Kazaa. To combat P2P sharing of copyrighted content, some companies intentionally inject decoy files, which have the same file name as a popular song. Liang *et al.* developed a multi-threaded crawler that queries all 30,000 Kazaa super-nodes for seven popular songs over the course of one hour. They show that the popularity of different versions of a song also follows a Zipf distribution. For most of the seven popular songs, over 50% of the copies are polluted.

A few other studies have examined the files shared by users as part of broader measurement studies on peer-to-peer systems. In 2001, Chu *et al.* [59] studied peer churn and the distribution of file popularity. They found that file popularity follows a log-quadratic distribution (which can be thought of as a second-order Zipf distribution). Saroiu *et al.* [60] examined many characteristics of peers in Napster and

Gnutella, such as their bottleneck bandwidth, latency, uptime, and number of shared files in 2001. They found that the number of shared files was heavily skewed.

Our study differs from the few previous studies on the shared files in P2P systems primarily by using a significantly larger population of peers (more than 1 million concurrent peers). Additionally, we make use of unbiased sampling for validation whenever possible.

Another group of studies passively captured P2P traffic at a router to characterize exchanged files among peers. Gummadi *et al.* [45] analyzed a 200-day trace of Kazaa traffic collected at the University of Washington, demonstrating that file transfers in Kazaa do not follow a Zipf distribution and argued that this difference is due to the "fetch-at-most-once" nature of downloads in file-sharing applications. Another analysis of Kazaa traffic was conducted by Leibowitz *et al.* [47] at a large Israeli ISP. They examined the changing popularity of exchanged files among peers and showed that the data-sharing graph exhibits small-world properties. Note that the pattern of exchanged files among peers affects the characteristics of shared files throughout the system, but is subject to shorter-term trends. In contrast, the files shared by a peer may be the result of transfers over the course of months or years, followed by a gradual pruning of unwanted files. In summary, these studies on exchanged files are closely related and complement our work.

6.2.2 Measurement Methodology

Our goal is to capture a *snapshot* of the Gnutella network at a given point of time that contains *(i)* all participating peers and *(ii)* the available files at each peer in the overlay (a *file snapshot*). In essence, we need to capture snapshots of the overlay topology and annotate each peer with its available files. A common approach to capture a snapshot is to use a P2P crawler.

Previous studies adopt one of the following ad-hoc sampling schemes to capture a partial snapshot of a P2P system: *(i) Partial Snapshot Through a Short Crawl*: Some studies periodically capture a small subset of participating peers (*i.e.*, a partial snapshot) through a short crawl [60], *(ii) Periodic Probe of a Fixed Group*: Other studies identify a subset of some participating peers (using a partial crawl or passive monitoring) and periodically probe the same group of peers to collect information about their available files [59]. In the absence of any solid understanding of churn and file characteristics in P2P systems, it is not clear whether these sampling strategies capture a representative population of peers. In Chapter 5, we showed that these ad-hoc sampling techniques could lead to significant bias towards the properties of the short-lived or high degree peers [16].

We developed the following two measurement methodologies to capture lists of shared files in the Gnutella network: *(i)* Capturing complete snapshots, and *(ii)* Capturing unbiased samples. We describe these measurement approaches in the following subsections. Capturing complete snapshots is our primary characterization technique and we use unbiased sampling for validation.

6.2.3 Capturing Complete Snapshots

Our goal is to capture the entire population of participating peers in the Gnutella network (*i.e.*, a complete snapshot) within a short period to minimize any potential bias in our characterization. The time required to obtain the list of available files at individual peers is significantly longer than for obtaining neighbor information. For example, obtaining a list of neighbor peers from a peer may take less than a second whereas the list of available files may take several minutes to download if the peer has many files. In summary, a topology crawl is significantly faster than a file crawl. Therefore, we decouple topology and content crawls to improve the accuracy of captured snapshots and conduct our snapshots in three steps as follows: First, we conduct a *topology crawl* to quickly capture all participating peers and their pairwise connectivity, *i.e.*, capturing the overlay topology. Second, we conduct a *content crawl* and collect the list of files available at each one of the peers identified during the topology crawl. Third, once the content crawl is completed, we conduct another topology crawl in order to identify those long-lived peers in the initial snapshots that remained in the system during the entire content crawl. At the end of these three steps, we create a snapshot of the overlay topology where each node is annotated with its available file and a label that describes whether it is long- or short-lived. Since some of the captured peers in the first topology crawl depart the system during the content crawl, the collected content in our measurement is slightly biased towards peers with longer uptime. This motivates our second technique, unbiased sampling, described in the next subsection.

Cruiser can perform a topology crawl for more than two million Gnutella peers in less than 15 minutes, and perform a content crawl within 5.5 hours, *i.e.*, capturing the annotated snapshot takes 6 hours, (15min + 5.5hr + 15min). During the content crawl, Cruiser collects the file name and content hash (SHA1) for each shared file on every reachable peer, resulting in a log file more than 10 GB in size.

6.2.4 Capturing Unbiased Samples

Capturing complete snapshots is a heavy-weight operation that provides a tremendous amount of information. However, even with a fast crawler, there will inevitably be some distortion as the network changes while the crawler operates. Sampling is a useful, light-weight alternative that does not suffer from distortion. However, commonly-used sampling techniques for measuring peer-to-peer systems tend to introduce considerable bias. We made use of the `ion-sampler` tool developed in Chapter 5 to collect unbiased samples. The tool allows us to gather many samples at once by running many walks in parallel. For this study, we use 1,000 parallel walks. We use `ion-sampler` to select peers, then gather the list of files available from each selected peer.

While sampling does not suffer from distortion, it provides fewer details than a full crawl, particularly about unusual peers, *e.g.*, in the tail of distributions. For these reasons, we employ sampling to validate the results from complete snapshots.

6.2.5 Dataset

We captured more than 50 snapshots of the Gnutella network, annotated with the list of files available at each peer. Our snapshots consists of daily snapshots from consecutive days during the following intervals: 6/8/2005–6/18/2005, 8/23/2005–

9/9/2005 and 10/11/2005–10/21/2005.[3] Furthermore, we have collected 24 snapshots, roughly once per week between 11/29/2005 and 05/30/2006.[4] To avoid any time-of-day effects, each snapshot was captured at approximately the same time-of-day (2pm PDT). These snapshots enable us to examine characteristics of available files in the system over both short and long timescales (*i.e.*, several days and several months). Additionally, we use unbiased samples collected on June 16th, 2006 and July 4th, 2006 to validate our finding in Section 6.2.7.

Figure 6.2a depicts the number of ultrapeers and leaf peers across captured snapshots during our measurement period. A subset of peers in each snapshot are unreachable by our crawler for one of the following reasons: *(i)* firewall or NAT blocking incoming traffic, *(ii)* severe network congestion or overloaded processor at the peer, *(iii)* peer departure, or *(iv)* the peer does not support the Browse Host protocol. Figure 6.2b shows the percentage of ultrapeers, leaf peers, and the total population in individual snapshots that were successfully contacted by the file crawler. This figure shows that in average around 50% of ultrapeers, 12% of leaf peers and 18% of the total population are reachable by the file crawler. Table 6.1 presents the minimum and maximum value for the percentage of peers within each group (ultrapeer, leaf, total population) for four different reasons across all snapshots. “TCP Refused” refers to attempts that were rejected by a TCP RST message. “Timeout” refers to attempts that did not receive any response. “Conn. Lost” refers to attempts that established a

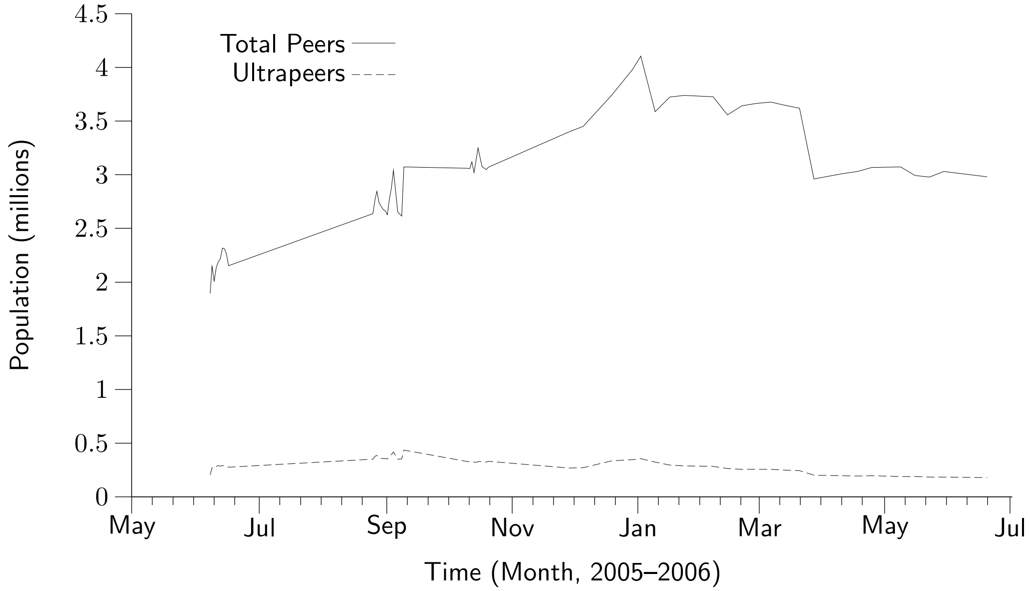
Type	TCP Refused	Timeout	Conn. Lost	App. Refused
Ultrapeer	31.30%–46.10%	1.74%–13.25%	1.78%–6.80%	0.97%–2.30%
Leaf Peers	53.53%–72.13%	1.12%–3.85%	4.24%–23.16%	1.27%–3.06%
All Peers	31.63%–46.74%	1.75%–13.03%	1.81%–7.08%	0.97%–2.30%

TABLE 6.1: The range of percentage of peers that failed during the file crawl for various reasons across all snapshots

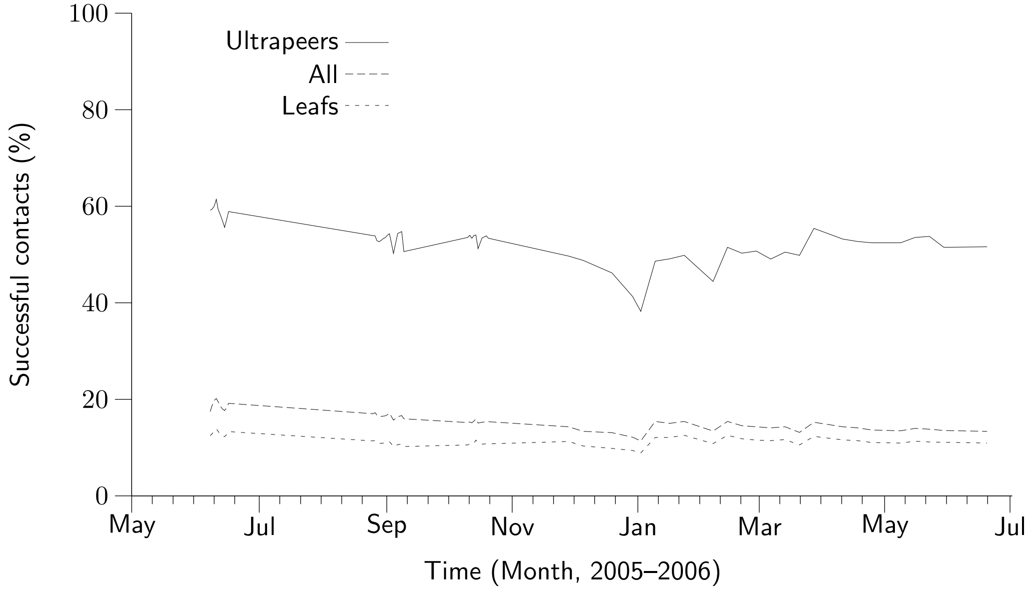
---

[3] We are missing snapshots for 9/6/2005, 9/8/2005, 10/17/2005, and 10/29/2005 due to a mixture of software, network, and power failures.

[4] The detailed characteristics of individual snapshots are available online at `http://mirage.cs.uoregon.edu/P2P/files/`.



(a) Number of Ultrapeers & Total Peer across individual snapshots



(b) Percentage of successfully contacted peers among ultrapeers, leaves and total peers

FIGURE 6.2: Population and reachability across captured snapshots

TCP connection, but the connection closed before the file list could be fully retrieved. Finally, "App. Refused" refers to attempts where TCP connected, but the request for the list of files was rejected at the application level. In a few cases, our crawler machines were under heavy load by other users during our crawls, leading to a large percentage of timeouts and lost connections. In the typical case, these values were only a few percent. Since ultrapeers are not allowed to be firewalled by the Gnutella protocol, any reported connection error for ultrapeers indicates that the contacted peer has departed. However, connection errors for leaf peers might occur due to peer departure or a firewall.[5] In Chapter 7, we will show that about half of all leaf peers captured in a topology crawl leave the overlay within a 5 hour period. Independent online statistics [105] report that around 70% of leaf peers in the Gnutella network are firewalled. These evidences support the accuracy of the high ratio of connection errors that we experienced for leaf peers (in the column labeled "TCP Refused"). In summary, while we aim to capture complete snapshots, we can successfully contact only 20% of all peers in one snapshot (around half a million peers) to obtain their list of available files primarily due to two reasons: *(i)* the number of departed peers during the long file crawl, and *(ii)* the large number of leaf peers behind firewalls.

6.2.6 Challenges and Problems

We briefly discuss several interesting problems that we experienced during data collection and data processing.

Low-bandwidth TCP Connection: Although Cruiser has a timeout mechanism that closes any idle connections after 20 seconds, we noticed that some crawls do not complete after the crawling queue becomes empty. Further examinations revealed that around 80 peers in each crawl send their data at an extremely low rate (around 20 bytes per second) which prevents Cruiser from closing their connections. We instructed Cruiser to terminate a crawl a few minutes after its crawling queue becomes empty. Given the negligible number of these misbehaved peers, this does not significantly affect our analysis.

---

[5] We are not aware of any reliable technique to distinguish between these two scenarios.

File Identity: We use the content hash of a file returned by the target peer to uniquely identify individual files. In our initial measurements, we observed many files with the same name but different content hashes (*e.g.*, setup.exe, login.bmp). This illustrates that the trimmed (or even complete) filenames used by previous studies (*e.g.*, [59]), are not reliable file identifiers. We discovered around 3,500 files without a content hash value in each snapshot and eliminated them from our analysis.

Post-processing: To compute the popularity of individual files in the system, we needed to keep track of more than 100 million distinct files discovered in each snapshot. The large number of files resulted in memory bottlenecks in our analysis. We leveraged the skewed distribution of popularity to address this problem as follows. We divide captured peers in a snapshot into seven segments where each segment contains a random subset of the peers. Then, we calculate the popularity of files within each segment, trimming all files with fewer than 10 copies in a segment, and combine all the trimmed segments into a single snapshot. This approximation eliminated several million distinct files and prevented memory bottlenecks during our post-processing. While this prevents us from performing analysis on the least popular files (with fewer than 70 copies in the entire network), it does not significantly affect conducted analysis on more popular files.

6.2.7 Empirical Observations

6.2.7.1 Ratio of Free Riders

The success of P2P file sharing systems depends on the willingness of participating peers to share their files. Previous studies have frequently reported that participating peers do not have an incentive to contribute their resources (*e.g.*, disk space and network bandwidth) to the system and thus only use resources offered by others, *i.e.*, become *free riders*. Adar *et al.* [54] report that 66% of Gnutella peers were free riders in 2000, while a study by Saroiu *et al.* in 2002 [60] found 25% were free riders, with 75% of peers sharing less than 100 files. A recent study also report 68% of peers are free riders in eDonkey [62].

Table 6.2 presents the range (minimum–maximum values) for the number of ultrapeers (first row), leaf peers (second row) and total peers (last row) across all snapshots. We further divide ultrapeers (row 3 and 4) and leaf peers (row 5 and 6) into short-lived and long-lived based on their presence in the second topology crawl as we discussed in Section 6.2.2. These grouping reveal any potential differences in the free riding between ultrapeers and leaves as well as long- versus short-lived peers. For each one of the above groups, the corresponding row in Table 6.2 presents the range of the following properties across all snapshots: *(i)* the number of contacted peers that provided their sharing list (2^{nd} column), *(ii)* the percentage of free riders (3^{rd} column), and *(iii)* the average number of shared files among peers in each group.

Table 6.2 shows several interesting points as follows. The percentage of free riders in Gnutella has significantly dropped from 25% in 2002 [60] to around 13% among all participating peers (*i.e.*, last row) and is drastically lower than the 68% recently reported in eDonkey [62]. We speculate that several factors have contributed in the observed drop in the percentage of free riders including the increase in access link bandwidth for average Internet users and active marketing efforts by the Gnutella vendors encouraging their users to share. Table 6.2 reveals that the percentage of free riding among ultrapeers (9.0–10.5%) is somewhat lower than that in leaf peers (10.8–15.9%). However, on average leaf peers share more files. Since leaf peers constitute a

	Contacted Peers	**Free Riders(%)**	**Files/Peer**
Ultra	94,329–188,107	8.96–10.54	324–358
Leaf	253,164–395,682	10.81–15.85	347–406
Long-lived Ultra	69,441–147,435	9.27–10.78	320–354
Short-lived Ultra	24,888–40,672	8.08–9.77	335–371
Long-lived Leaf	166,241–240,969	11.39–16.93	369–437
Short-lived Leaf	86,923–154,713	9.93–13.79	316–358
Total	410,252–552,702	10.63–13.41	340–393

TABLE 6.2: Percentage of free riders, its breakdown across different groups, and the mean number of shared files among peer

larger portion of the total population (86–90%), their behavior has a greater impact on system performance. Long-lived peers appear to have a slightly higher percentage of free riders compared to short-lived peers. The correlation between lifetime and number of files shared is inconclusive. Long-lived leaf peers share more files than short-lived leaf peers, but long-lived and short-lived ultrapeers share about the same number of files.

6.2.7.2 Degree of Resource Sharing Among Cooperative Peers

We now focus our attention on cooperative peers and characterize their willingness to contribute their resources (*i.e.*, both files and storage space). During our analysis, we noticed that the sharing lists of many peers contain duplicate files. This occurs because most Gnutella clients simply put various folders with potentially duplicate content under the sharing folder. We have excluded all duplicate files from captured sharing lists (which account for around 10 million files or roughly 9% of all captured files) in our results.

Presenting these distributions as histograms results in a "messy", difficult-to-interpret tail of the distribution, due to the small number of peers in that region. This in turn makes it difficult to properly examine the slope of distribution. Presenting the distribution as a Complementary Cumulative Distribution Function (CCDF) yields a smooth tail without losing any information while retaining the linear slope that a power-law distribution exhibits on a log-log scale (since $\int_x^\infty u^{-\alpha} du \propto x^{-\alpha}$). Toward this end, we compute the CCDF of the number of shared files and shared bytes per peer for individual snapshots. To present the variability of these distributions across all snapshots in a compact and clear fashion, we compute the MIN-CCDF and MAX-CCDF, defined as follows:

$$\begin{aligned} \text{MIN-CCDF}(x) &= \min(\text{CCDF}_i(x) \text{ for all snapshots } i) \\ \text{MAX-CCDF}(x) &= \max(\text{CCDF}_i(x) \text{ for all snapshots } i) \end{aligned}$$

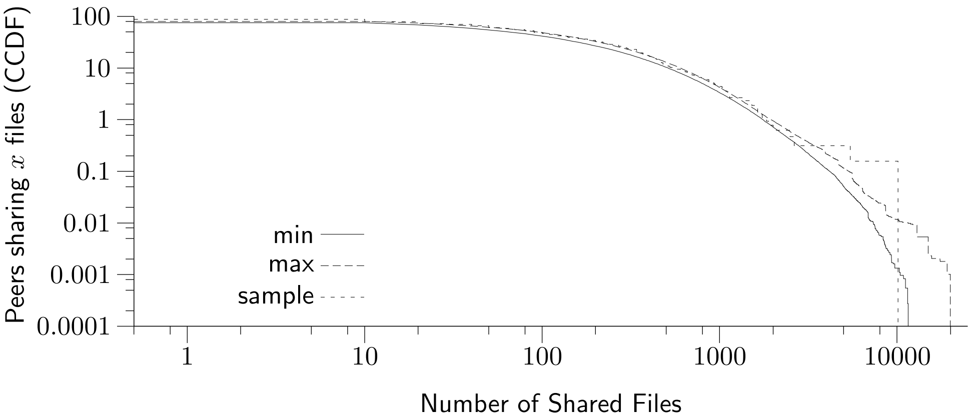
Figures 6.3a and 6.3b present the MIN-CCDF and MAX-CCDF of shared files and shared bytes across all snapshots, *i.e.*, the CCDFs from all the snapshots fall in between the MIN-CCDF and MAX-CCDF curves in the corresponding graph. Figures 6.3a and 6.3b also include the CCDF for shared files and shared bytes per peer based on samples collected with `ion-sampler`, to verify whether the results from complete snapshots are significantly affected by distortion or not (as described in Section 6.2.2). Note that the MIN-CCDF and MAX-CCDF curves in both figures are very close which indicates that the the CCDF for shared files and shared bytes per peer across all snapshots are very stable. Furthermore, the CCDFs from sampling are consistent with the results from complete snapshots, except in the tail of the distribution where the precision of sampling is low.

To test these distributions for power-law behavior, we recall that power-law distributions are *scale-free*. This means that the “slope” (α) of distribution is consistent over many orders of magnitude on a log-log scale in the tail of the distribution. While the histograms presented in the conference version of this work appear approximately linear [15], it is not possible to closely fit a line over the entire range of x values due to the subtle downward curvature.[6] This curvature is more apparent in the CCDFs in Figures 6.3a and 6.3b. Based on these considerations, the distributions of the number of files and bytes shared by individual peers are heavily skewed but do not appear to follow a power-law. Most peers share a moderate amount of files (or bytes) while a few peers contribute an enormous amount of content (or space). The median value for shared files is around 70 files while 0.01% of peers share more than 7,500 files. The median value of shared space is around 650 MB while 0.1% of peers contribute more than 85 GB.

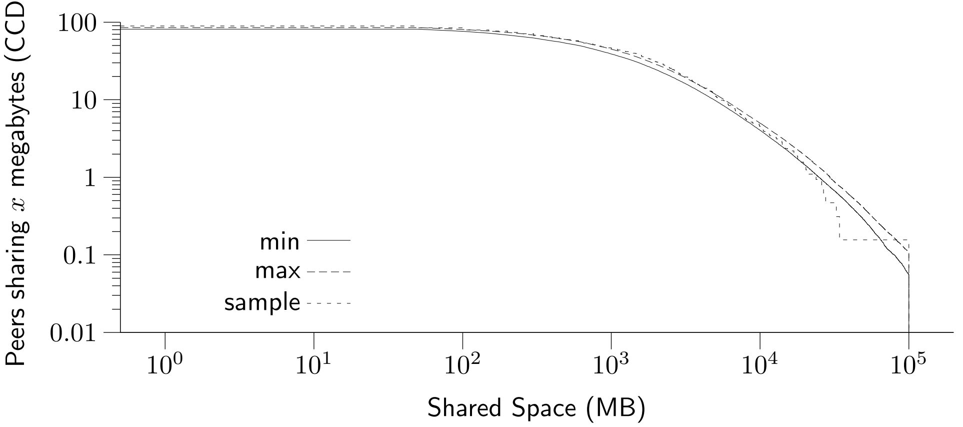
Correlation between Contributed Files and Bytes: Saroiu *et al.* [60] reported a strong correlation between the number of shared files and the volume of shared data across Gnutella peers in 2002. Figure 6.4a shows this correlation as a scatter-plot across all cooperative peers in one snapshot (June 13$^{\text{th}}$, 2005). Each point in this figure represents the number of shared files versus the shared disk space for each cooperative peer. We observe a greater variability than what has been reported by Saroiu *et al.* in 2002. However, this may simply be due to the larger number of data

---

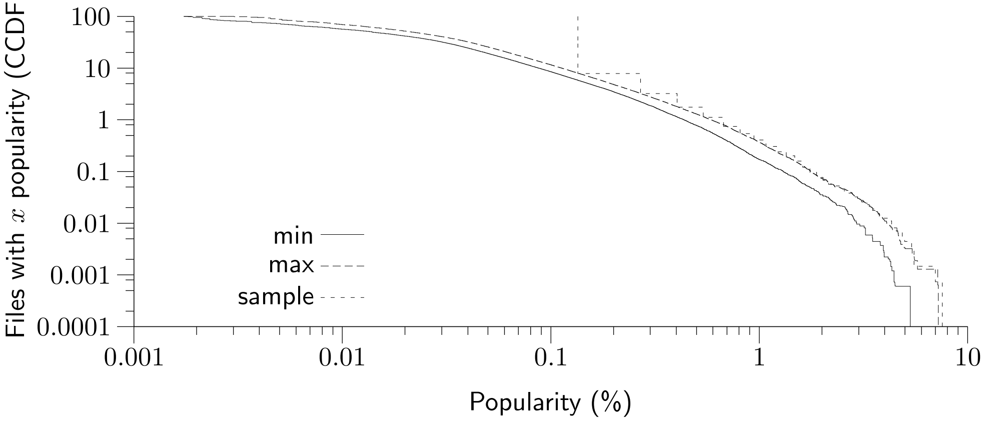
[6] We also found that it was not possible to closely fit a shifted Pareto distribution to any of these three distributions.



(a) Distribution of number of shared files across peers



(b) Distribution of number of shared bytes across peers



(c) Distribution of file popularity across unique files

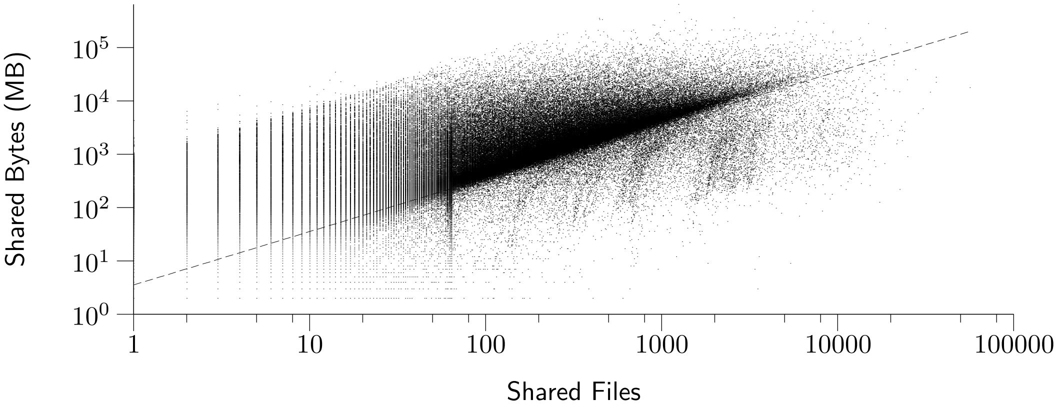
FIGURE 6.3: Distributions of files and bytes

points in our scatter-plot. To verify this issue, we sampled one out of every hundred data points in Figure 6.4a and the selected samples are shown in Figure 6.4b. Figure 6.4b reveals a strong correlation between the number of files and the number of bytes contributed by each peer which was obscured in Figure 6.4a. We performed a least-squares fit to a line through the origin (0 files = 0 bytes), finding a slope of 3.544 MB per file. This is consistent with the size of a MP3 file that is a few minutes long (*i.e.*, a typical song). Figure 6.4c shows the correlation between the contributed bytes and files by each peer using the data gathered by `ion-sampler` on June 16th, 2006—more than a year after the presented snapshot data in Figure 6.4a. Figure 6.4c fits to the same line and is consistent with the results from complete snapshots.

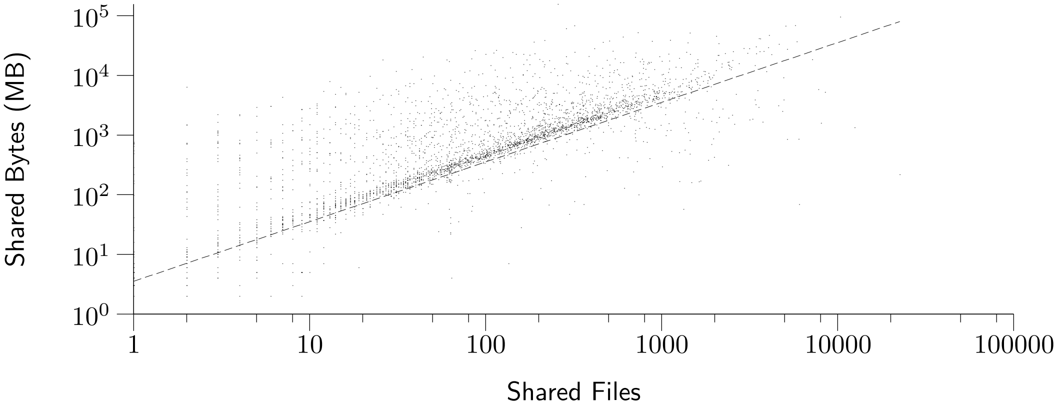
To gain a deeper insight about the correlation between contributed files and bytes, we derive the CCDF distribution of file size among all files in individual snapshots. Figure 6.5a depicts the CCDF distribution of file size in two snapshots that are six months apart in a linear-log scale. Figure 6.5a shows that the distribution of file size has changed very little between the two snapshots that are six months apart. This figure also indicates that 60% of all files are a few megabytes in size (*i.e.*, the graph has a knee between 1 MB and 10 MB). Furthermore, the majority of other files (around 40%) are smaller than 1 megabyte in size. Figure 6.5b shows the same CCDFs for file size distribution of two snapshots on a log-log scale. This figure illustrates that a very small portion of files (around 0.1%) have size larger than 1 gigabyte.

6.2.7.3 File Popularity Distribution

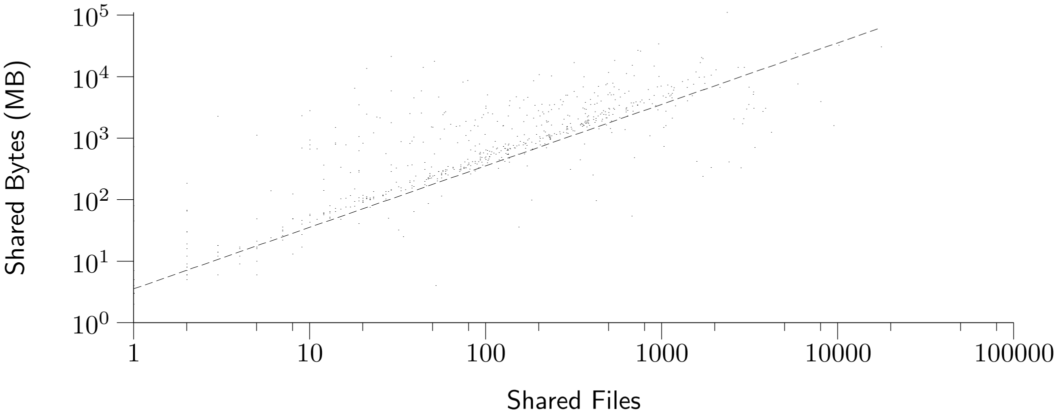
The distribution of popularity for individual files throughout the system is an important property that shows the degree of similarity and thus availability of individual files among participating peers. Chu *et al.* [59] showed that the file popularity follows a log-quadratic distribution, which can be viewed as a second-order power-law distribution, among Gnutella peers in 2001. Furthermore, Fessant *et al.* [62] recently reported a Zipf distribution for the file popularity in eDonkey. However, none of these studies have captured a large number of peers.



(a) All peers from a snapshot

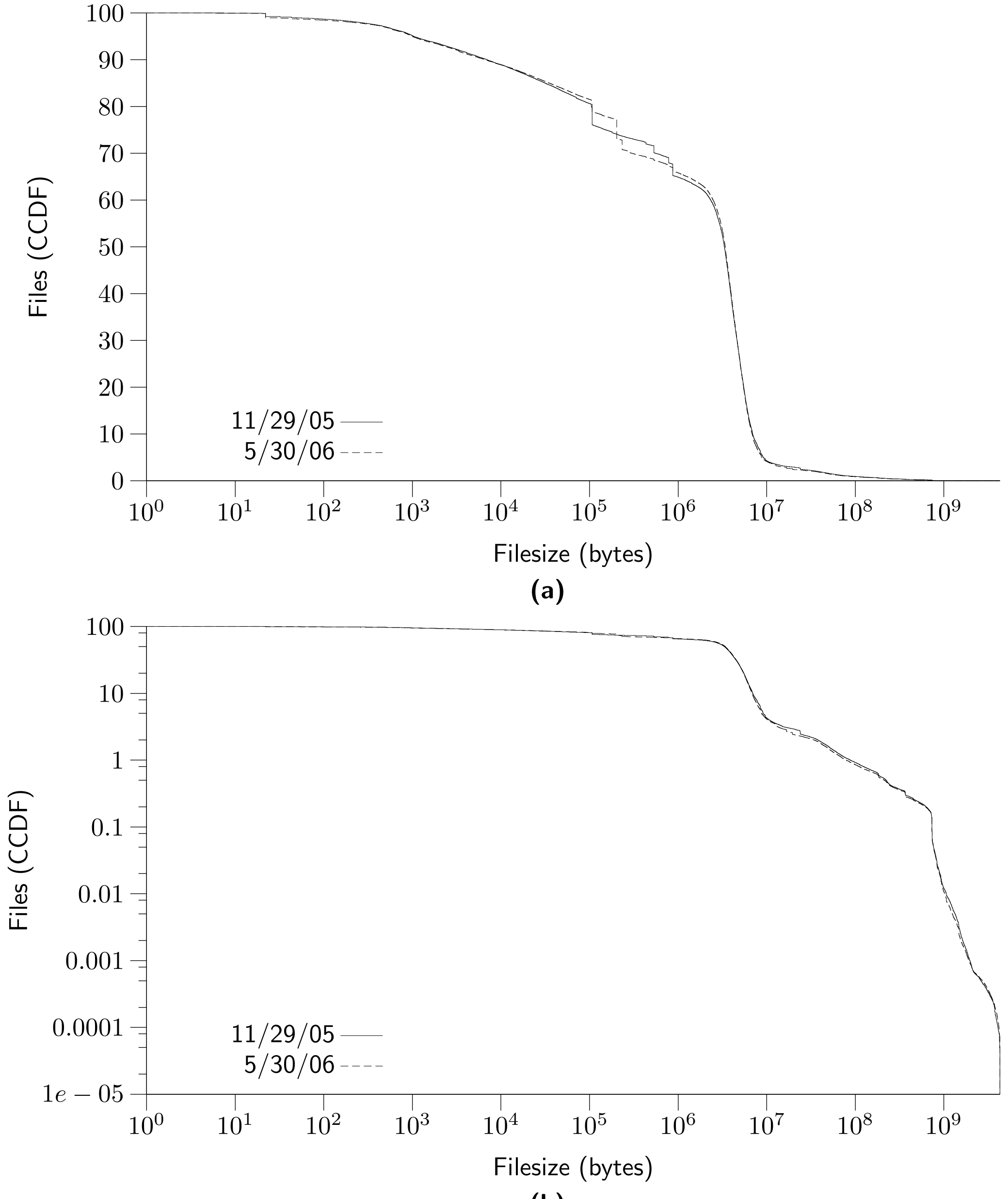


(b) One in every 100 peers from a complete snapshot



(c) All peers captured via sampling

FIGURE 6.4: Scatter-plot of number of shared bytes as a function of number of shared files. Each point represents one peer. The reference line is 3.544 MB per file.



(a)

(b)

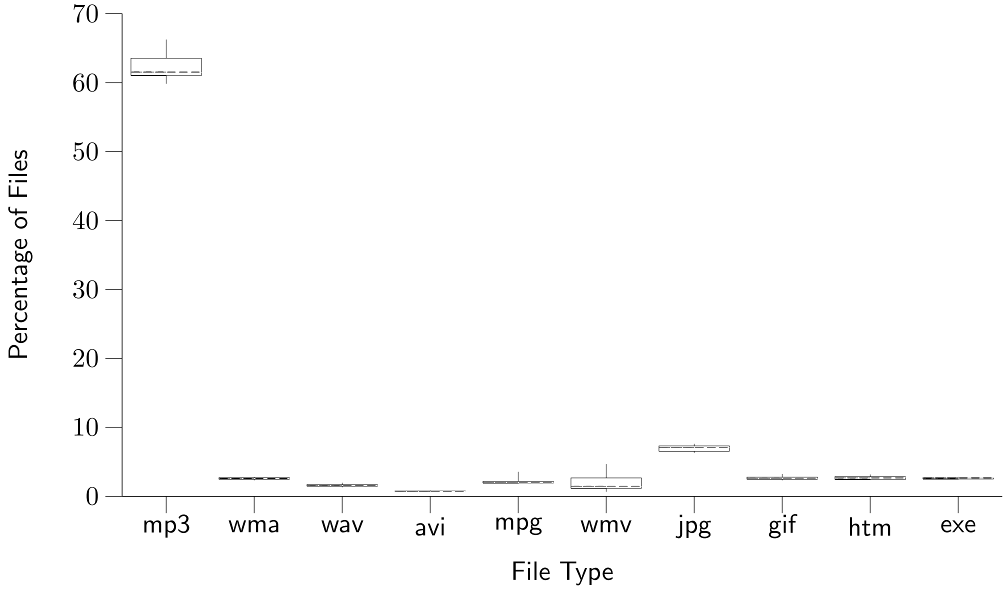
FIGURE 6.5: Distribution of filesize across all files

Figure 6.3c shows the range of CCDF for file popularity as a function of its rank (in log-log scale) among cooperative peers across all snapshots. Each snapshot typically contains more than 800 terabytes worth of content in more than 100 million unique files based on information from 0.4 million peers, constituting 18.5% of identified peers. If we assume that unreachable peers have similar profiles, the volume of available content in the Gnutella network is around 4,400 terabytes. Figure 6.3c illustrates two points: *(i)* file popularity is heavily skewed, and *(ii)* the distribution of file popularity has remained very stable across the eleven month measurement period. A small number of files are very popular while for most files only a few copies are available in the network. Again, we also present data collected using unbiased sampling, which closely matches the snapshot data except for very unpopular files where sampling has poor precision.

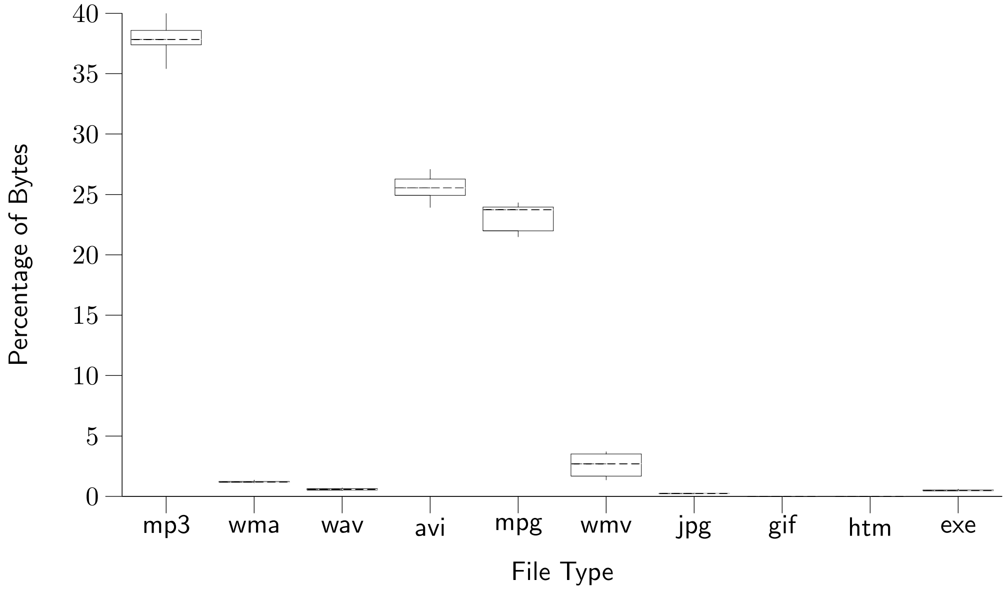
6.2.7.4 File Type Analysis

We have also examined the distribution of available files among Gnutella peers across different types of video and audio formats. This basically illustrates what types of content are available in the system and thus exchanged among peers. Chu *et al.* [59] conducted similar analysis for Gnutella peers in 2001 and reported that audio files constitute 67.2% of files and 79.2% of bytes. However, video files were significantly less popular and only contributed 2.1% of files and 19.1% of bytes. Using our snapshots, we analyze the various types of audio and video files based on file extensions.

Figure 6.6a and 6.6b depict the contribution of the most common file types in the total number of files and the total number of bytes across cooperative peers, respectively. As box-and-whisker plots, these figures present the variations of the contribution by different file types across all snapshots by showing the 75th percentile (the top of the box), 25th percentile (the bottom of the box), the median value (the line through the box), the maximum value (the top of the line) and the minimum value (the bottom line). The distribution of file types across all snapshots is fairly consistent with respect to both files and bytes (*i.e.*, all boxes are small). Note that except for MP3 files, each file types individually make up at most a few percent of all



(a) Contributed files per file type



(b) Contributed bytes per file type

FIGURE 6.6: Breakdown of file types based on the contributed number of files or bytes by each file type

files. MP3 audio files are significantly more popular than any other file type, making up two-thirds of all files and occupying more than one third of all disk space in the system.

Although non-media files (jpg, gif, htm, exe, txt) are among the top ten most popular types, audio and video files collectively occupy more than 93% of bytes and make up more than 73% of all files in the system. These figures also reveal that video files (avi, mpg, wmv) are significantly larger than other file types in the system. The audio files (MP3, wma, wav) account for 67% of files and 40% of bytes whereas video files constitute around 6% of files but 52.5% of bytes among Gnutella peers (*i.e.*, most files are audio files, but most bytes are in video files). Comparing to the reported results by Chu at al. in 2001 [59], video files have become three times more popular and occupy almost three times more space in the system.

We are unable to apply our sampling technique for file type analysis because it samples random *peers* not random *files*. Consequently, analyzing the file data gathered with `ion-sampler` shows significant correlations based on the preferences of the peers collected. For example, in one instance we saw that 16% of files had the jpg extension, due to sampling two peers with large jpg archives.

6.2.8 Summary

This section presented a measurement-based characterization of available files in the Gnutella file sharing network. We discussed the challenges in capturing an accurate snapshot of available files in P2P file-sharing applications, and then presented two new measurement methodologies to achieve this goal. We used our parallel crawler and our peer sampler to obtain accurate snapshots of available files among peers in the Gnutella network. Using these snapshots, we conducted analysis to provide a better understanding of the distributions and correlations of available files throughout the system.

We plan to continue this work in the following directions: We continue to collect many more snapshots to characterize properties of Gnutella files over longer timescales. Furthermore, we plan to further examine the models that properly capture each characteristics of available files in Gnutella. These models can be used in simulation based evaluations of file sharing applications.

CHAPTER 7

Dynamic Peer Properties

One of the most important, and least understood, characteristics of peer-to-peer systems is their dynamic nature. A peer joins the system when a user starts the application, contributes some resources while making use of the resources provided by others, and leaves the system when the user exits the application. We define one such join-participate-leave cycle as a *session*. The independent arrival and departure by thousands—or millions—of peers creates the collective effect we call *churn*. The user-driven dynamics of peer participation must be taken into account in both the design and evaluation of any P2P application. For example, the distribution of session length can affect the overlay structure [11], the resiliency of the overlay [70], and the selection of key design parameters [29]. Moreover, every simulation or analysis study of a peer-to-peer system relies on some model of churn. Towards this end, researchers and developers require an accurate model of churn in order to draw accurate conclusions about peer-to-peer systems.

However, accurately characterizing churn requires fine-grained and unbiased information about the arrival and departure of peers, which is challenging to acquire in practice, primarily due to the large size and highly dynamic nature of these systems. Therefore, the characteristics of churn in large-scale P2P systems are not currently well understood. Several measurement studies [45, 48, 106] have presented a high level view (*e.g.*, the CDF of session lengths [60] or median session length [45]) of churn as part of broader characterizations of P2P systems. While these studies have revealed that peer participation is highly dynamic, their findings are dramatically different. For example, the reported median session lengths varies from one minute to one hour [5]. In the absence of a reliable model for churn, researchers must make assumptions about the distribution of arrival times and session lengths that may be incorrect.

This chapter takes a major step towards increasing our understanding of churn by conducting deeper analysis and relying on more accurate measurements. One of our contributions is to identify a number of key challenges in characterizing churn that arise from factors such as measurement limitations, network conditions, and peer dynamics. We then develop techniques to address these difficulties or at least bound the resulting error. As a result, our measurements are significantly more accurate and representative.

Our second contribution is an examination of churn at two levels: *(i) Group-level characteristics* that capture the behavior of all participating peers collectively, and *(ii) Peer-level characteristics* that capture the behavior of specific peers across multiple appearances in the system over time. Furthermore, we examine some of the underlying causes and implications of our findings on the design of P2P systems. To ensure the broad applicability of our results, we study churn in three types of widely-deployed P2P systems: Gnutella, an unstructured file-sharing system; Kad, a Distributed Hash Table (DHT); and BitTorrent, a content-distribution system. Examining multiple systems allows us to explore the similarities and differences in churn behavior between different types of P2P systems. Our main results can be summarized as follows:

- Group-level properties of churn exhibit similar behavior across all three applications, but per-peer properties in BitTorrent are significantly different.
- Session lengths are fit by Weibull or log-normal distributions, but not by the exponential or Pareto.
- Past session length is a good predictor of the next session length in Gnutella and Kad, but not BitTorrent.
- The availability of individual peers exhibits a strong correlation across consecutive days.

- In BitTorrent, peers frequently remain in the system long after their downloads complete.

The remainder of the chapter is laid out as follows. Section 7.1 presents the related work on characterizing churn. Section 7.2 describes the three systems we examine in more detail. Section 7.3 identifies common pitfalls in studying churn and how we address them in our study. Sections 7.4 and 7.5 present our results, while Section 7.6 describes some key implications of our results on the design of P2P applications. Finally, Section 7.7 concludes the chapter.

Material in this chapter was adapted from material previously published in conference proceedings [19]. The material was co-authored with Prof. Reza Rejaie. The experimental work and writing are entirely mine. Prof. Reza Rejaie provided technical guidance and editorial assistance.

7.1 Related Work

We are not aware of any prior study that focuses primarily on churn in P2P networks. However, several studies present a passing investigation of session length as part of wider characterizations of P2P applications. We divide these studies into two groups based on their measurement technique as follows:

Passive Monitoring: As part of a study on P2P flows in a large ISP network, Sen *et al.* [48] use passive measurement at several routers to monitor flows in FastTrack, Gnutella, and Direct-Connect. They present a CDF of the duration an IP address is active (the *ontime*), based on a threshold, $\delta = 30$ minutes, of inactivity. They show that the ontime is heavy-tailed, but does not follow a Zipf distribution when ranked. As part of a study on workload characterization in Kazaa, Gummadi *et al.* [45] present session lengths based on passive monitoring of a router at the University of Washington. They found that session lengths are heavy-tailed, with a median session length of 2.4 minutes while the 90th percentile is 28.25 minutes. In general, passive monitoring techniques to characterize churn are likely to underestimate session lengths because some peers may not be continuously generating traffic through the observation point. Difficulties in correctly identifying peer-to-peer flows [3] can also limit measurement accuracy. Furthermore, it is difficult to determine whether the subset of captured users is representative of the entire P2P user population, especially if data is collected at a small number of measurement points.

Active Probing: Several studies use active probing or crawling to characterize P2P networks and present the behavior of session length across peers. Chu *et al.* [59] present the session length distribution in Napster and Gnutella and fit it to a log-quadratic distribution. In their insightful characterization of peers in Napster and Gnutella, Saroiu *et al.* [60] present a CDF of session durations, showing session lengths are heavily skewed. They also present CDFs of peer availability in these systems. Bustamante and Qiao [68] monitored peers in Gnutella to motivate preferential neighbor selection based on uptime. Their tool measures the length of sessions for peers which return to the network during their measurement period, and they fit peer session lengths to the Pareto distribution. More recently, Liang *et al.* [56] similarly provide a CDF of session lengths for super-nodes in the Kazaa network, based on active probing. Finally, Bhagwan *et al.* [61] examine availability in the Overnet DHT using probing. They show there is little correlation between the availability of different peers. Prior studies of BitTorrent [64, 65] have used tracker logs to show that session lengths are heavily skewed. One study of BitTorrent [66] examines the lingering and downtime distributions and conclude they are both exponentially distributed.

Each of these studies show that session-lengths are not Poisson, and some of the studies further conclude that session lengths are heavy-tailed (or Pareto). Reports of the median session length vary dramatically, from one minute to one hour (as summarized in [5]). In this study, we identify common pitfalls in measuring churn, such as biased peer selection and false negatives, which we believe are the main contributing factors for the conflicting results reported by previous studies. Leveraging our more accurate measurements, we find that session-lengths are not heavy-tailed or Pareto, but are more accurately modeled by a Weibull distribution. Additionally, we conduct a more detailed study of churn, exploring aspects such as the inter-arrival distribution and correlations across sessions. Finally, our work compares churn characteristics across different P2P systems.

7.2 Background

Before we discuss our methodology, we need to introduce the systems we examine. The Gnutella and Kad overlays provide a keyword-search function that allows users to locate files that have the keyword in the filename. Gnutella performs these lookups using an unstructured topology, while Kad uses a DHT. BitTorrent, on the other hand, forms an overlay to facilitate the rapid transfer of very large files (100+ MB) to a large number of peers. In the following subsections, we briefly describe the relevant aspects of these systems and introduce our datasets.

Overall, our goal is to measure the arrival and departure time of peers so that we can compute characteristics such as session lengths and inter-arrival intervals. Each system provides slightly different hooks for measurement, each with advantages and disadvantages. The two most important properties are the *precision* in measuring arrival and departure times and the ability to capture a *representative* set of sessions. We address the issues of possible inaccuracies or bias in our data in Section 7.3.

Our datasets consist of two types: *(i)* centralized logs that capture the arrival and departure time of each peer, and *(ii)* sequences of snapshots that record the peers present at a particular time. By comparing snapshots, we can deduce when a peer arrived and departed. The length of time required to capture a snapshot, Δ, determines the precision in determining the arrival or departure time. All of the snapshots were captured with Cruiser, the fast, distributed P2P crawler developed in Chapter 4. Cruiser operates by progressively exploring an overlay topology, querying peers for a list of their neighbors, and adding newly discovered peers to its queue until the queue is exhausted. The start time and length of our datasets are summarized in Table 7.1.

Gnutella: Gnutella is currently one of the most popular P2P systems, with more than 1 million simultaneous users [107]. It uses a two-tier overlay structure, similar to FastTrack and eDonkey. Most peers are *leaf peers*, while a small fraction of peers act as *ultrapeers*. The leaf peers connect to a handful of ultrapeers, which index the content of the leaves. Searches spread out from ultrapeers using a modified expanding-ring search.

Our Gnutella data consists of snapshots of the entire Gnutella network taken with Cruiser. Using a special crawler hook provided by modern Gnutella [108], Cruiser can capture 1.3 million peers within 7 minutes. Since the snapshots capture all peers in the system, they are necessarily representative. The datasets consist of five sets of 48-hour periods of back-to-back snapshots, with crawl durations ranging from 4 to 10 minutes.

Kad: Kad is a Kademlia-based [24] P2P search network used by the eMule P2P file-sharing software [109]. To our knowledge, Kad is the largest deployed DHT, with more than 1 million simultaneous users. Similar to other DHTs, each peer has a persistent, globally-unique identifier of length b bits (in Kad's case $b = 128$ bits). Keywords are hashed to b bits and stored on the peer with the closest matching identifier. Each peer stores a structured routing table pointing to other nodes in the network such that the expected number of overlay hops to perform any lookup is $O(\log n)$, where n is the population size.

We used Cruiser to capture a subset of the DHT identifier space, called a *zone*. As an input parameter, Cruiser accepts a zone, specified as a Kad ID address and mask, analogous to an IP subnet address. We use the "slash notation" to specify Kad ID zones. For example "0x594/10" specifies all Kad identifiers where the high-order 10 bits match the high-order 10 bits in the hexadecimal number 0x594. Monitoring a larger zone includes more sessions, but requires more time to crawl, and thus decreases the precision.

Since each peer selects its ID uniformly at random[1], each zone is a uniformly random sample of the Kad network as a whole and therefore representative. We used

---

[1] Technically, the identifiers are selected using a pseudo-random number generator seeded with entropy collected by the operating system. Because the identifiers are 128 bits, the probability of collisions occurring is extremely small.

the Kad version of Cruiser to collect 48-hour slices of back-to-back snapshots of 4 different zones within the Kad overlay. The zone addresses and crawl durations (Δ) are given in Table 7.1 and Figure 7.1b.

BitTorrent: BitTorrent is a popular P2P application for distributing very large files (100+ MB) to a large group of users. Unlike most P2P systems, which form one large overlay, BitTorrent has a distinct overlay for each file. To download a file, peers exchange different blocks of the content until each peer has the entire file. The peers locate one another using a rendezvous point, called a *tracker*, whose address is provided to new members out of band. Each new peer contacts the tracker via HTTP, periodically sends an update of its progress to the tracker, and informs the tracker when it departs. Each peer may receive the entire file across multiple sessions, [B*i.e.*, it may obtain only a subset of blocks in one session and resume the download later. Many peers may give up without downloading the whole file [64]. The tracker logs its interactions with peers, providing the arrival and departure times of peers with one second resolution. While the tracker records the arrival time of all peers, the departure time is only captured for peers which depart gracefully. We have obtained

Dataset	Start Date	Duration	Kad Zone
Gnutella 1	Oct. 14, 2004	2 days	
Gnutella 2	Oct. 21, 2004	2 days	
Gnutella 3	Nov. 25, 2004	2 days	
Gnutella 4	Dec. 21, 2004	2 days	
Gnutella 5	Dec. 27, 2004	2 days	
Kad 1	Apr. 13, 2005	2 days	0xab0/10
Kad 2	Apr. 16, 2005	2 days	0x594/10
Kad 3	Apr. 18, 2005	2 days	0xe14/10
Kad 4	Apr. 21, 2005	2 days	0x734/12
BitTorrent Red Hat	Mar. 21, 2003	3 months	
BitTorrent Debian	Feb. 22, 2005	2 months	
BitTorrent FlatOut	Nov. 11, 2004	2 months	

TABLE 7.1: Measurement collections

tracker logs from three long BitTorrent networks: Debian ISO images, a Red Hat ISO image, and a demo of the game FlatOut.

7.3 Pitfalls in Characterizing Churn

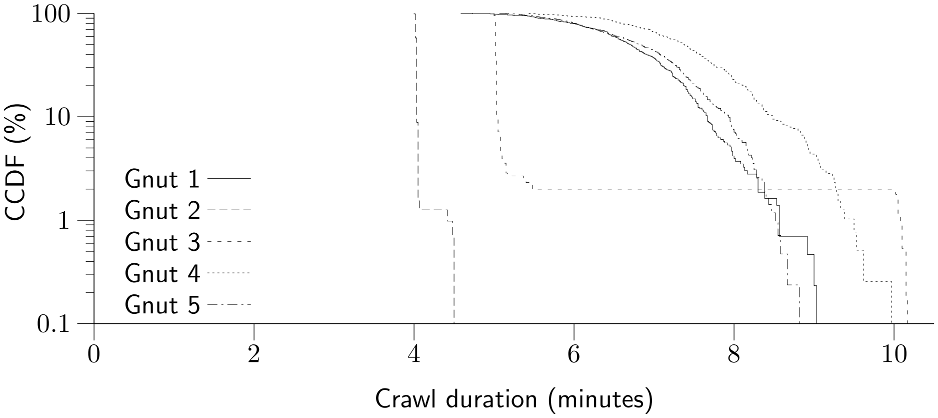
In the previous section, we described how our data was collected and the granularity of the measurements. However, we have not yet addressed how to cope with limitations of the data, such as peers missed during a crawl or sessions longer than the measurement period. While we cannot address every possible event that might introduce error into the measurement and analysis of churn, the following subsections discuss the pitfalls most likely to cause significant error. In many cases, we are able to overcome the problem entirely. In some cases, we must settle for bounding or estimating the error.

7.3.1 Missing Data

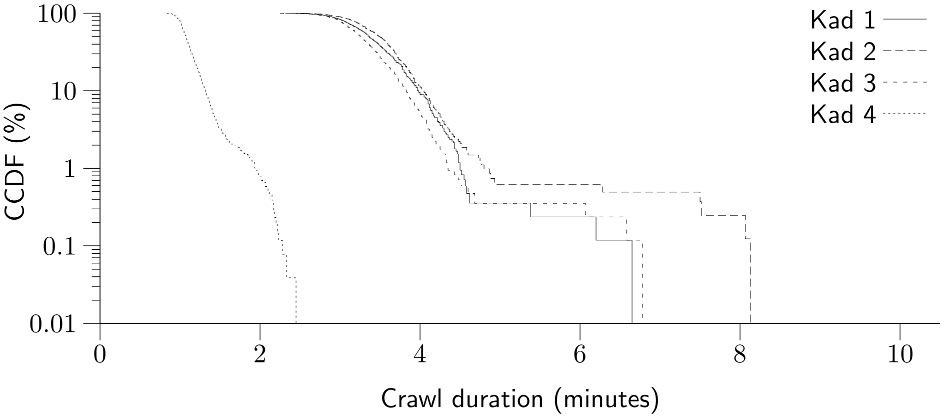
One of the challenges in collecting data for studying churn is that it requires continuous measurement. If the observation software crashes or loses network connectivity, data will be missing, essentially breaking the dataset into two pieces. If the gap is unnoticed and the data is processed as one piece, it may introduce considerable error.

In Gnutella and Kad, our data is composed of a series of snapshots captured by a crawler. Each snapshot contains the start time of the crawl. Examining the sequence of these times, we did not find any significant gaps in the data. Within each dataset, the time from the start of one crawl to the start of the next is reasonably close to the mean. The distributions are shown in Figures 7.1a and 7.1b.

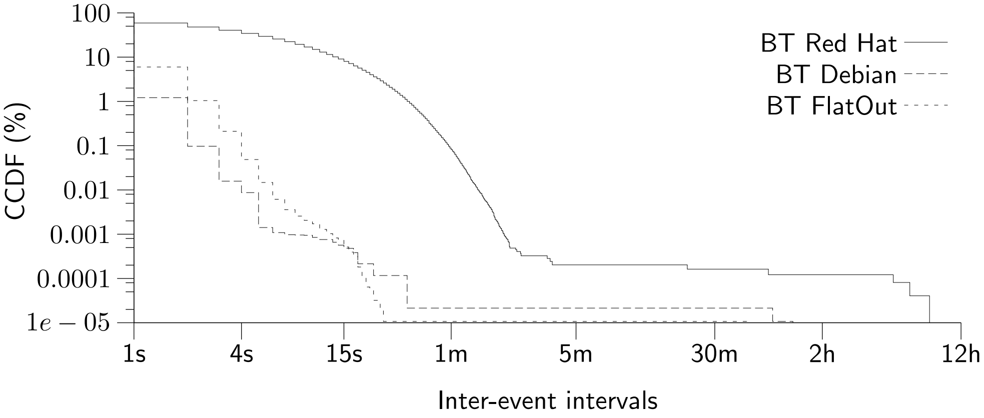
For BitTorrent, our data consists of logs from BitTorrent trackers which record each event with 1-second granularity. In our data, typically there are just a few seconds between each event. However, as shown in Figure 7.1c there are a few outlying points with unusually large gaps between one event and the next. In total, we found five significant gaps in the Red Hat log, two in the Debian log, and one in the FlatOut



(a) Gnutella crawl duration distributions



(b) Kad crawl durations distributions



(c) BitTorrent inter-event time distributions

FIGURE 7.1: Inter-event time distributions

log. The shortest of these gaps is twenty minutes while the longest is eight hours. All other inter-event intervals are less than 4 minutes. Although the Red Hat log has been studied by other researchers [64, 110, 111], to our knowledge these gaps have not been previously reported. We also checked if the clock ever ran backward, but it did not. It seems very unlikely that an otherwise busy tracker receiving several hits per minute would abruptly have an eight-hour period of silence. Therefore, we conclude that during these gaps the tracker was not running or was experiencing network connectivity problems. An alternative explanation is that the clock time on the tracker was changed. Instead of using the entire logs, we restrict ourselves to the largest contiguous portion between gaps.

7.3.2 Biased Peer Selection

The most common approach [56, 59, 60] for measuring churn has been to select a set of peers and ping them at regular intervals to determine when their sessions begin and end. However, polling a closed population of peers repeatedly captures sessions from a small fraction of all peers (those who return regularly). This introduces bias in two ways: *(i)* the potential for bias in selecting the peers and *(ii)* consecutive sessions of the same peer are correlated (as we will show in Section 7.5.2). Selecting peers based on criteria that are correlated with session length necessarily leads to biased results. Selecting based on query responses [59, 60] may be biased if there is a correlation between session length and number of files shared. Similarly, conducting a partial crawl of the topology [60] may be biased due to a correlation between session length and a peer's degree [11]. One way around these difficulties is to monitor the entire system, implicitly detecting new peers.

In BitTorrent, the tracker is a central monitoring point that captures the arrival time of all peers and the departure time of all peers that depart gracefully. Sessions that ended ungracefully were eliminated from our analysis (22% in Red Hat, 70% in Debian, 27% in FlatOut). While tracker logs provide a comprehensive account for a particular tracker, it is possible that different trackers see different behavior. To account for this, we present results from three different trackers, serving two different types of files: two Linux ISO images and one game demo.

For Gnutella, we use snapshots of all the peers in the system collected with Cruiser at approximately 7 minute intervals. For Kad, we use Cruiser's Kad module to capture all the peers within a particular zone of the DHT. Since each peer selects its Kad ID uniformly at random, there is no correlation between a peer's zone and its session length. Unlike sampling a set of addresses and probing them, zones allow us to monitor the arrival of new peers and therefore do not suffer the drawbacks of closed populations. For these reasons, zones are a representative way of characterizing the overall system behavior with respect to churn. In Section 7.3.4, we address the possibility that some peers may be missing from the snapshots (*i.e.*, false negatives).

7.3.3 Handling Long Sessions

Because we can only make observations for a finite length of time, we must carefully account for long sessions. Given a measurement window of length τ, we can compute the length of each session that begins and ends within the window. However, this would lead to a significant bias towards shorter sessions. For example, we could only observe a session of length τ if it started exactly at the beginning of our measurements, whereas we would have τ opportunities to observe a session of unit length. Worse still, we are not able to measure any sessions longer than τ.

We use the "create-based method" employed by Saroiu *et al.* [60] to overcome this dilemma. We divide the measurement window into two halves, and only consider sessions that begin during the first half. This provides equal opportunity to observe session lengths less than $\frac{\tau}{2}$. We cannot make unbiased measurements of the length of longer sessions using this method. However, we can observe how many sessions started with a length greater than $\frac{\tau}{2}$, allowing us to *count* sessions that continue past the end of our measurement window, even though we cannot measure them. In summary, our sample includes *all* the sessions which *begin* in the time period $[0, \frac{\tau}{2}]$, either by measuring their session length or by noting that the session length is greater than $\frac{\tau}{2}$.

Our initial measurements, as well as previous studies [59], showed fluctuations in network size correlated with the time of day. In this initial study, we are interested in studying the average behavior and reserve exploring the influence of time of day for future work. Therefore, we must capture all times of day roughly equally by either choosing $\tau = k \cdot 2$ days, where k is a positive integer, or chosing $\tau \gg 2$ days.

7.3.4 False Negatives

Unlike BitTorrent, in Gnutella and Kad there is the possibility that a peer is actually online even though it is missing from a snapshot. This could occur, for example, due to significant network congestion between the observation point and the peer. Even when the chance of false negatives is small, the impact on measurements of long sessions is significant. For example, polling every 5 minutes with a 1% chance of a false negative per peer can decrease the observed 1-day sessions by 94%. Because false negatives are inaccuracies in the measurements themselves, we cannot directly measure how often they occur. However, we can draw some inferences from the data. We consider two types of false negatives that could theoretically occur:

- **Systematic failures:** Some peers may be very prone to being missed by the crawler, perhaps due to routing instability between the observation point and the peer. If the failure is permanent and the peer is never captured, no sessions are observed and the peer is simply omitted from the data and does not significantly impact the results. If the failure is intermittent, the peer will appear to flap, frequently going up and down. In Section 7.5.3, we show that few peers exhibit flapping behavior.
- **Random failures:** Random failures are cases where the crawler randomly misses some fraction, λ, of peers during each crawl. Even if λ is quite low, the cumulative probability of a false negative grows exponentially with the session length. The observed session length distribution is therefore limited by the exponential distribution with rate parameter λ. We can leverage this fact to compute an upper-bound on λ as follows, using our Gnutella 1 dataset as an example.

We observed that 2.2% of sessions had a length of 1 day or more. Using the mean crawl duration of $\Delta = 6.7$ min, we compute the predicted decrease in these sessions according to the exponential distribution: $\exp\left(-\lambda \cdot \frac{1\ \text{day}}{\Delta}\right)$. For example, if $\lambda = 10\%$, then we would expect to observe only 1 in every 3.1 trillion sessions that are 1 day or longer. Since 2.2% is substantially more than 1 in 3.1 trillion, λ must be significantly smaller than 10%. We can compute an upper-bound on λ by solving $2.2\% = \exp\left(-\lambda \cdot \frac{1\ \text{day}}{6.7\ \text{min}}\right)$ which yields $\lambda = 1.8\%$. In fact, λ must be significantly lower, otherwise the observation of 2.2% would only be possible if nearly all sessions were at least 1 day in length.

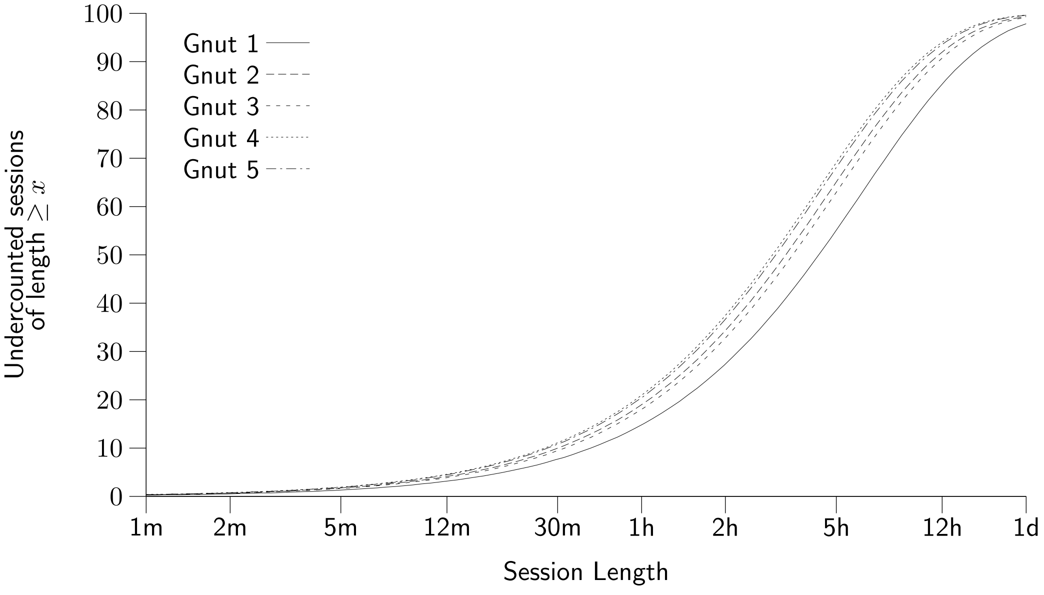
Given an upper-bound on λ, we can compute an upper-bound on the number of reduced observations as a function of session length, shown in Figure 7.2. Short and moderate length sessions (up to an hour) are not significantly affected; however, long sessions (5 hours and longer) may be dramatically undercounted. We factor this limitation into our conclusions as we analyze our results.

7.3.5 Handling Brief Events

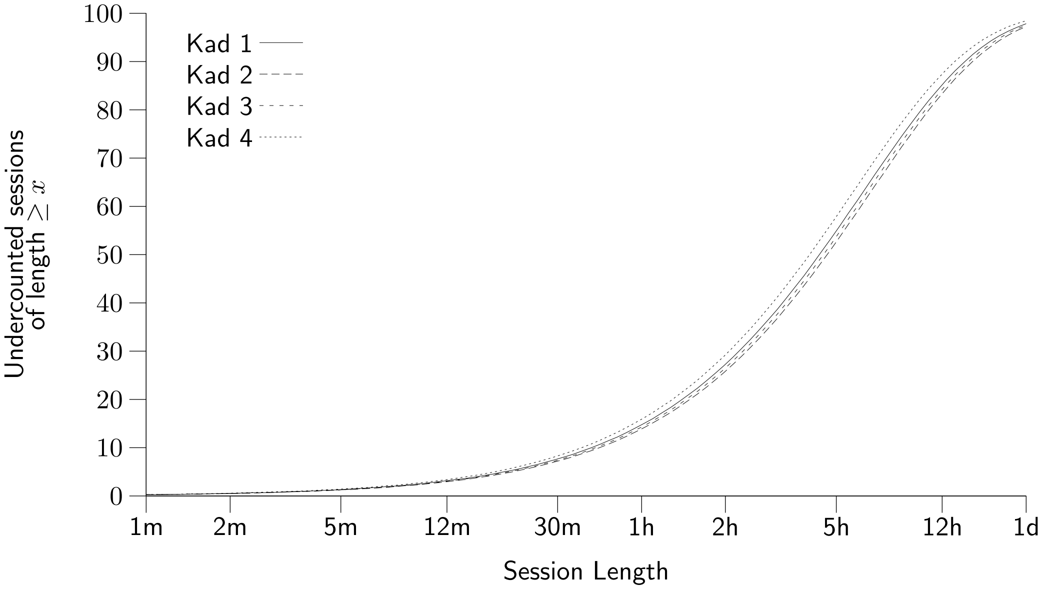
The granularity of polling can lead to measurement oversights. If a brief departure is missed, two sessions may look like one longer session. On the other hand, very short sessions may not be observed at all.

In BitTorrent, a centralized tracker records all events, so brief events are not a problem. Even if the events are shorter than the logs' granularity (1 second), they are still recorded in order.

For Gnutella and Kad, we are limited by the snapshot granularity of around 7 minutes and 4 minutes, respectively. Very short sessions will be missing from the data, but this does not adversely impact our observations of other sessions. Brief departures pose a larger problem, as they may artificially inflate the number of observed long sessions. However, in Section 7.5.3 we present evidence that few peers repeatedly come and go, suggesting that the number of errors of this type is small. Moreover, it is outweighed by the problem of false negatives, described in the previous subsection, which causes the number of long sessions to be under-counted.



(a) Gnutella



(b) Kad

FIGURE 7.2: Upper-bound on the percentage of missed observations due to false negatives

7.3.6 NAT

Network Address Translation (NAT) devices present an obstacle to observing churn in two ways. First, they can make it difficult to observe a peer at all, since it is not possible to send the peer unsolicited packets to check its status. This could prevent any NATed or firewalled peers from being included, as many as 73% of all peers [105]. Second, if there are several peers behind a single NAT device, they may all look like one peer with the NAT device's external IP address. As a result, as long as any of the actual peers is up, the single observed peer will appear to be up, creating one artificially long session. Typical home users with a NAT and only one peer behind it do not pose a problem; however, erroneous data points will be collected if a large organization has many users all connecting to the same P2P network through a NAT device.

In BitTorrent, we don't need to contact peers, since they contact the tracker, overcoming the first difficulty. To overcome the second difficulty, we use a heuristic to identify IP addresses which appear to be NAT devices with multiple peers and eliminate those IP addresses from our dataset. BitTorrent clients generate a random identifier on start-up, which is transmitted to the tracker and stored in the tracker logs. If an IP address is downloading the same file using multiple identifiers simultaneously, we classify it as a NAT device with multiple peers. Table 7.2 shows the number of IP addresses observed and eliminated from each log.

As a DHT, Kad requires all participating peers to be able to directly receive unsolicited TCP and UDP packets. Therefore, none of the Kad peers are behind NAT devices, eliminating the NAT problem. While there are NATed peers that make

Log	Total IPs	NAT (Eliminated)
BT Red Hat	115,016	9,022 (8%)
BT Debian	23,880	3,964 (17%)
BT FlatOut	1,250	64 (5%)

TABLE 7.2: Total IP addresses in BitTorrent logs and the fraction eliminated due to NAT

use of the Kad overlay, they do not participate in the overlay itself and are not part of our study.

For Gnutella, the snapshots captured by Cruiser contain all the peers directly contacted as well as all of their neighbors, including NATed peers. Therefore, contacting NATed peers is not necessary to establish their presence. However, we lack a good heuristic for Gnutella to detect when there are multiple peers behind one NAT device. This introduces a potential bias in our Gnutella data towards long sessions. Again, the typical home user with a NAT device does not pose a problem; only large organizations with multiple P2P users all behind one NAT device will cause measurement errors.

7.3.7 Dynamic Addresses

DHCP and PPP dynamically assign IP addresses to hosts. Under normal circumstances, as long as the host remains up, its IP address will not change. Therefore, dynamic address assignment is not a problem for measuring the length of sessions. However, it can become problematic for measuring the gap between sessions or correlations across consecutive sessions. When the peer is not continuously participating in the P2P network, the peer's host may have gone down and later returned with a different IP address.

Dynamic address assignment can make it difficult to make conclusions about user behavior. However, it is sometimes useful to know whether *any* peer is likely to be available at a particular IP address. For example, imagine designing the bootstrapping mechanism that helps integrate the peer into the overlay when the application starts. When the application exits, it stores to disk a cache of IP addresses of other peers so that when it starts again it can attempt to contact them. Ideally, the application wants to store IP addresses that have a high probability of being up later. It doesn't matter if the IP address represents the same user; the important part is whether there is a peer at that IP address. For these reasons, dynamic addresses are not a serious problem as long as only appropriate conclusions are drawn from the data.

Studying user behavior requires persistent unique identifiers that precisely identify particular users across sessions even when their IP address changes. Kad is the only one of the systems we examine that provide persistent unique identifiers. While BitTorrent peers also use unique identifiers, these identifiers are *not* persistent across sessions; the BitTorrent application selects a new identifier each time it starts[2]. Characterizing the behavior using persistent identifiers is useful for the design of P2P applications that store persistent state across sessions. Therefore, when studying behavior across sessions in Section 7.5, we examine the behavior of users in Kad and the behavior of IP addresses in Gnutella and BitTorrent.

7.4 Group-Level Characterization

This section explores two fundamental properties of churn which do not rely on maintaining peer identity across sessions as follows: *(i)* the inter-arrival time is the time that passes from the start of one session to the start of the next session (not necessarily by the same peer) and *(ii)* the session length is the time that passes from the start of a session until the end of that session. In other words, the inter-arrival distribution captures the pattern of when peers arrive and the session length distribution captures how long they stay in the system. Prior simulation and analysis studies have typically assumed both distributions to be exponential [5, 29, 104], though some studies have modeled the session length distribution as Pareto [69, 70] (*i.e.*, heavy-tailed) as suggested by earlier measurement studies [45, 48, 68]. We found that neither exponential nor Pareto distributions were consistent with our session length data.

Additionally, to provide greater insight into the implications of the session length distribution, Section 7.4.4 empirically examines how long peers in the system have been up (the *uptime*) and Section 7.4.5 examines the power of uptime as a predictor for how much longer peers will remain in the system (the *remaining uptime).*

Throughout Sections 7.4 and 7.5, we present many complementary cumulative distribution functions (CCDFs). Tables 7.3 and 7.4 list the number of observations

---

[2] This is true of the official BitTorrent application at the time our traces were collected. More recent versions of BitTorrent or third-party implementations may have different behavior.

Dataset	Fig. 7.1	Fig. 7.3,7.12	Fig. 7.4,7.5a	Fig. 7.5b,7.5c	Fig. 7.6	Fig. 7.9	Fig. 7.10	Fig. 7.11
Gnutella 1	430	N/A	5,624,972	N/A	154,804,229	5,670,886	2,850,712	3,118,797
Gnutella 2	714	N/A	11,713,861	N/A	263,595,662	11,700,224	8,697,797	3,182,355
Gnutella 3	561	N/A	9,743,605	N/A	221,638,443	9,749,129	6,573,694	3,368,625
Gnutella 4	390	N/A	13,232,322	N/A	154,978,617	13,256,624	9,579,572	3,834,708
Gnutella 5	423	N/A	12,117,124	N/A	155,671,531	12,116,071	8,550,825	3,737,315
Kad 1	844	N/A	7,104	N/A	404,751	7,136	5,037	2,299
Kad 2	810	N/A	6,326	N/A	390,874	6,303	4,348	2,254
Kad 3	845	N/A	6,161	N/A	397,301	6,175	4,280	2,135
Kad 4	2,573	5,676	2,723	N/A	337,722	2,713	2,205	540
BitTorrent Red Hat	2,457,473	43,956	25,782	4,058	804,296	5,627	2,294	147,536
BitTorrent Debian	9,386,976	160,020	28,047	7,250	5,856,787	13,590	16,729	137,382
BitTorrent FlatOut	9,387,684	1,050	648	398	8,409	129	52	5,542

TABLE 7.3: Number of observations for data presented in figures

Dataset	Observations	1 Hour	2 Hours	8 Hours
Gnutella 1	5,624,972	2,307,729	1,493,194	456,830
Gnutella 2	11,713,861	3,137,153	1,691,327	412,641
Gnutella 3	9,743,605	3,307,803	1,891,606	474,570
Gnutella 4	13,232,322	4,028,592	2,046,154	322,369
Gnutella 5	12,117,124	3,758,452	1,927,332	350,549
Kad 1	7,104	2,681	1,968	810
Kad 2	6,326	2,719	1,974	805
Kad 3	6,161	2,566	1,918	814
Kad 4	2,723	791	557	205
BT Red Hat	23,266	4,526	3,727	1,853
BT Debian	24,335	7,660	5,886	2,504
BT FlatOut	600	170	67	15

TABLE 7.4: Number of observations for Fig. 7.7 and 7.8

from each dataset used in constructing these CCDFs. The values vary from one figure to another depending on what type of behavior we are observing.

7.4.1 Distribution of Inter-Arrival Time

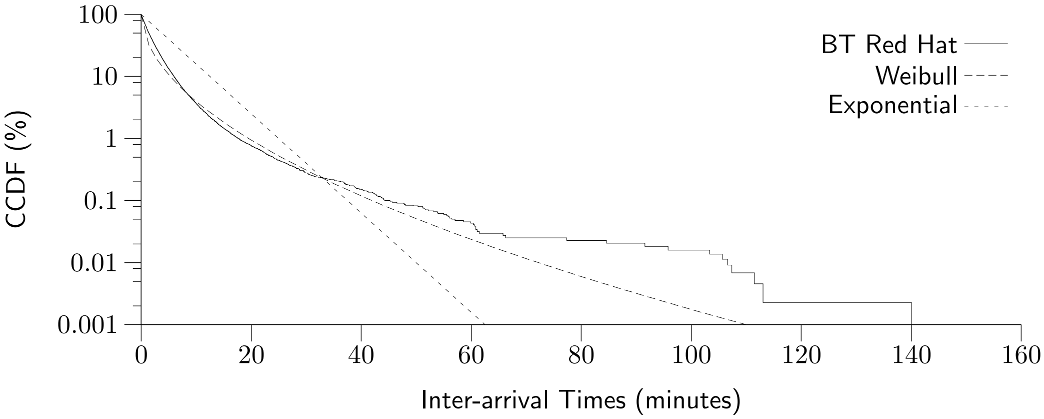
The distribution of peer inter-arrival times is an important distribution for understanding churn because it captures the pattern of how peers arrive. To measure inter-arrival times, we must be able to observe individual arrival events. In Gnutella, this is not possible since tens of thousands of new peers arrive between two consecutive snapshots. In contrast, measuring inter-arrival time in BitTorrent is straightforward since tracker logs capture arrival times with one-second granularity. To examine inter-arrival time in Kad, we use our Kad 4 trace which monitors a small fraction of the network with 1 minute snapshots. However, even at 1 minute, the granularity is poor. Therefore, we rely primarily on our BitTorrent data.

Figure 7.3 presents log-linear plots of the distribution of peer inter-arrival time for four datasets: BT Red Hat, BT Debian, BT FlatOut, and Kad 4. All four datasets appear roughly linear on the log-linear scale, which might suggest that the exponential distribution is a good fit. However, for the FlatOut and Red Hat datasets, there is slight curvature on the left side of the graphs, making a good fit impossible. For the Debian datasets, there is also some curvature, but it is much less pronounced. The Kad dataset, shown in Figure 7.3b, has poor granularity but does serve to illustrate that the inter-arrival behavior in Kad is at least roughly similar to that found in BitTorrent.

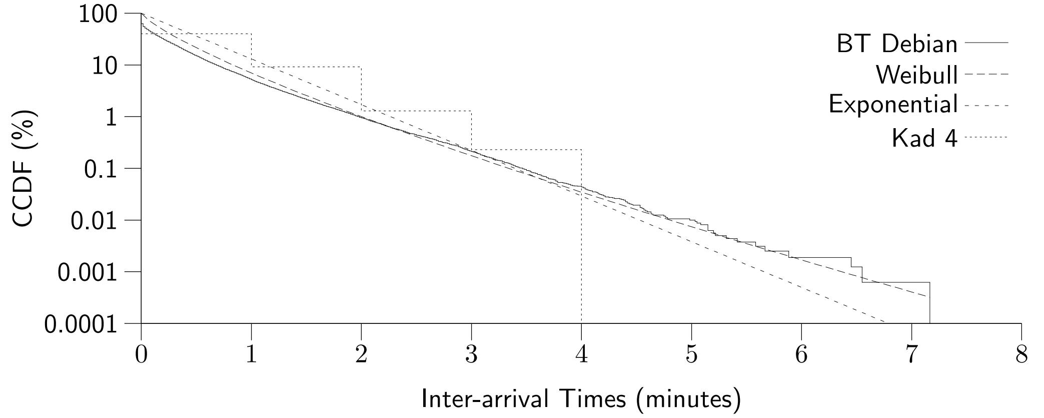
In addition to the data, Figure 7.3 shows a fit of the exponential distribution, indicating that there are more extreme values than are captured by that model. For example, in the FlatOut data, 33% of inter-arrival times are less than 10 minutes, while the exponential model predicts 9%. On the other hand, in the FlatOut data, 1.5% of inter-arrival times are longer than 10 hours, while the exponential model predicts 0.38%. Exponential distributions are typically used to model behavior resulting from a large number of independent events. However, peer arrivals are not *completely* independent. Users are less likely to be active during certain times of day (or during certain days of the week), and a surge of arrivals may occur when a link to a file appears on a popular website. Deviations from the exponential model are caused by these correlations. Despite these deviations, some may still find it acceptable due to its simplicity. However, a more accurate model is desirable.

Weibull distributions are a more flexible alternative to exponential distributions. In fact, exponential distributions are a special case of the Weibull distributions where the shape parameter is $k = 1$. For inter-arrival times, a Weibull distribution provides a much better fit, most noticeably shown in Figures 7.3a and 7.3c, with scale parameter $k = 0.79$ for Debian, $k = 0.53$ for Red Hat, and $k = 0.62$ for FlatOut. We found the parameters of the distributions using the non-linear least-squares method on the log-linear transform of the CCDF.

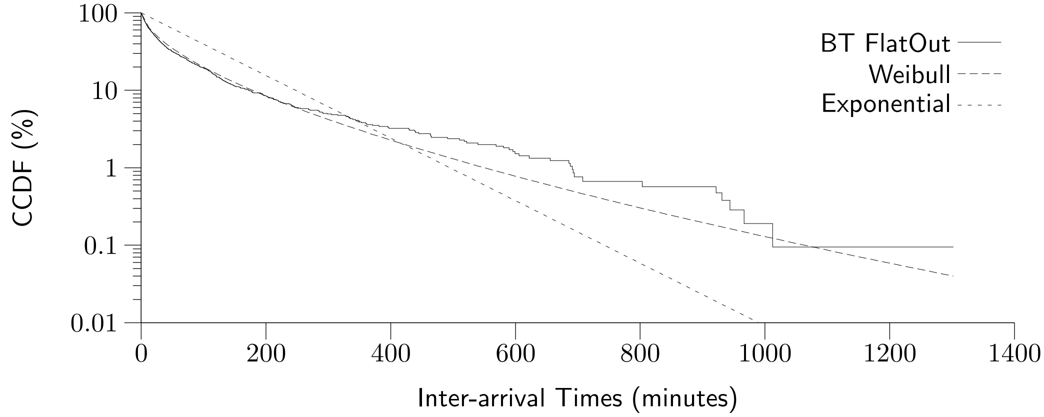
An alternative hypothesis is that over short time scales, inter-arrivals can be described by an exponential distribution, but the parameter of the distribution varies with the time of day. To explore this hypothesis, we divided the Debian dataset into 1 hour segments, containing a few hundred events each on average. For each segment, we computed the best fit via Maximum Likelihood Estimators for several types of distributions and computed the Anderson–Darling goodness-of-fit test statistic. At the $p = 5\%$ level, neither exponential, Weibull, nor log-normal distributions regularly provided good fits. We repeated the experiments on the Red Hat dataset which contains only a few dozen events per hour on average. Exponential and Weibull distributions were each able to fit the data in more than 93% of all cases. At the



(a) BitTorrent Red Hat



(b) BitTorrent Debian and Kad



(c) BitTorrent FlatOut

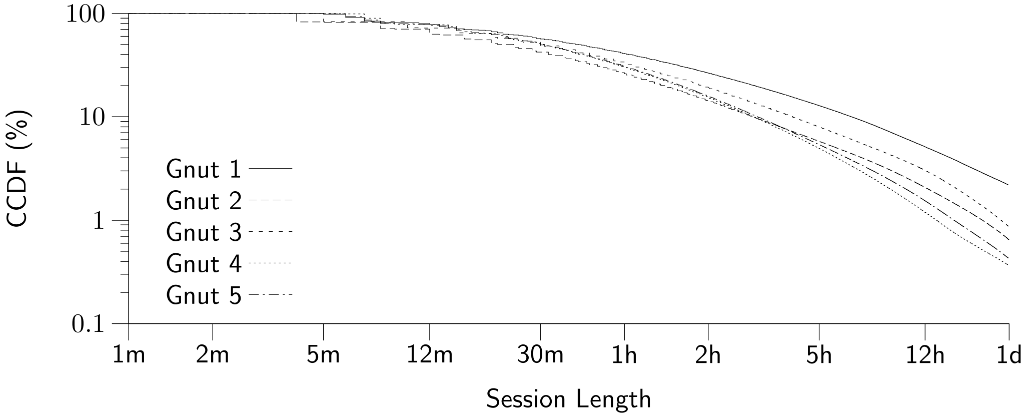
FIGURE 7.3: Inter-arrival time CCDFs

$p = 5\%$ level, we would expect at least 5% of all cases to fail the test, for a good candidate distribution, so 93% is quite good. We did not examine the FlatOut data in this way because it is too sparse to meaningfully divide into 1 hour segments.

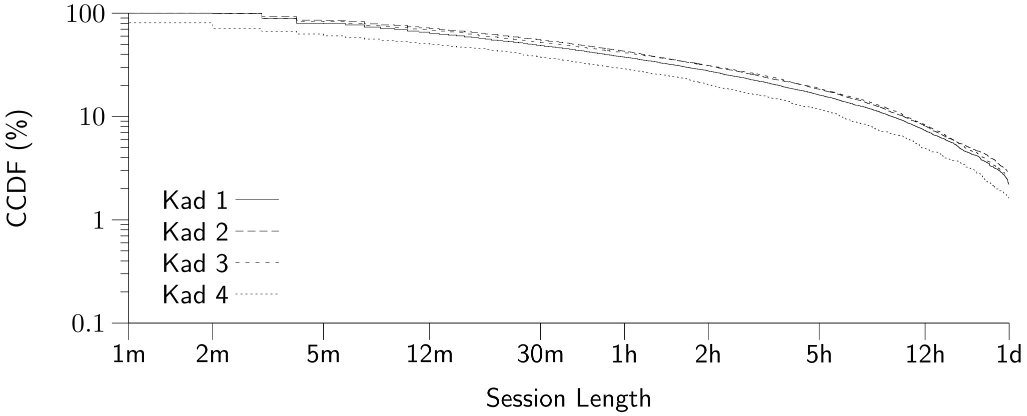
An interesting question is: Why does the exponential distribution appear to be a good candidate with the Red Hat data but not the Debian data? The Debian data provides a larger number of events per 1 hour segment, which increases the power of the Anderson–Darling goodness-of-fit test. Consequently, if exponential (and Weibull) distributions are close, but imperfect, models of user behavior, they might be accepted with a small number of data points but rejected when we supply a large number of data points (*i.e.*, enough data points for the test to detect the difference). On the other hand, the difference could be to a fundamental difference in user behavior between the two datasets. To explore this issue, we further divided the Debian events into 6 minute segments so that each segment has a few dozen on average. Only 28% of segments could be fit with an exponential distribution and 38% with a Weibull distribution, suggesting that there are fundamental differences in inter-arrival patterns from one torrent to another.

7.4.2 Distribution of Session Length

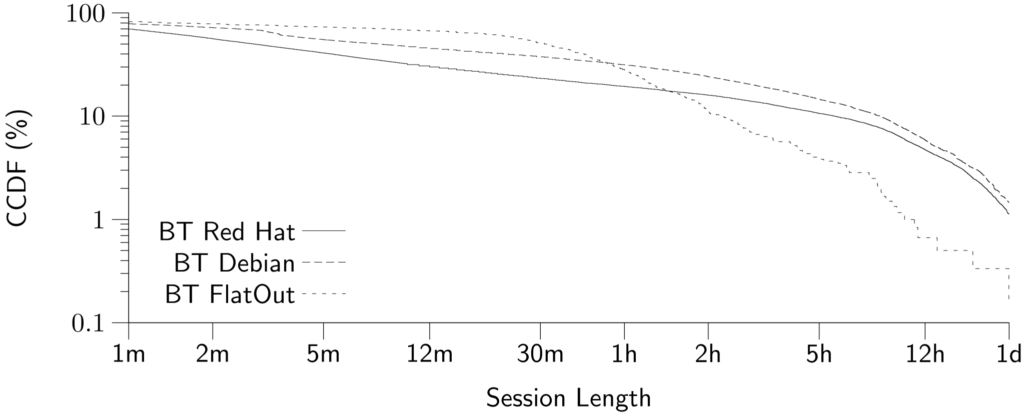
One of the most basic properties of churn is the session length distribution, which captures how long peers remain in the system each time they appear. Figure 7.4 presents the distribution of session length across different datasets for each system. Figure 7.5a presents a different angle on the session lengths in BitTorrent using a wider x-axis scale. These results demonstrate the following points: First, within a single system, the distribution of session length from different datasets are very similar. This implies that the distribution of session length does not significantly change over time. Second, and more importantly, the distributions in different systems are similar, suggesting that user behavior, which determines the session length distribution, is consistent across different P2P systems. Third, session lengths clearly do not follow the commonly-used exponential distribution, which features a sharp "knee" when plotted on log-log scale due to the relative absence of large values.



(a) Gnutella



(b) Kad



(c) BitTorrent

FIGURE 7.4: Session length CCDFs

Prior studies [45, 48, 68, 69] have reported that peer sessions lengths are heavy-tailed. A heavy-tailed distribution is one with the following property [36]:

$$P[X > x] \propto x^{-\alpha}, \text{ as } x \to \infty,\ 0 < \alpha < 2$$

The parameter α is called the *tail index* and is equal to the "slope" of the tail on a log-log plot. As a result, if a distribution is heavy-tailed, on a log-log plot of the CCDF the tail will appear linear with a "slope" between 0 and 2. Examining the tails in our BitTorrent data, we find α does not fall in this range. By fitting a line to the log-log transform of the tail of the data, we found $\alpha = 2.5$ for Red Hat, $\alpha = 2.7$ for Debian, and $\alpha = 2.1$ for FlatOut. Therefore, we must conclude that *session lengths in BitTorrent are not heavy-tailed.*

Since neither the exponential nor heavy-tailed distributions proved consistent with our observations, we investigated two other models that might provide a good fit: log-normal distributions and Weibull distributions. These two models are not heavy-tailed yet can include more extreme values than exponential distributions, making them good candidates. We found that log-normal provided a decent fit for all three BitTorrent datasets, but significantly overestimated the number of sessions longer than one day in the Red Hat and Debian datasets. The Weibull distribution was able to provide a tighter fit with shape (k) and scale (λ) parameters $k = 0.34,\ \lambda = 21.3$ for Red Hat, $k = 0.38,\ \lambda = 42.4$ for Debian, and $k = 0.59,\ \lambda = 41.9$ for FlatOut. We additionally found that the BitTorrent data could not be closely fit by a shifted Pareto distribution (which allows for slight curvature).

The Gnutella and Kad CCDFs (Figures 7.4a and 7.4b) exhibit slight downward curvature, very similar to that seen in BitTorrent over the same scale (Figure 7.4c). This suggests that they, too, are not heavy-tailed. However, due to the potential for significantly under-counting the number of long sessions in Gnutella and Kad (as described in Section 7.3.4), we cannot rule out the possibility that this curvature is a measurement artifact. All of the Gnutella and Kad session length datasets fit tightly to log-normal distributions and reasonably well to Weibull distributions.

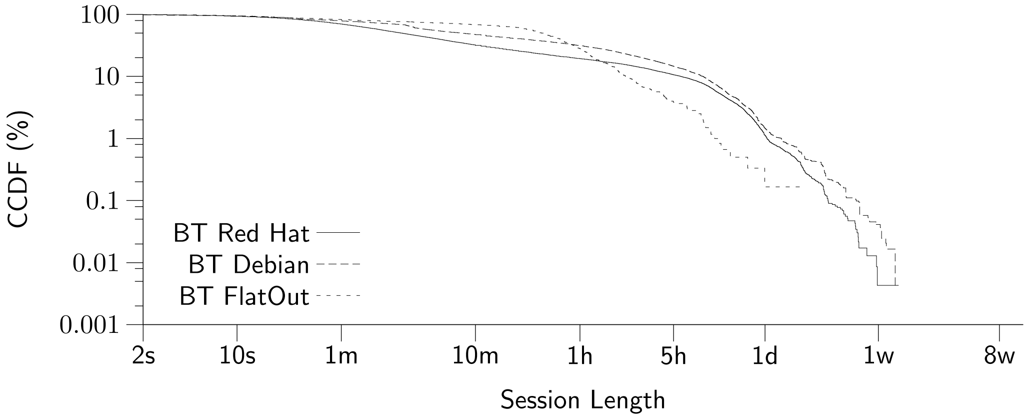
In summary, while most sessions are short (minutes), some sessions are very long (days or weeks). This differs from exponential distributions, which exhibit values *closer together in length, and heavy-tailed distributions, which have more pronounced extremes (years). The data is better described by Weibull or log-normal distributions.* We will explore the consequences of this further in the following subsections.

7.4.3 Lingering after Download Completion

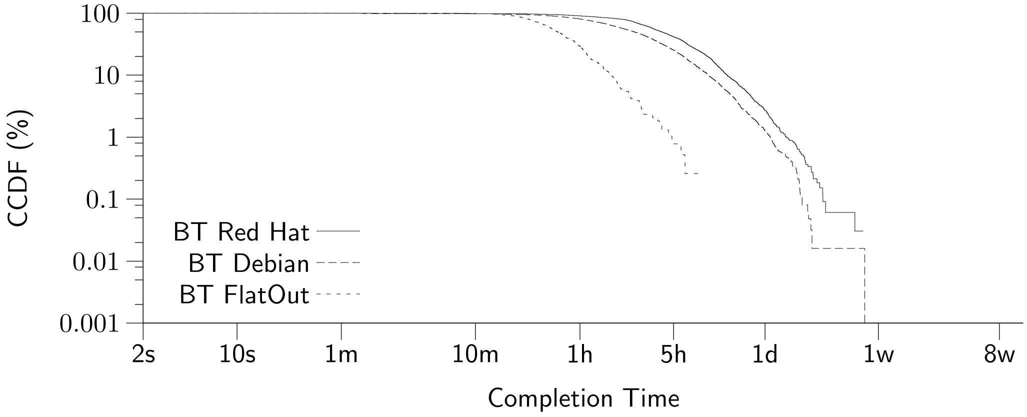
We found the similarity of session length distributions between BitTorrent and the other systems surprising. While Gnutella and Kad applications provide a keyword-search feature and manage the downloading of many files, BitTorrent is designed to download a single large file.[3] Intuitively, we would expect users to exit BitTorrent soon after the file transfer completes, which would tightly couple the session length with the transfer time, suggesting a multi-modal distribution based on common access-link bandwidths (and therefore common download durations). To investigate this issue, we divide the BitTorrent sessions into two parts: *(i)* the time to completely download the file and *(ii)* the additional time the peer lingers in the system. We call these the *completion time* and the *lingering time*, respectively. Figures 7.5b and 7.5c depicts the distribution of these two values, for sessions that download the entire file in a single session (16–61% of all sessions, see Table 7.3). Interestingly, many peers linger for a few hours after their download is complete, and a few peers linger for days or weeks, particularly in the Red Hat and Debian traces. By comparing Figures 7.5a, 7.5b, and 7.5c, we can see that the tail of the session length distribution is dominated by peers who are lingering long after their download has completed. Guo *et al.* [66, Fig. 13(a)] also examine the lingering time distribution (which they call the "seeding time") and conclude it follows an exponential distribution due to its linear shape on a log-linear plot. However, the significant curvature on the left side of their plot (similar to our Figure 7.3c) is inconsistent with an exponential distribution but could be modeled by a Weibull distribution.

---

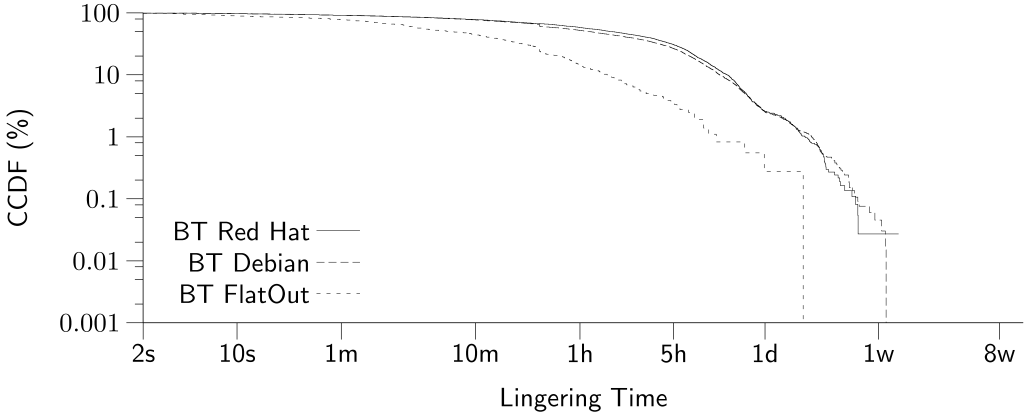
[3] Modern versions of BitTorrent can manage several file downloads within one instance of the application, but at the time of our data each file required a separate instance.



(a) BitTorrent session time distributions



(b) Completion time distributions



(c) Lingering after download completion distributionsd

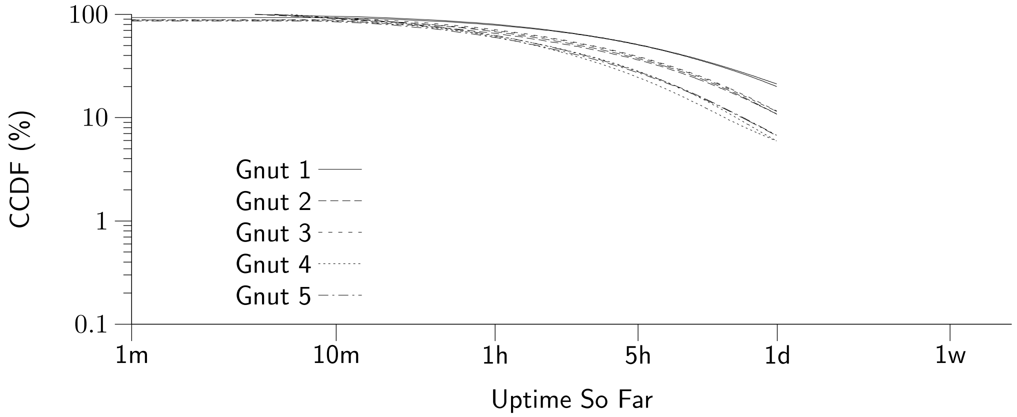
FIGURE 7.5: Interactions with transfer time

7.4.4 Distribution of Peer Uptime

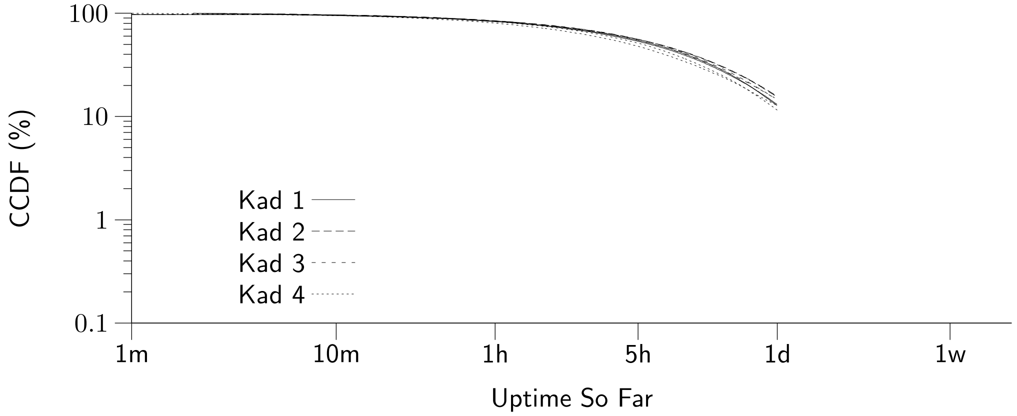
The previous subsections presented the distribution of session length across all sessions. However, they did not illustrate what combination of these peers might coexist in the system at any point of time. To address this issue, we turn our attention to how long the peers currently in the system have been present (their *uptime*).

To compute the uptime distribution, we slightly alter the methodology described in Section 7.3.3 to avoid bias towards short-lived peers as follows. Again, we divide each measurement window of length τ into two halves, A and B. For peers in B, we can either compute their uptime or we know that their uptime is at least $\frac{\tau}{2}$. For Gnutella and Kad, for each snapshot in B, we observe the uptime for each peer. For BitTorrent, where we have tracker logs instead of snapshots, we observe the uptime of each peer per once per minute during B. We then examine the distributions of all observed uptimes.

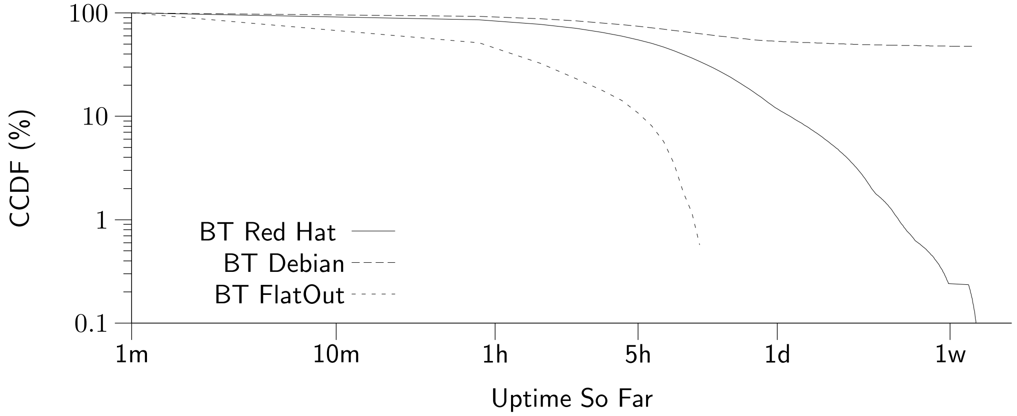
Figure 7.6 shows the CCDF of the uptime for co-existing peers within a snapshot and reveals several interesting points. First, the uptime distribution exhibits very similar behavior across different systems, except for the Debian dataset. The uptime distribution in that dataset is heavily influenced by a large number of long-lived "seed" peers run by the Debian organization. Second, a significant fraction of peers have an uptime longer than half our measurement period (shown by a gap between the rightmost data-point and the x-axis). More specifically, roughly 10%–20% of peers per snapshot in Gnutella and Kad have an uptime longer than one day, and around 1–3% of BitTorrent peers have an uptime longer than two weeks for the Red Hat and Debian traces. Third, and most importantly, these distributions are heavily weighted towards uptimes longer than a couple of hours, *i.e.*, they show that the majority of peers in each snapshot are long-lived peers. For example, if we randomly select a peer from these systems, the probability that the selected peer has been up for more than five hours is roughly 40% in Gnutella, 55% in Kad, and 60% in BitTorrent for the Linux ISO images. This effect is significantly less pronounced, but still present, in the FlatOut trace, where 15% of active peers have been up for more than five hours.



(a) Gnutella



(b) Kad



(c) BitTorrent

FIGURE 7.6: CCDF of the uptime of peers that are in the system at any given moment

The combination of the uptime distribution (Figure 7.6) and the session length distribution (Figure 7.4) present an enlightening view of churn in P2P systems as follows: *At any point of time, a majority of participating peers in the system are long-lived peers. However, the remaining small portion of short-lived peers join and leave the system at such a high rate that they constitute a relatively large portion of sessions.* Describing this from a different angle, the session length of a randomly selected session (Figure 7.4) is likely to be short whereas the uptime of a randomly selected active peer from the system (Figure 7.6) is likely to be long.

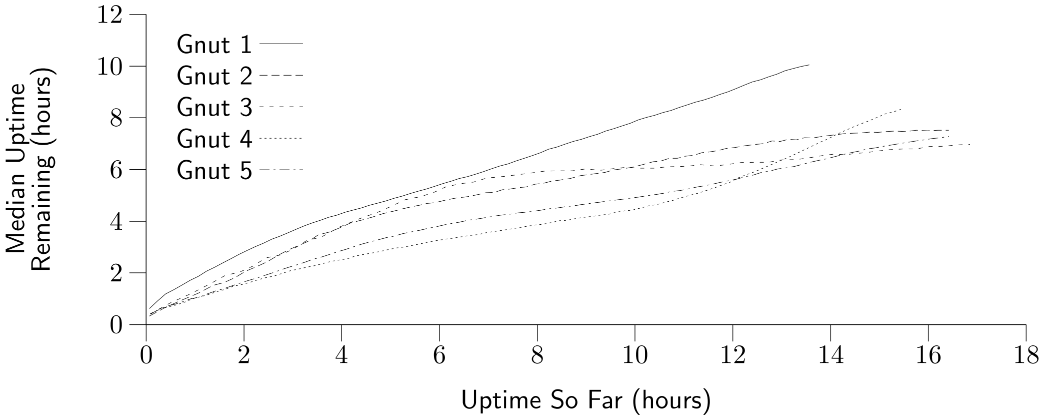
7.4.5 Uptime Predictability

One interesting question is whether a peer's uptime is a good predictor of its remaining uptime. Although Figure 7.4 suggests this is the case, to empirically explore this property we examine the correlation between uptime and remaining uptime at two levels. Figure 7.7 depicts the CCDF of the median[4] remaining uptime as a function of a peer's current uptime. It shows that while uptime is in general a good predictor of remaining uptime, its strength is different across systems and for different uptime values. More specifically, peer uptime in Gnutella is a good indicator of remaining uptime regardless of uptime value; the median peer has a remaining uptime between 50% and 100% of its uptime so far. However, the uptime of Kad peers is a stronger predictor of remaining uptime up to around 4 hours. Beyond that, the median peer's remaining uptime increases only slowly. Nevertheless, at $x = 16$ h, Kad peers have approximately the same remaining uptime as their Gnutella counterparts. In the BitTorrent datasets, we see a rapid increase in the median remaining uptime as the uptime approaches one hour, but peers up for at least two hours all have approximately the same remaining uptime. The FlatOut trace appears to wobble, which is likely noise given the relatively small number of sessions in this dataset.

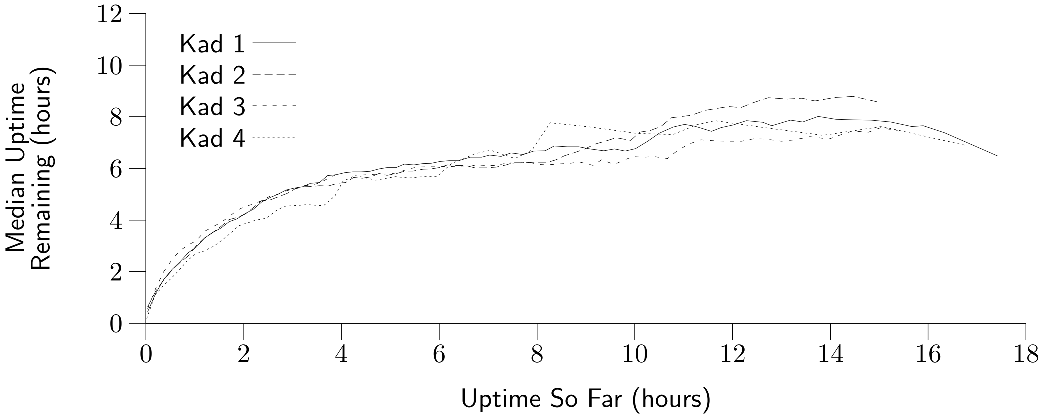
While Figure 7.7 presents the median remaining uptime, we would also like to understand the variance of this predictor. If the variance is low, that would make the

---

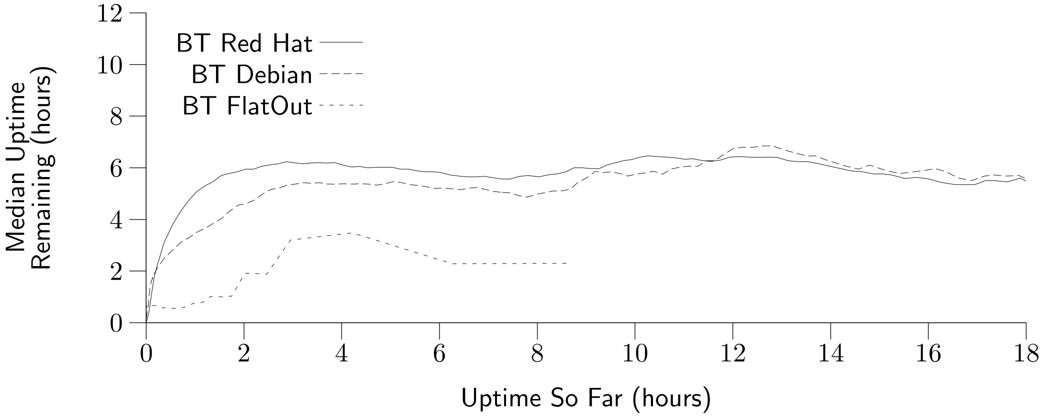
[4] Since we cannot measure the length of very long sessions as discussed in Section 7.3.3, we cannot compute the mean. However, we do know how *many* of these sessions there are and can thus find the median.



(a) Gnutella



(b) Kad



(c) BitTorrent

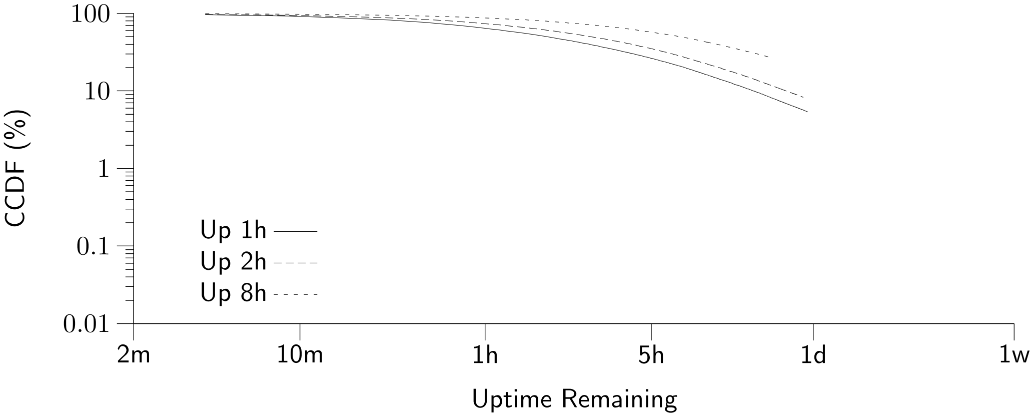
FIGURE 7.7: Median remaining uptime as a function of current uptime

predictor more useful. To explore this, Figure 7.8 shows the CCDF of the remaining uptime for peers currently up for 1 hour, 2 hours, and 8 hours, from one trace of each system. The CCDFs show that the reliability of the predictions is highly variable, covering a broad range of times. For example, in Gnutella around 50% of peers up for 8 hours will be up for at least another 8 hours. However, the bottom 20% of the peers will be up for less than 2 more hours, while the top 30% will be up for more than 16 hours! *In summary, our results show that while uptime is on average a good indicator of remaining uptime, it exhibits high variance. Therefore it should only be used when a bad prediction does not have a major cost but making better choices on average improves overall performance.*

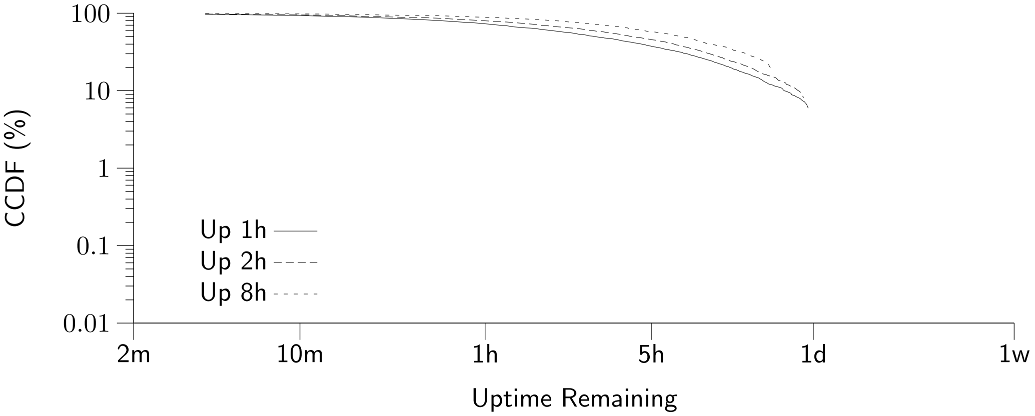
7.5 Peer-Level Characterization

In this section, we characterize the behavior of a peer across multiple appearances. These characterizations are useful for both the design and evaluation of peer-to-peer systems. For example, in the previous section we showed that the current uptime of a peer is a rough predictor for the remaining uptime. A peer that has just arrived cannot use this fact to make much of a prediction. Examining multiple appearances can reveal whether previous session lengths are good predictors of future session lengths. We examine the following characteristics: *(i)* the distribution of downtime, *(ii)* the correlation between consecutive session lengths, and *(iii)* the correlation of availability on consecutive days.

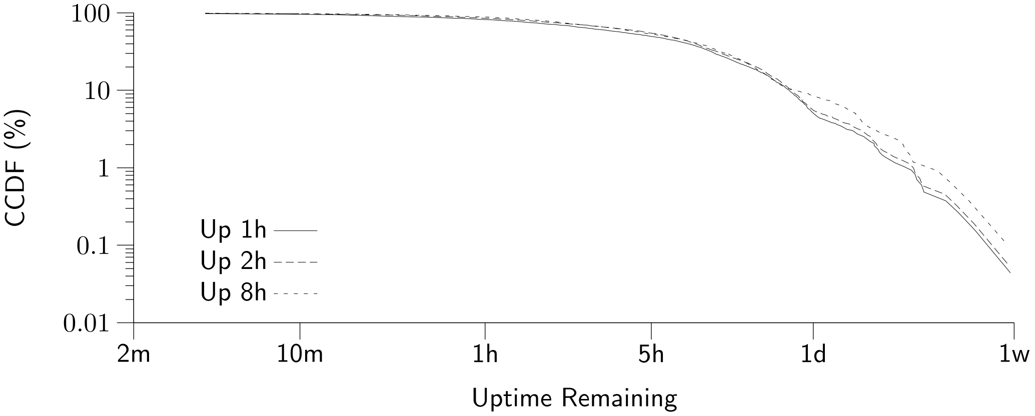
As described in Section 7.3.7, Kad is the only one of our systems in which individual peers have a *persistent* unique identifier. Thus, our characterizations of Kad reflect the behavior of individual users, while our characterizations of Gnutella and BitTorrent reflect the behavior of IP address. Both types of characterization are useful. The behavior of users is important when persistent state is stored across sessions (such as the list of filenames that a user shares), while the behavior of IP addresses is important when we want to know if *any* peer will be available at the address (such as when an incoming peer bootstraps).



(a) Gnutella 1



(b) Kad 1



(c) BitTorrent Red Hat

FIGURE 7.8: CCDF of remaining uptime for peers already up 1 hour, 2 hours, and 8 hours

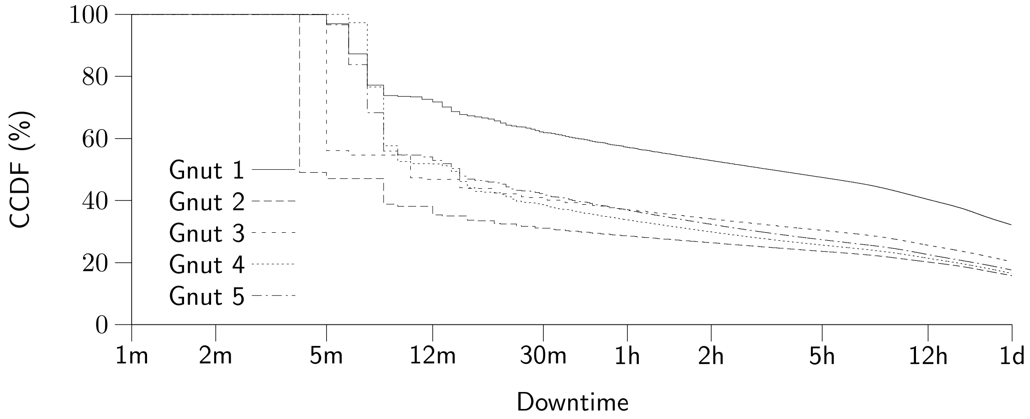
7.5.1 Distribution of Downtime

We define downtime as the interval between the moment a peer departs and its next arrival. To ensure an unbiased selection of downtime measurements, we again apply the window-halving methodology described in Section 7.3.3. The possibility in Gnutella and Kad that the crawler misses some peers, described in Section 7.3.4, has the opposite effect on downtime observations. Rather than increasing the number of long events, it may significantly increase the number of short events. Figure 7.9 presents the distribution of downtime for Kad (based on ID) as well as Gnutella and BitTorrent (based on IP address) and Table 7.3 lists the number of observations from each dataset. The gaps between the rightmost data-point and the x-axis represent the fraction of downtime events which were too long to measure without bias as well as instances where the peer never returns (*i.e.*, infinite downtime). One interesting feature of these figures is that the behavior is approximately the same in Gnutella and Kad, despite one of the measurements being based on ID while the other is based on IP.

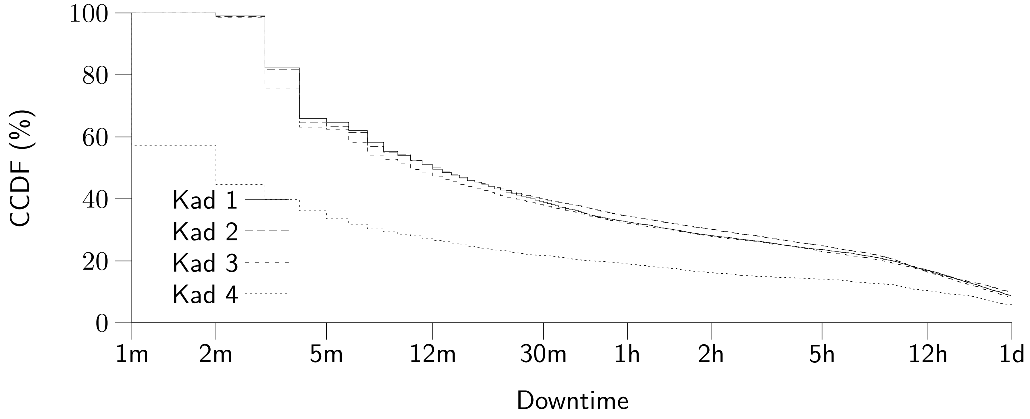
These CCDFs reveal a number of interesting qualities. First, the behavior in BitTorrent is significantly different. In Gnutella and Kad, most departing peers return within 1 day (70–85% in Gnutella and 90% in Kad), while in BitTorrent most peers do not appear to return at all. Second, in Gnutella and Kad peers that will return have a strong tendency to return sooner rather than later. For example, departed peers return within 1 hour 40–70% of the time in Gnutella and 65–80% of the time in Kad. In BitTorrent, a departed peer is somewhat more likely to reappear within a few minutes, but after that has a relatively equal probability of returning at any point between 10 minutes and 1 week, if it returns at all.[5] This difference between systems is likely due to their different modes of use. In Gnutella and Kad, users make regular use of the overlay to discover and download new files. Users join a BitTorrent overlay, on the other hand, to download one particular file. While they may depart prematurely and complete a download later, once they have downloaded the file they

---

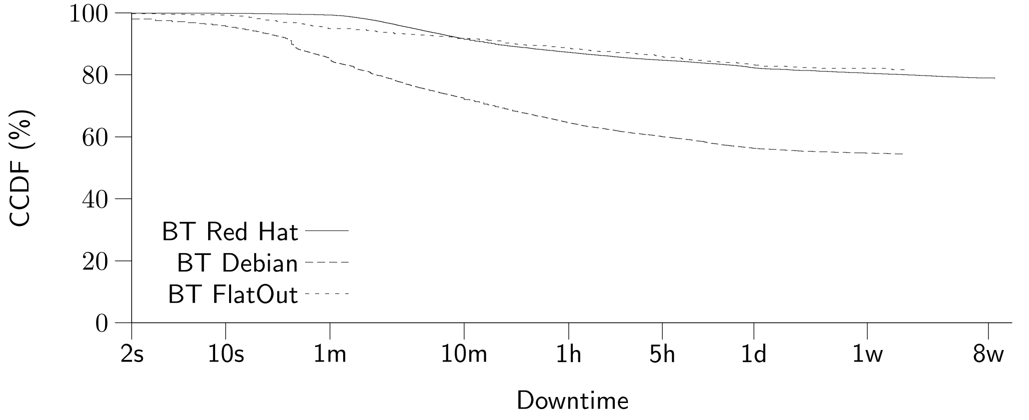
[5] Guo *et al.* [66] character the downtime of peers in BitTorrent (which they call the “sleeping time”) as exponential. However, this conclusion may be an artifact caused by incorrectly handling long-sessions as described in Section 7.3.3.



(a) Gnutella (based on IP address)



(b) Kad (based on node ID)



(c) BitTorrent (based on IP address)

FIGURE 7.9: Downtime CCDFs

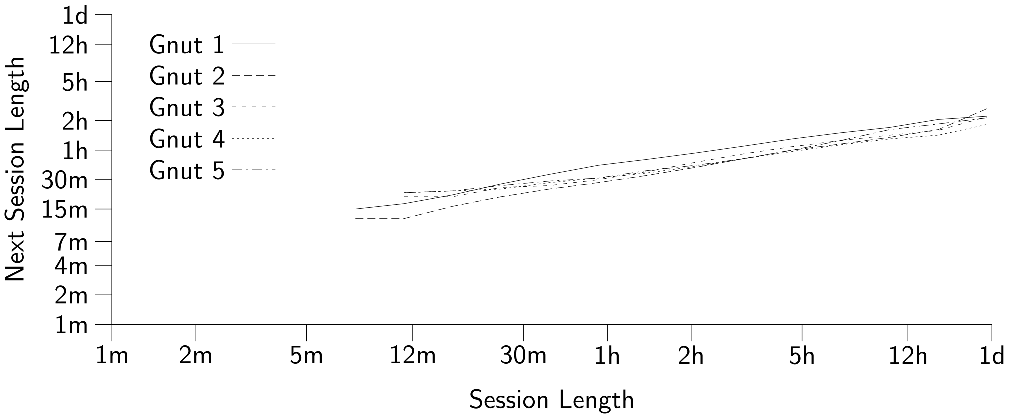
have little motivation to return. *In summary, our results show that in a content-distribution system, departed peers are unlikely to return. However, those that return are equally likely to return at any time. In contrast, most peers in file-sharing systems do return, and there is a strong tendency to return sooner rather than later.*

7.5.2 Correlation in Session Length

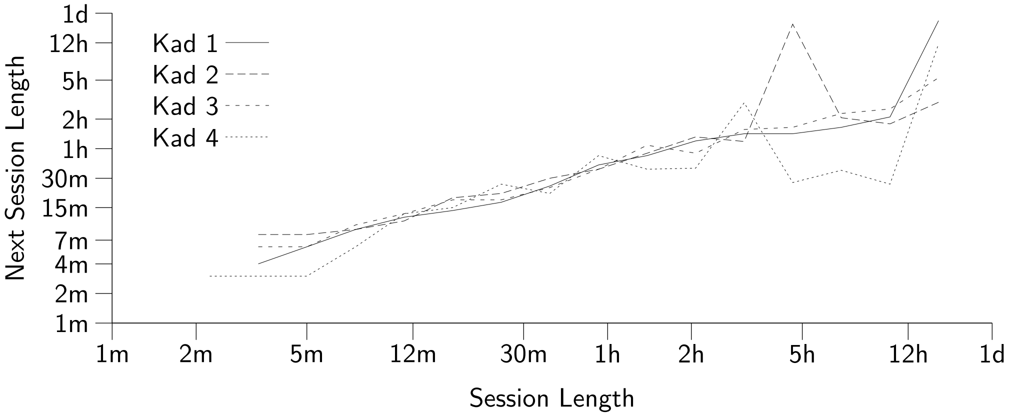
Another interesting question is "How correlated are session lengths across different appearances of a single peer?" Characterizing such an effect illustrates whether past session lengths of a peer are a good predictor of its future session lengths. For example, in Gnutella, a peer can promote itself to an Ultrapeer earlier if it can reliably estimate that its remaining uptime is likely long. We explore this correlation across *consecutive sessions* as follows. For a peer p that appears n_p times during our measurements, there are $(n_p - 1)$ pairs of consecutive sessions with session lengths $(s_{p,i}, s_{p,i+1})$ for $i \in [1, n_p - 1]$. Given the set of all pairs, across all peers, with a first session of length x, Figure 7.10 presents the median[4] second session's length. These figures show that there is a strong correlation between consecutive session lengths in Kad (based on ID) and in Gnutella (based on IP address). However, session lengths in BitTorrent do not exhibit a clear correlation. This result is not surprising because the pattern of participation in BitTorrent is likely to be different from Kad or Gnutella. In summary, *past session length of a peer is a good predictor of its next session length in both structured and unstructured file-sharing applications, but not in content-distribution systems such as BitTorrent.*

7.5.3 Correlation in Availability

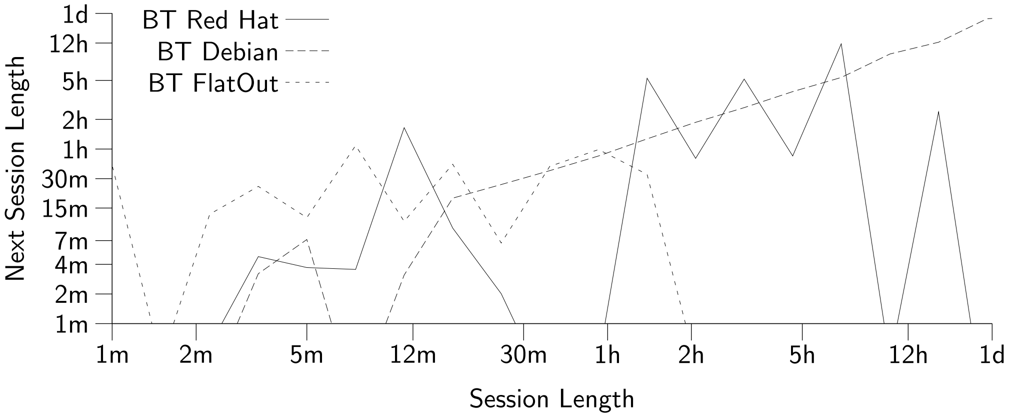
The availability of a peer is the fraction of time the peer is available in the system, which is a coarse measure of the peer's overall participation regardless of its pattern of appearance. For example, a node with 50% availability during one day, might appear just once for 12 hours, or appear 4 times and stay 3 hours during each appearance. In this section we examine correlations in the availability of peers, from one day to another.



(a) Gnutella (based on IP address)



(b) Kad (based on node ID)



(c) BitTorrent (based on IP address)

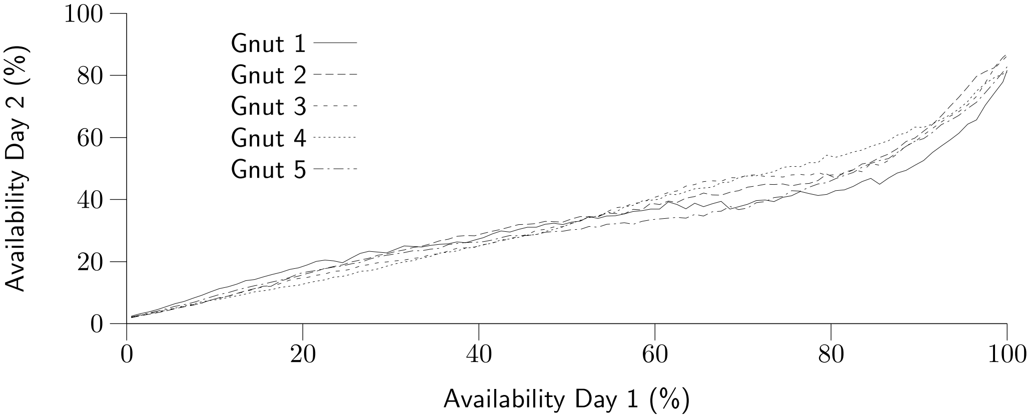
FIGURE 7.10: Correlation between consecutive session length

To study peer availability, we divide each two-day dataset into two windows of one-day length and examine the correlation between the availability of peers over these two consecutive days. For the longer BitTorrent datasets, we use each pair of consecutive days. Figure 7.11 depicts the median availability in the second day as a function of the availability on the first day. These results show that a strong correlation exists for availability in Gnutella and Kad. However, BitTorrent does not exhibit such a clear correlation, except for one case: if a peer is up for a full 24 hours, it is likely to be up for the next full 24 hours as well.

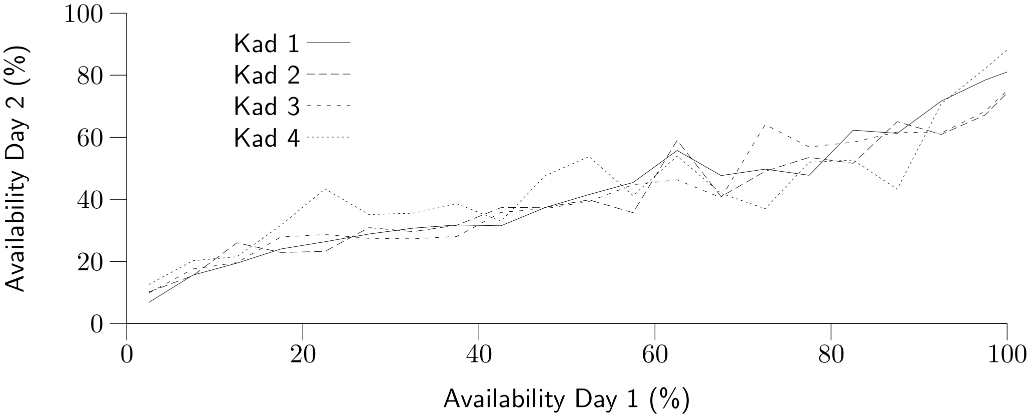
One interesting question is: "How many appearances per day does a peer make?" Figure 7.12 presents the distribution of the number of appearances per day. It shows that more than half the peers in both systems appear only once per day while a very small number of clients may return to the system very often (up to 60 times per day). Bhagwan *et al.* [61] show the availability of different peers are mostly independent. Their results are complimentary to ours, since we examine the correlation in availability of a *single peer* across multiple measurement windows while they looked for correlations across multiple peers in the same measurement window. *In summary, the availability of individual peers across two consecutive days are strongly correlated. Furthermore, most observed peers appear only once per day.*

7.6 Design Implications

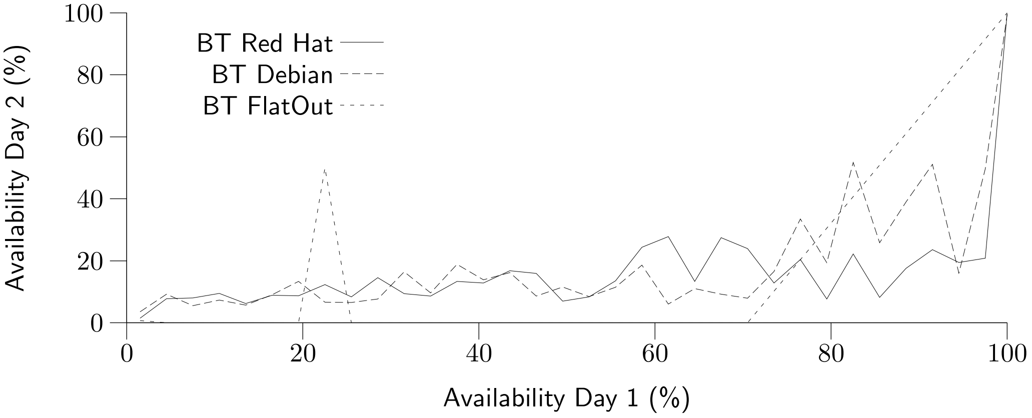
In this section, we discuss a couple of key implications of our findings on the design of P2P applications. To gracefully cope with churn, P2P systems must be able to efficiently handle the significant fraction of peers who join the system for just a few minutes. In particular, churn could significantly affect the connectivity of P2P overlays. To improve resiliency against churn, each peer should prefer to maintain routing information about other stable long-lived peers. This approach implies that long-lived peers maintain state about each other and provide a backbone of connectivity among peers. For example, peers in a DHT should select long-lived peers as neighbors to ensure better connectivity and resiliency against churn (*e.g.*, [24, 69]).



(a) Gnutella (based on IP address)

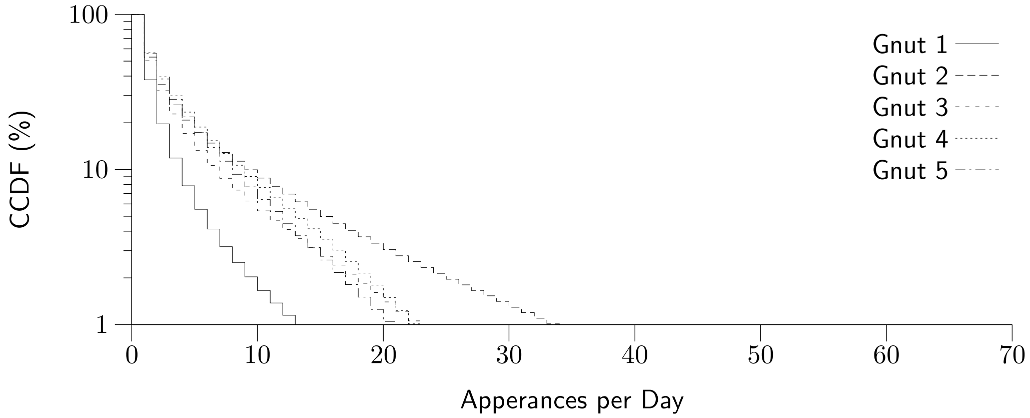


(b) Kad (based on node ID)

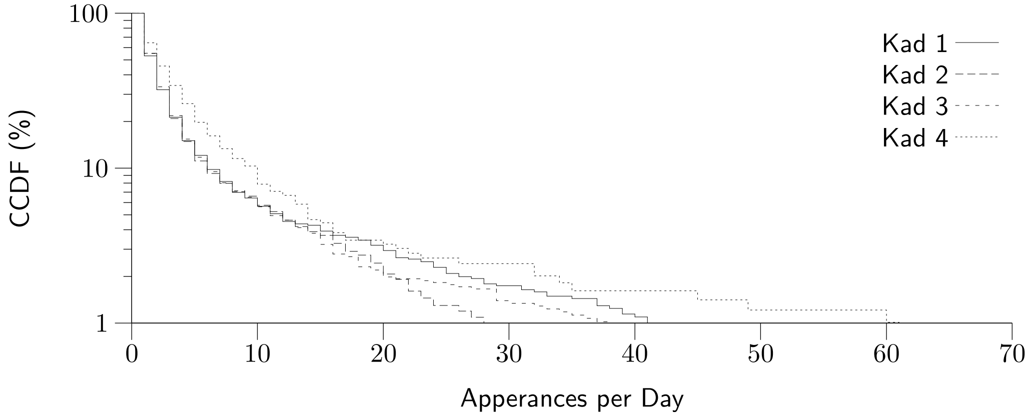


(c) BitTorrent (based on IP address)

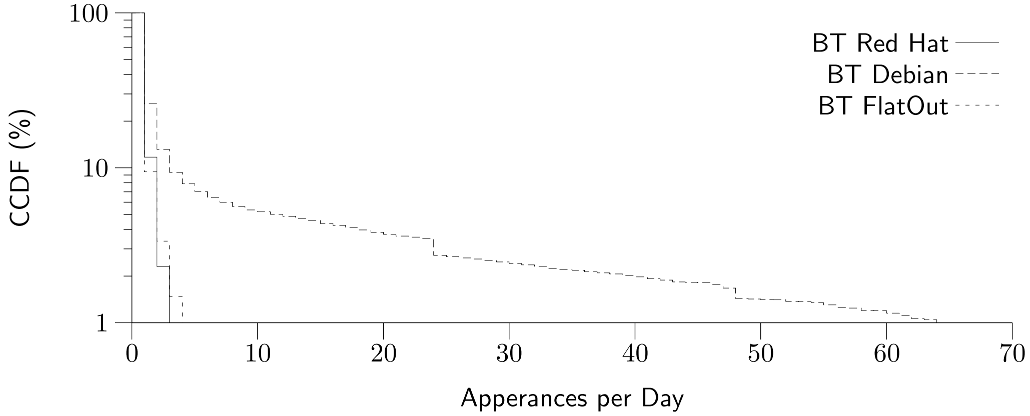
FIGURE 7.11: Correlation in availability



(a) Gnutella (based on IP address)



(b) Kad (based on node ID)



(c) BitTorrent (based on IP address)

FIGURE 7.12: Distribution of appearances per day

Our results suggest two reliable strategies for identifying long-lived peers as follows: *(i)* randomly selecting a set of peers that are all active at the same time is likely to capture long-lived peers since the distribution of uptime among peers up at any given moment (Figure 7.6) is weighted more heavily towards long-lived peers, or *(ii)* observed peers must be weighted by the number of times they are observed. The key point is that observations made *over time* become skewed towards the larger number of short-lived peers if the observations are not weighted back in favor of the long-lived peers.

Our results motivate a scalable and low cost bootstrapping mechanism that does not require a central bootstrapping node. In particular, we showed that 20%–30% of peers at any moment have an uptime longer than one day. This implies that each peer can select and cache IP addresses of several long-lived peers using one of the strategies we describe above. To connect to the system at any later time, each peer can contact cached long-lived peers to locate a participating peer in the system. Maintaining a sufficiently large cache of long-lived peers ensures that each peer can always successfully bootstrap to the system without the need to contact a well-known, centralized address. In contrast, selecting cached peers based on a first-in first-out strategy, as currently implemented by many Gnutella clients, tends to generate a list of short-lived peers that are unlikely to be available.

Finally, the strong correlation in the length of consecutive sessions and availability implies that a P2P application can roughly estimate the duration of its session length (or availability per day) based on its last observed behavior. The advantage of this approach is that it does not require the application to wait in order to predict that the user may have a long session.

7.7 Summary

This paper took a major step towards increasing our understanding of churn by characterizing different aspects of peer dynamics in three different classes of P2P systems: Gnutella, Kad and BitTorrent. We identified an array of measurement pitfalls in characterizing churn, such as biased peer selection and false negatives, and either addressed them or bound their resulting error. Our main findings, listed in introduction of this Chapter, present new insight into peer dynamics which can be used in the design and evaluation of churn-aware P2P applications. In particular, we found that the session length distribution is neither Poisson nor Pareto and is more accurately modeled by a Weibull distribution. Good models for these distributions are required for making accurate evaluations of proposed P2P protocols, via either simulation or analysis.

CHAPTER 8

Static Connectivity Properties

For structured overlays (DHTs), the graph properties are rigidly controlled by the protocol and therefore do not require an empirical study to derive useful models about their static connectivity. Unstructured overlays, on the other hand, use ad-hoc overlay construction mechanisms leading to graphs with properties that cannot be easily derived from examining the protocol description. Characterizing the graph properties of unstructured overlays is fundamental to understanding the behavior of existing systems and necessary for the evaluation of new protocols intended to run over existing unstructured overlays (*e.g.*, [112–114]).

To perform these characterizations, this chapters presents a case study of the Gnutella network, using global snapshots captured by Cruiser. The following graph properties are included: *(i)* degree distribution, *(ii)* shortest path length distribution, *(iii)* clustering coefficient, and *(iv)* resiliency (as defined in [60]).

A handful of prior studies have previously examined graph properties of the Gnutella network [6–9, 115]. One common characteristic shown in most of these studies is a power-law degree distribution. Another common observation is that Gnutella exhibits "small world" properties [40]: short path lengths and a large clustering coefficient. One early study [60] examined the resiliency of a subgraph of the Gnutella overlay. They found the overlay was resilient to random peer removals, but fragmented quickly when removing highest-degree peers first. However, these studies examined the Gnutella network in its infancy when it contained only tens of thousands of peers, at most, and did not have a two-tier architecture. Additionally, they did not explore the accuracy of their captured snapshots as we did in Chapter 4. This chapter explores each of these properties using substantially larger, more accurate snapshots.

This work focuses on developing an accurate understanding of the topological properties and dynamics of large-scale unstructured P2P networks. Such an understanding is crucial for the development of P2P networks with superior features including better search, availability, reliability and robustness capabilities. For instance, the design and simulation-based evaluation of new search and replication techniques has received much attention in recent years [99, 116–118]. These studies often make certain assumptions about topological characteristics of P2P networks (*e.g.*, a power-law degree distribution) and usually ignore the dynamic aspects of overlay topologies. However, little is known today about the topological characteristics of popular P2P file sharing applications. An important factor to note is that properties of unstructured overlay topologies cannot be easily derived from the neighbor selection mechanisms due to implementation heterogeneity and dynamic peer participation. Without a solid understanding of the topological characteristics of file-sharing applications, the actual performance of proposed search and replication techniques in practice is unknown and cannot be meaningfully simulated.

The key findings of this study of static connectivity properties include the following:

- In contrast to earlier studies [6, 7, 9], we find that node degree does not exhibit a power-law distribution. We show how power-law degree distributions can be a result of measurement artifacts.
- While the Gnutella network has dramatically grown and changed in many ways, it still exhibits the clustering and the short path lengths of a small world network.
- The overlay topology is highly resilient to random peer departure and even systematic removal of high-degree peers.

This chapter is based on material under submission to a journal, which includes material previously presented at conferences [11, 12]. The material was co-authored with Prof. Reza Rejaie and Dr. Subhabrata Sen. The experimental work is entirely mine. The writing is primarily mine, with contributions by Prof. Reza Rejaie and Dr. Subhabrata Sen, who also provided technical guidance.

8.1 Methodology

The data presented in this Chapter were collected using Cruiser, which is described in Chapter 4. We examine four snapshots for illustrative purposes to demonstrate graph properties of the overlay topology. We have examined many other snapshots and observed similar trends and behaviors. Therefore, we believe the presented results are representative. Presenting different angles of the same subset of snapshots allows us to conduct cross comparisons and also relate various findings. Table 8.1 presents the basic properties of the four snapshots we will examine in this chapter. Our follow-up study [17] examines how these properties have changed over time. In Chapter 9, we will hundreds of back-to-back snapshots to examine how the overlay topology evolves with time.

Once information is collected from all reachable peers, we perform some post-processing to remove any obvious inconsistencies that might have been introduced due to changes in the topology during the crawling period. Specifically, we include edges even if they are only reported by one peer, and treat a peer as an ultrapeer if it neighbors with another ultrapeer or has any leaves. Due to the inconsistencies, we might over-count edges by about 1% and ultrapeers by about 0.5%.

Crawl Date	Total Nodes	Leaves	Top-level	Unreachable	Top-Level Edges
09/27/04	725,120	614,912	110,208	35,796	1,212,772
10/11/04	779,535	662,568	116,967	41,192	1,244,219
10/18/04	806,948	686,719	120,229	36,035	1,331,745
02/02/05	1,031,471	873,130	158,345	39,283	1,964,121

TABLE 8.1: Sample crawl statistics

8.2 Overlay Composition

The two-tier overlay topology in modern Gnutella (as well as other unstructured P2P networks) consists of ultrapeers that form a "spaghetti-like" top-level overlay and a large group of leaf peers that are connected to the top-level through multiple ultrapeers. We treat individual snapshots of the overlay as graphs and apply different forms of graph analysis to examine their properties. We pay special attention to the top-level overlay since it is the core component of the topology. Throughout our analysis, we compare our findings with similar results reported in previous studies. However, it is important to note that we are unable to conclusively determine whether the reported differences (or similarities) are due to changes in the Gnutella network or due to inaccuracy in the captured snapshots of previous studies.

Implementation Heterogeneity: The open nature of the Gnutella protocol has led to several interoperable implementations. It is important to determine the distribution of different implementations (and configurations) among participating peers since the implementation design choices directly affect the overall properties of the overlay topology. This will help us explain some of the observed properties of the overlay. Table 8.2 presents the distribution of different implementations across discovered ultrapeers. This table shows that a clear majority of contacted ultrapeers use the LimeWire implementation. We also discovered that a majority of LimeWire ultrapeers (around 94%) use the most recent version of the software available at the time of the crawl. These results reveal that while heterogeneity exists, nearly all Gnutella users run LimeWire or BearShare.

We are particularly interested in the number of connections that are used by each implementation since this design choice directly affects the degree distribution of the overall topology. For LimeWire, this information can readily be obtained from the source code. However, not all implementations are open, and users can always change

Implementation:	LimeWire	BearShare	Other
Percentage:	74%–77%	19%–20%	4%–6%

TABLE 8.2: Distribution of implementations

the source code of open implementations. Thus, we must still collect this information from running ultrapeers in action.

Our measurements reveal that LimeWire's and BearShare's ultrapeer implementations prefer to serve 30 and 45 leaves, respectively, whereas both try to maintain around 30 neighbors in the top-level overlay.

8.3 Node Degree Distributions

The introduction of the two-tier architecture in the overlay topology along with the distinction between ultrapeers and leaf peers in the modern Gnutella protocol demands a close examination of the different degree distributions among different group of peers.

Node Degree in the Top-Level Overlay: Previous studies report that the distribution of node degree in the Gnutella network exhibited a power-law distribution [6–8] and later changed to a two-segment power-law distribution [6, 9]. To verify this property for the modern Gnutella network, Figure 8.1 depicts the distribution of node degree in log-log scale among all peers in the top-level overlay for the 10/18/04 snap-

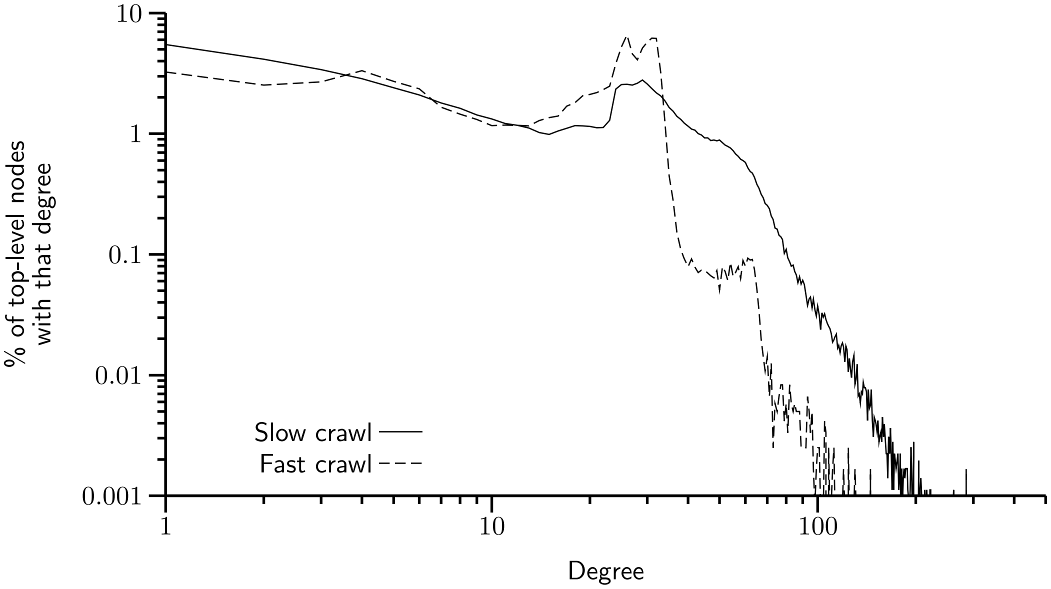


FIGURE 8.1: Observed top-level degree distributions of a slow and a fast crawl

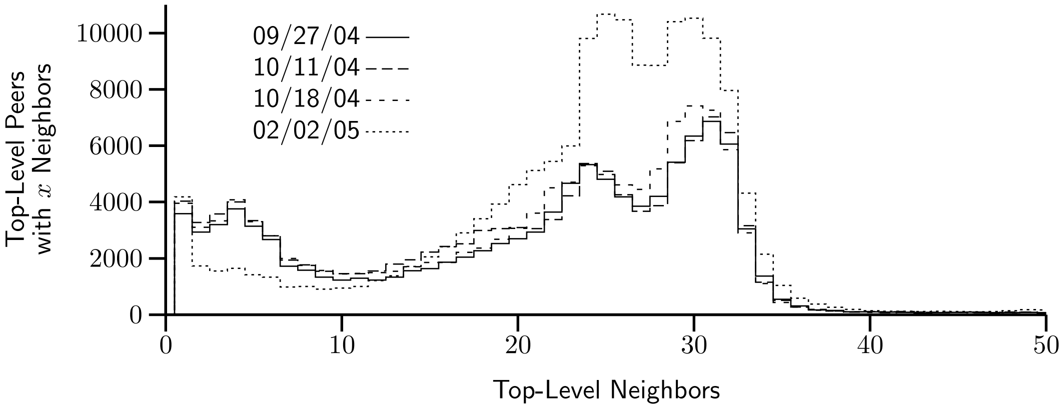
shot (labeled “Fast Crawl”). This distribution has a spike around 30 and does not follow a power-law, which would exhibit a line-like tail when plotted in log-log scale. A key question is *to what extent this difference in degree distribution is due to the change in the overlay structure versus error in captured snapshots by earlier studies.* To examine this question, we captured a distorted snapshot by a slow crawler,[1] which is similar to the 50-connection crawler used in an earlier study [6]. The depicted degree distribution based on this distorted snapshot in log-log scale is similar to the distribution previously reported in [6, Fig. 6], which they describe as multi-modal distribution with a power-law tail. To properly compare these snapshots with different sizes, the y-axis in Figure 8.1 is normalized by the number of peers in the snapshot. *To a slow crawler, peers with long uptimes appear as high degree because many short-lived peers report them as neighbors. However, this is a mischaracterization since these short-lived peers are not all present at the same time. More importantly, this finding demonstrates that using distorted snapshots that are captured by slow crawlers can easily lead to incorrect characterizations of P2P overlays.*

Figure 8.2a presents the degree distribution of top-level peers for the four snapshots presented in Table 8.1, in linear scale. Because we were unable to contact every top-level peer, the distribution in Figure 8.2a is biased slightly low since it does not include all edges.[2] To address this problem, we split the data into Figures 8.2b and 8.2c, which depict the neighbor degree distribution for reachable and unreachable peers, respectively. The data in Figure 8.2b is unbiased since it includes data only from peers we contacted successfully, *i.e.*, we discovered every edge connected to these peers. The spike around a degree of 30 is more pronounced in this figure. Figure 8.2c presents the observed degree distribution for unreachable top-level peers (*i.e.*, overloaded or NATed). This distribution is biased low since we cannot observe the connections between pairs of these peers. In this data, a much greater fraction of

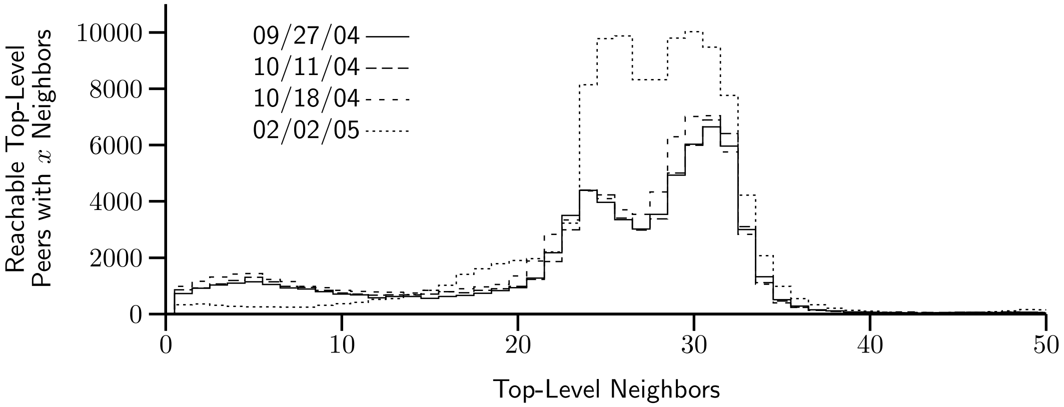
---

[1] To reduce the crawling speed, we simply limited the degree of concurrency (*i.e.*, number of parallel connections) to 60 in Cruiser.

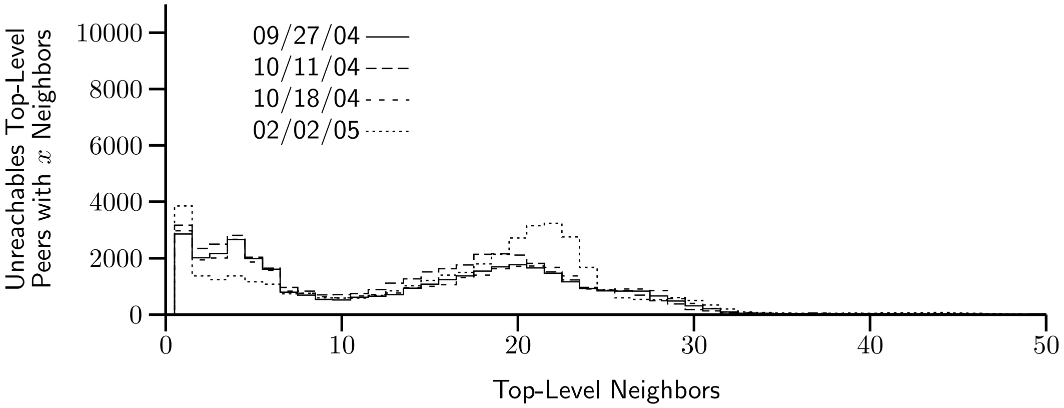
[2] The degree distribution for all the presented results is limited to 50, which includes all but a small percentage ($< 1\%$) of peers with larger degree that are discussed later.



(a) Top-level Degree Distribution



(b) Reachable Degree Distribution



(c) Unreachable Degree Distribution

FIGURE 8.2: Different angles of the top-level degree distribution in Gnutella topology

peers have an observed degree below 30. Many of these peers probably have a true degree closer to 30, with the true distribution likely similar to that in Figure 8.2b.

The degree distribution among contacted top-level peers has two distinct segments with a spike in degree around 30, resulting from LimeWire and BearShare's behavior of attempting to maintain 30 neighbors. The few peers with higher degree represent other implementations that try to maintain a higher node degree or the rare user who has modified their client software. The peers with lower degree are peers which have not yet established 30 connections. In other words, the observed degree for these peers is temporary. They are in a state of flux, working on opening more connections to increase their degree. To verify this hypothesis, we plot the mean degree of peers as a function of their uptime in Figure 8.3, which shows uptime and degree are correlated. The standard deviation for these measurements is quite large (around 7–13), indicating high variability. When peers first arrive, they quickly establish several connections. However, since node churn is high, they are constantly losing connections and establishing new ones. As time passes, long-lived peers gradually accumulate stable connections to other long-lived peers. We further explore this issue in Chapter 9 when we examine overlay dynamics.

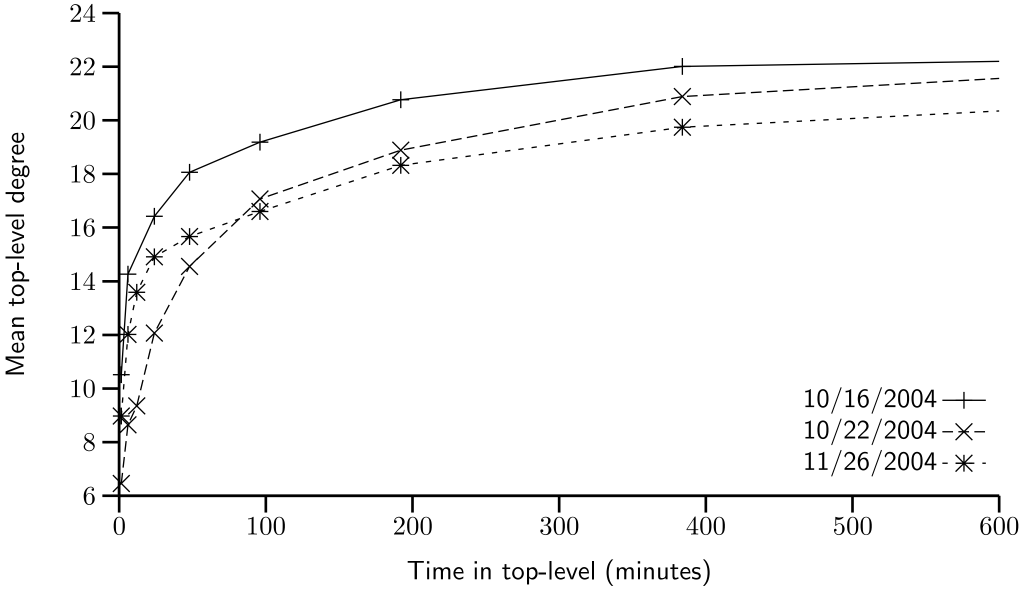


FIGURE 8.3: Mean degree as a function of uptime. Standard deviation (not shown) is large (7–13).

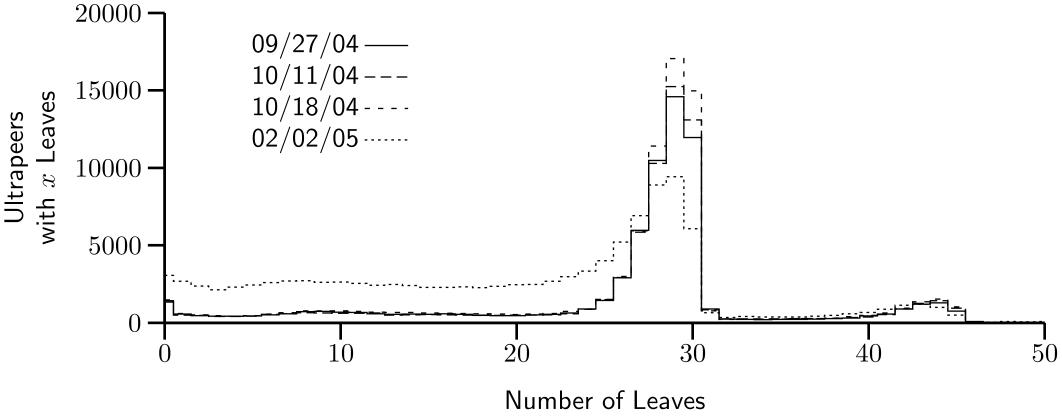
Node Degree For Leaves: To characterize properties of the two-tier topology, we have examined the degree distribution between the top-level overlay and leaves, and vice versa. Figure 8.4a presents the degree distribution of connections from ultrapeers to leaf peers. A distinct spike at 30 is visible, with a secondary spike at 45. The first two spikes are due to the corresponding parameters used in the LimeWire and BearShare implementations, respectively. This figure shows that a significant minority of ultrapeers are connected to less than 30 leaf peers, which indicates availability in the system to accommodate more leaf peers.

In Figure 8.4b, we present the degree of connectivity for leaf peers. This result reveals that most leaf peers connect to three ultrapeers or fewer (the behavior of LimeWire), and a small fraction of leaves connect to several ultrapeers. A few leaves ($< 0.02\%$, not shown) connect to an extremely large number of ultrapeers (100–3000).

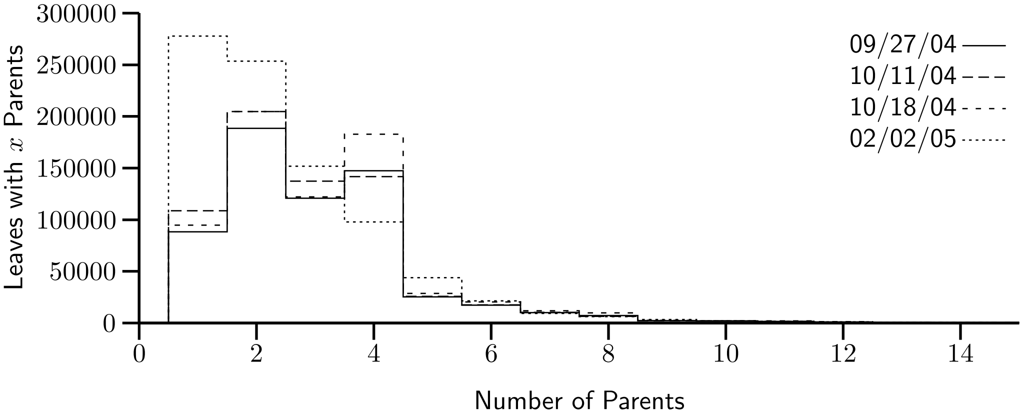
Implications of High Degree Peers: In all degree distributions in this subsection, we observed a few outlier peers with an unusually high degree of connectivity. The main incentive for increasing degree is to reduce their mean distance to other peers. To quantify the benefit of this strategy, Figure 8.4c presents the mean distance to other peers as a function of node degree, averaged across peers with the same degree. We show this for both the top-level overlay and across all peers. This figure shows that the mean path to participating peers exponentially decreases with degree. In other words, there are steeply diminishing returns for increasing degree as a way of decreasing distance to other peers.

Turning our attention to the effects of high-degree peers on the overlay, for scoped flood-based querying, the traffic these nodes must handle is proportional to their degree for leaves and proportional to the square of their degree for ultrapeers. Note that high-degree ultrapeers may not be able, or may not choose, to route all of the traffic between their neighbors. Thus, they may not actually provide as much connectivity as they appear to, affecting the performance of the overlay.

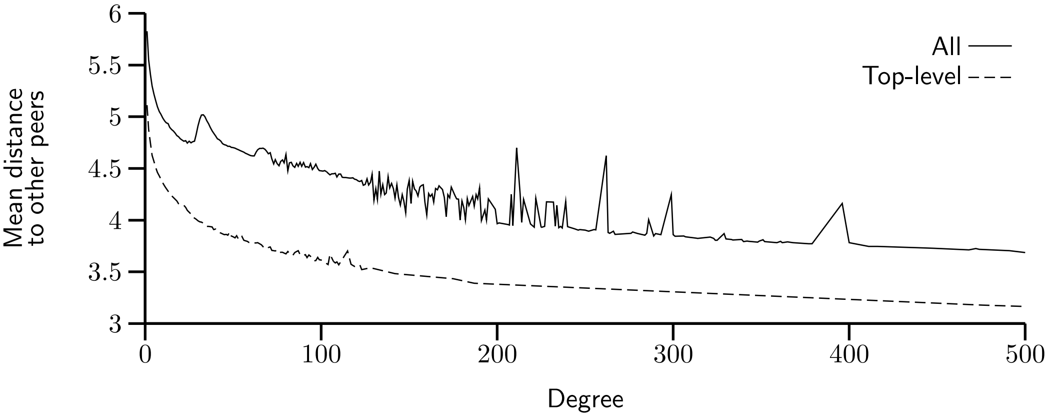
During our analysis, we discovered around 20 ultrapeers (all on the same /24 subnet) with an extremely high degree (between 2500 to 3500). These high-degree peers are widely visible throughout the overlay, and thus receive a significant portion of exchanged queries among other peers. We directly connected to these high degree peers



(a) Degree distribution from ultrapeers to leaves



(b) Leaf Parents



(c) Correlation between ultrapeer's degree and its mean distance from other ultrapeers from the 10/18/04 snapshot

FIGURE 8.4: Different angles of degree distribution in Gnutella

and found they do not actually forward any traffic.[3] We removed these inactive high degree peers from our snapshots when considering path lengths since their presence would artificially improve the apparent connectivity of the overlay.

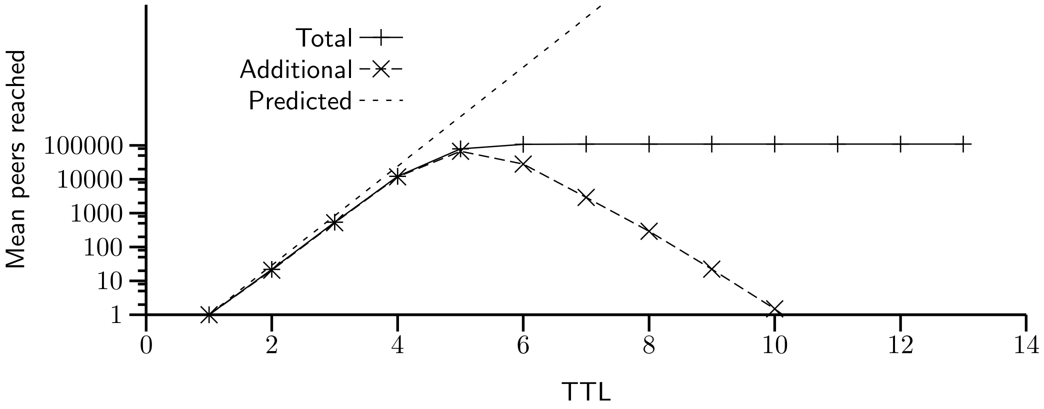
8.4 Reachability

The degree distribution suggests the overlay topology might have a low diameter, given the moderately high degree of most peers. To explore the distances between peers in more detail, we examine two equally important properties of overlay topologies that express the reachability of queries throughout the overlay: *(i)* the reachability of flood-based queries, and *(ii)* the pairwise distance between arbitrary pairs of peers.

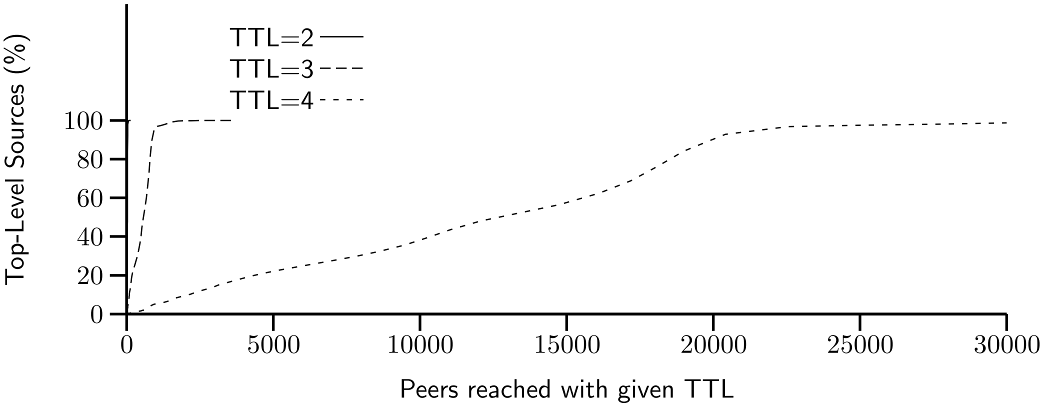
Reachability of Flood-Based Query: Figure 8.5a depicts the *mean* number of newly visited peers and its cumulative value as a function of TTL, averaged across top-level peers in a single snapshot. The shape of this figure is similar to the result that was reported by Lv et al. [9, Figure 3] which was captured in October 2000, with a significantly smaller number of peers (less than 5000). Both results indicate that the number of newly visited peers exponentially grows with increasing TTL up to a certain threshold and has diminishing returns afterwards. This illustrates that the dramatic growth of network size has been effectively balanced by the introduction of ultrapeers and an increase in node degree. Thus, while the network has changed in many ways, the percentage (but not absolute number) of newly reached peers per TTL has remained relatively stable. Figure 8.5a also shows the number of newly visited peers predicted by the Dynamic Querying formula (assuming a node degree of 30), which we presented in Chapter 2.1 (pg. 8). This result indicates that the formula closely predicts the number of newly visited peers for TTL values less than 5. Beyond 5, the query has almost completely saturated the network.

---

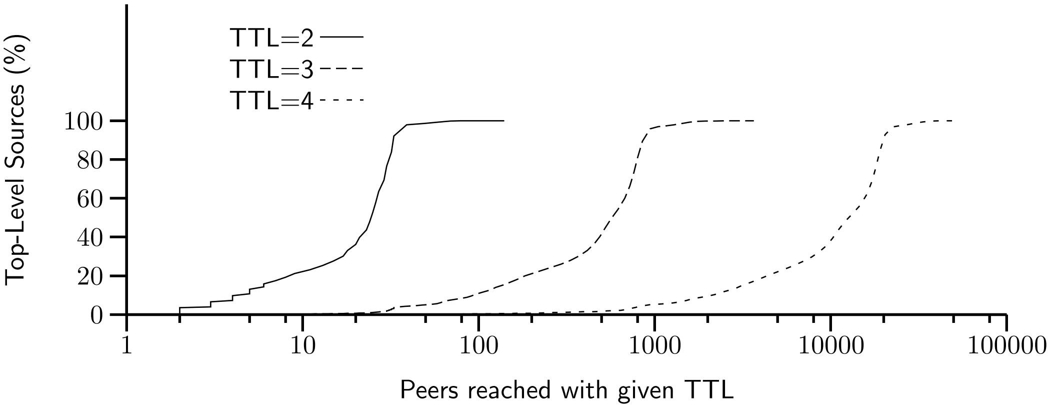
[3] To our surprise, it appears that these peers monitor exchanged messages among other participating peers. They could be trying to locate copyright infringement among Gnutella users or collecting ratings information to measure which songs consumers might like to buy.



(a) Mean Top-Level Peers Searched by TTL from the 9/27/2004 snapshot



(b) Cumulative Top-Level Peers Searched CDF



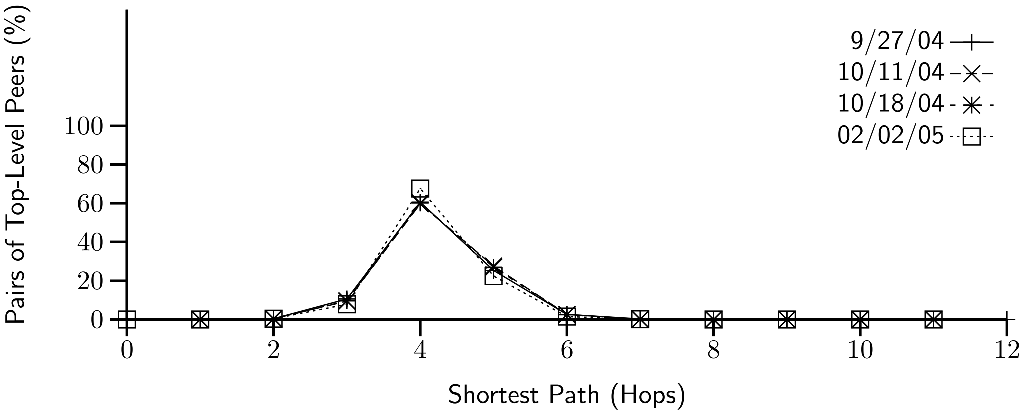
(c) Cumulative Top-Level Peers Searched CDF, log scale

FIGURE 8.5: Search radius as a function of TTL in Gnutella

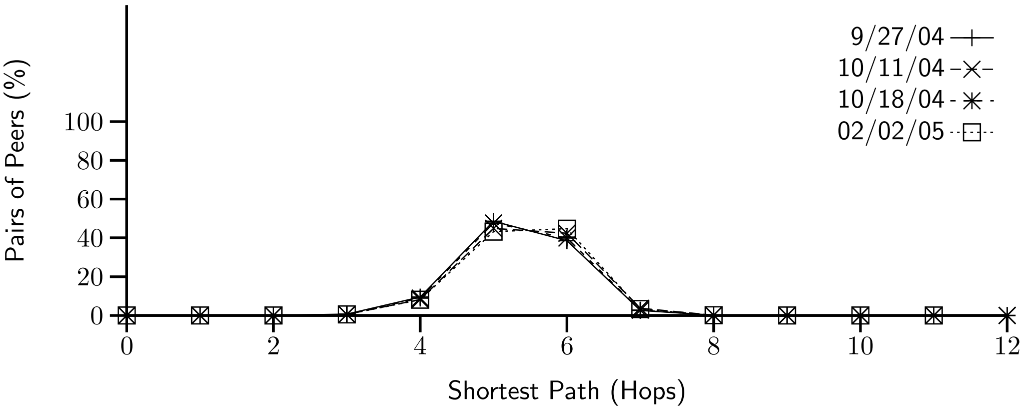
Figures 8.5b and 8.5c shows a different angle of reachability for the same snapshot by presenting the Cumulative Distribution Function (CDF) of the number of visited peers from top-level peers for different TTL values. This figure shows the distribution of reachability for flood-based queries among participating peers. Figure 8.5c uses a logarithmic x-scale to magnify the left part of the figure for lower TTL values. The figure illustrates two interesting points: First, the total number of visited peers using a TTL of n is almost always an order of magnitude higher compared to using a TTL of $(n-1)$. In other words, TTL is the primary determinant of the mean number of newly visited peers independent of a peer's location in the overlay. Second, the distribution of newly visited peers for each TTL is not uniform among all peers. As TTL increases, this distribution becomes more skewed, a direct effect of node degree. More specifically, if a peer or one of its neighbors has a very high degree, its flood-based query reaches a proportionally larger number of peers.

Pair-wise Distance: Figure 8.6a shows the distribution of shortest-path lengths in terms of overlay hops among all pairs of top-level peers from four snapshots. Ripeanu et al. [6] presented a similar distribution for the shortest-path length based on snapshots that were collected between November 2000 and June 2001 with 30,000 peers. Comparing these results reveals two differences: *(i)* the pairwise path between peers over the modern Gnutella topology is *more homogeneous in length, with shorter mean value* compared with a few years ago. More specifically, the old snapshots show 40% and 50% of all paths have a length of 4 and 5 hops whereas our results show 60% of all paths having a length of 4. *(ii)* the results from our snapshots are nearly identical with one another; whereas in [6], there is considerable variance from one crawl to another. In summary, *the path lengths have become shorter, more homogeneous, and their distribution is more stable.*

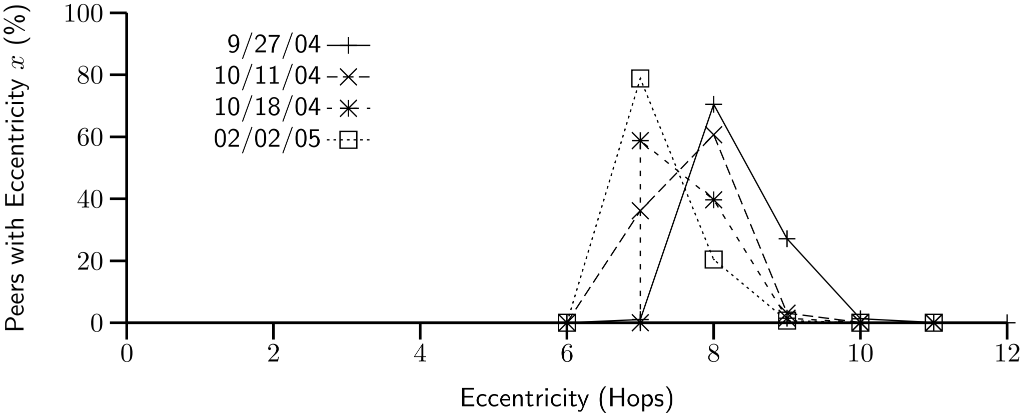
Effect of Two-Tier Topology: To examine the effect of the two-tier overlay topology on path length, we also plot the path lengths between all peers (including leaves) in 8.6b. If each leaf had only one ultrapeer, the distribution of path length between leaves would look just like the top-level path lengths (Figure 8.6a), but right-shifted by two. However, since each leaf peer has multiple parents, the path length distribution between leaves (and thus for all peers) has a more subtle relationship with



(a) Ultrapeer-to-ultrapeer shortest paths



(b) Distribution of path lengths across all pairs of peers



(c) Distribution of Eccentricity in the Top-level Overlay

FIGURE 8.6: Different angles on path lengths

Figure 8.6a. Comparing Figures 8.6a and 8.6b shows us the cost introduced by using a two-tier overlay. In the top-level, most paths are of length 4. Among leaves, we see that around 50% of paths are of length 5 and the other 50% are of length 6. Thus, getting to and from the top-level overlay introduces an increase of 1 to 2 overlay hops. **Eccentricity:** The longest observed path in these four snapshots was 12 hops, however the vast majority (99.5%) of paths have a length of 5 hops or less. To further explore the longest paths in the topology, we examined the distribution of eccentricity in the top-level overlay. The eccentricity of a peer is the distance from that peer to the most distant other peer. More formally, given the function $P(i,j)$ that returns the shortest path distance between nodes i and j, the eccentricity, E_i of node i is defined as follows: $E_i = \max(P(i,j),\ \forall j)$. Figure 8.6c shows the distribution of eccentricity in four topology snapshots. This figure shows that the distribution of eccentricity is rather homogeneous and low which is an indication that the overlay graph is a relatively balanced and well-connected mesh, rather than a chain of multiple groups of peers.

8.5 Small World

Recent studies have shown that many biological and man-made graphs (*e.g.*, collaborations among actors, the electrical grid, and the WWW graph) exhibit "small world" properties. A study by Jovanovic et al. [115] in November–December 2000 concluded that the Gnutella network exhibits small world properties as well. Our goal is to verify to what extent recent top-level topologies of the Gnutella network still exhibit small world properties despite growth in overlay population, an increase in node degree, and changes in overlay structure. As we described in Chapter 2 (pg. 15), in these graphs the mean pairwise distance between nodes (L_{actual}) is small while the clustering coefficient (C_{actual}) is high compared to random graphs (L_{random}, C_{random}) with the same number of vertices and edges, *i.e.*, L_{actual} and L_{random} are close, but C_{actual} is orders of magnitude larger than C_{random}.

Table 8.3 presents these properties for modern Gnutella. Because computing the true mean path lengths (L_{random}) is computationally expensive for large graphs, we

Graph	L_{actual}	L_{random}	C_{actual}	C_{random}
New Gnutella	4.17–4.23	3.75	0.018	0.00038
Old Gnutella	3.30–4.42	3.66	0.02	0.002
Movie Actors	3.65	2.99	0.79	0.00027
Power Grid	18.7	12.4	0.08	0.005
C. Elegans	2.65	2.25	0.28	0.05

TABLE 8.3: Small world characteristics

used the mean of 500 sample paths selected uniformly at random. We also include the information presented by Jovanovic et al. [115] and three classic small world graphs [40]. All three classic small world graphs in the table exhibit variants of these conditions. Snapshots of modern Gnutella satisfy these conditions which means that modern Gnutella still exhibits small world properties.

Comparing the clustering coefficient between modern Gnutella and old Gnutella shows that modern Gnutella has less clustering. A plausible explanation is the increased size, which provides the opportunity for more diverse connectivity to other peers. A high clustering coefficient implies a larger fraction of redundant messages in flood-based querying. The observed clustering could be a result of factors like peer bootstrapping, the peer discovery mechanism, and overlay dynamics. Further analysis is needed to better understand the underlying causes. Chapter 9 shows how peer churn is one factor that contributes to clustering.

8.6 Resilience

We also examine the resilience in different snapshots of the Gnutella overlay topology using two different types of node removal: *(i)* random removal and *(ii)* removing the highest-degree nodes first. An early study [60] conducted the same analysis on Gnutella based on a partial topology snapshot, finding that the overlay is resilient to random departures, but under highest-first node removal quickly becomes very fragmented (after removing just 4% of nodes).

Figure 8.7 depicts the percentage of remaining nodes in the topology in the largest component (y-axis) as a function of the percentage of nodes removed (x-axis), in both the random and highest-first node removal. *This figure shows the Gnutella overlay is not only extremely robust to random peer removals but also exhibits high resilience to highest-first node removal.* Even after removing 85% of peers randomly, 90% of the remaining nodes are still connected. For the highest-first case, after removing the 50% of peers with the highest-degree, 75% of the remaining nodes remain connected. There are two possible factors contributing to this difference with earlier results [60]: *(i)* the higher median node degree of most nodes in modern Gnutella, and *(ii)* a non-negligible number of missing nodes and edges in the partial snapshot of the earlier study. Our result implies that complex overlay construction algorithms (*e.g.*, [119]) are not a necessary prerequisite for ensuring resilience in unstructured overlays.

8.7 Summary

Using Gnutella, this chapter presents the first detailed characterization of an unstructured two-tier overlay topology that is typical of modern popular P2P systems, based on accurate and complete snapshots captured with Cruiser. We characterized

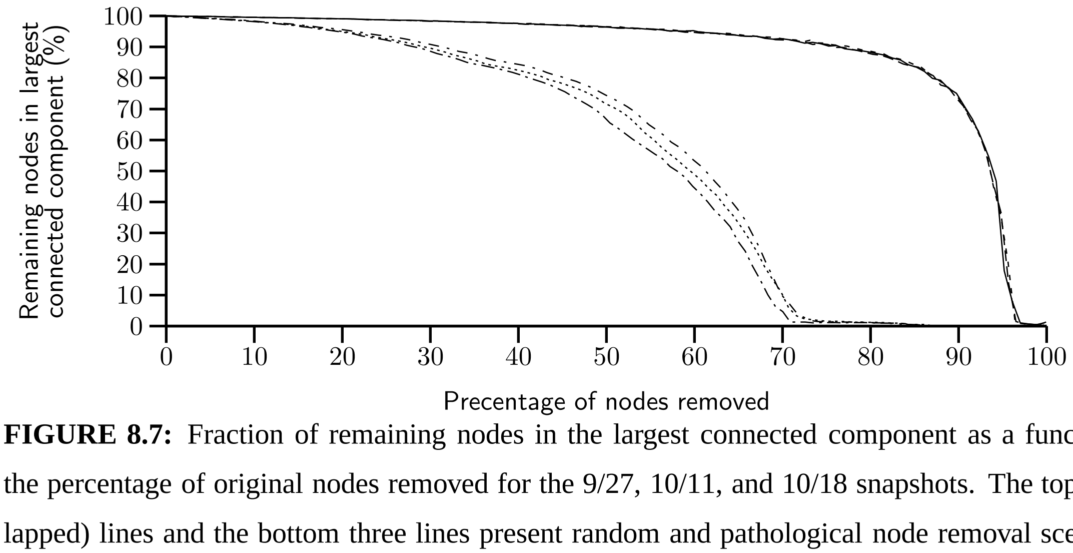


FIGURE 8.7: Fraction of remaining nodes in the largest connected component as a function of the percentage of original nodes removed for the 9/27, 10/11, and 10/18 snapshots. The top (overlapped) lines and the bottom three lines present random and pathological node removal scenarios, respectively.

the graph-related properties of individual snapshots, demonstrating the following key points:

- The degree distribution is not power-law as previously believed. Rather, most ultrapeers have close to 30 neighbors, while a significant minority have fewer neighbors.
- Shortest-path lengths are still short.
- Gnutella exhibits the high clustering coefficient of a small world.
- Gnutella is resilient to node removals, even when targeting highest-degree nodes first.

This chapter developed essential insights into the behavior of overlay topologies which are necessary to improve the design and evaluation of peer-to-peer file-sharing applications. We have made the snapshots presented in this chapter available online[4] for use by other researchers in trace-driven simulations. To date, they have been downloaded more than 80 times.

---

[4] `http://mirage.cs.uoregon.edu/P2P/root-snapshots.html`

CHAPTER 9

Dynamic Connectivity Properties

Portions © 2006 IEEE. Reprinted, with permission, from D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed DHT," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

The peer dynamics discussed in the previous chapter have secondary effects. The way that peers arrive and depart induces certain characteristics on the overlay graph. This chapter examines these effects empirically and through simulation. Our main finding is that peer dynamics lead implicitly to the emergence of a stable core of connected, long-lived peers.

Another aspect of dynamic connectivity is the way searches propagate over the overlay. We present the first empirical study of search performance in a Distributed Hash Table, finding that performance is actually better than predicted theoretically in prior work.

9.1 Stable Core

In Chapter 8, we characterized the graph-related properties of individual snapshots of the overlay topology. However, in practice the overlay topology is inherently dynamic since the peer population is constantly changing. These dynamics can significantly affect the main functionality of the overlay which is to provide connectivity and efficiently route messages (*e.g.*, queries, responses) among participating peers. Characterizing overlay dynamics enables us to examine their impact on performance of P2P applications. For example, a query or response message can be routed differently or even dropped as a result of changes in the edges of the overlay. To our knowledge, aggregate dynamics of unstructured P2P overlay have not been studied and thus these dynamics can not be incorporated in meaningful simulation-based evaluations of P2P protocols.

There are two basic causes for dynamics in the overlay topology as follows:

- Dynamics of Neighbor Selection: Two existing peers in the overlay may establish a new (or tear down an existing) connection between them. Such a change in edges is not triggered by users and thus *protocol-driven*.
- Dynamics of Peer Participation: When a peer joins (or leaves) the network, it establishes (or tears down) its connections to other participating peers in the overlay. Therefore, these changes in overlay edges are *user-driven*.[1]

Note that the user-driven dynamics of peer participation are likely to exhibit similar distributions in different P2P applications [19, 106]. Therefore, identifying the effect of user-driven dynamics on one overlay provides useful insights for the design and evaluation of other unstructured P2P overlays.

To characterize the dynamics of the Gnutella network, we investigate *(i)* whether a subset of participating peers form a relatively stable core for the overlay, *(ii)* what properties (such as size, diameter, degree of connectivity, and clustering) this stable core exhibits, and *(iii)* what underlying factors contribute to the formation and properties of such a stable core

The material in this section is adapted from material under submission to a journal, which includes material previously presented at conferences [11, 12]. The material was co-authored material with Prof. Reza Rejaie and Dr. Subhabrata Sen. The ex-

---

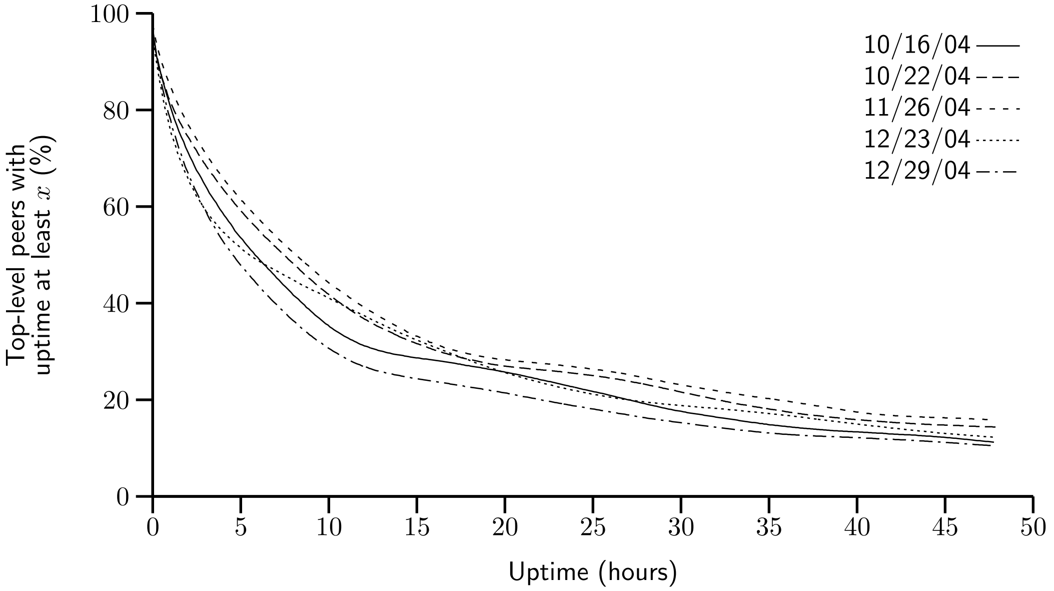
[1] Note that most P2P applications do not run as a daemon. Therefore, peer arrival/departure is a moderately reliable indication of user action. We are mindful that dynamic IP addresses could force some peers to leave and rejoin the network with a new address. Nevertheless, we group such changes as user-driven since they are beyond the control of the P2P protocol.

perimental work is entirely mine. The writing is primarily mine, with contributions by Prof. Reza Rejaie and Dr. Subhabrata Sen, who also provided technical guidance.

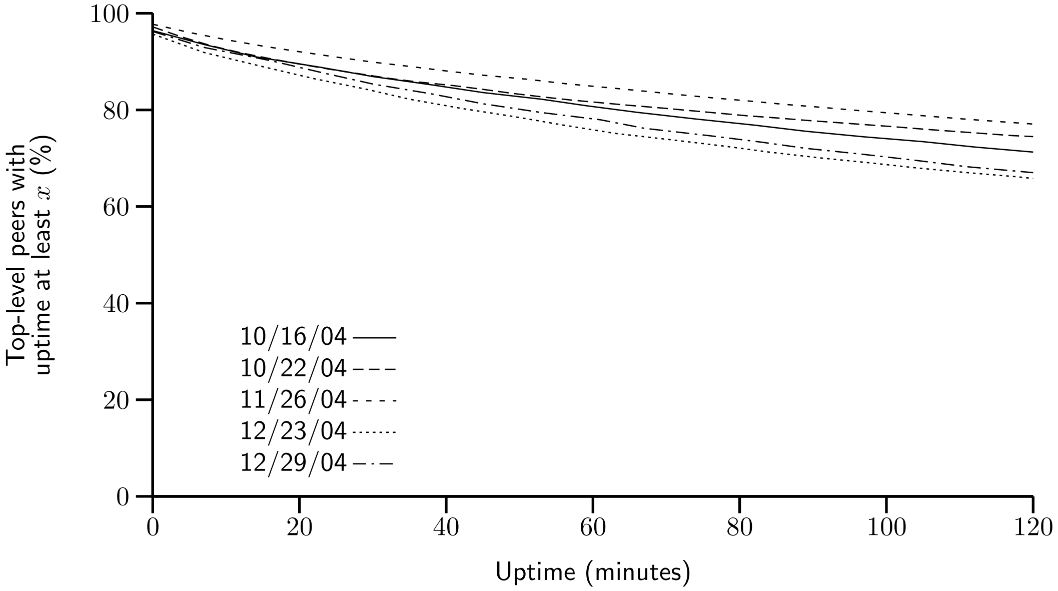
Methodology: By definition, if the overlay has a stable core, it must be composed of the long-lived ultrapeers. Short-lived peers are not stable, and leaf peers are not part of the core since they do not provide connectivity. Therefore, to identify the stable core of the overlay at any point of time, we select the subset of top-level peers who have been part of the overlay for at least τ minutes, *i.e.*, whose uptime is longer than a threshold τ. We call this subset of peers the *stable peers*, or $SP(\tau)$, and only focus on this subset in our analysis. By changing τ, we can control the minimum uptime of selected peers and thus the relative stability and size of $SP(\tau)$.

To conduct this analysis, we use several slices of our dataset where each slice represents a period of 48 hours of continuous back-to-back snapshots of the overlay topology, with hundreds of snapshots per slice. We treat the last captured snapshot over each 48 hour period as a reference snapshot. Any peer in the reference snapshot must have joined the overlay either before or during our measurement period. By looking back through the snapshots, we can determine (with accuracy of a few minutes) the arrival time of all peers that joined during the measurement period. For those peers that were present for the entire measurement period, we can conclude that their uptime is at least 48 hours. Having this information, we can annotate all peers in the reference snapshot with their uptime information. Figure 9.1a depicts the CCDF of uptime among existing peers in the reference snapshot for several slices (Figure 9.1b presents the initial part of the same graph). In essence, this figure presents the distribution of uptime among participating peers in steady state, implying that the size of $SP(\tau)$ exponentially decreases with τ. This behavior is more visible over longer time scales. Furthermore, this also implies that the total number of possible connections within $SP(\tau)$ dramatically decreases with τ. These findings are consistent with our study of peer churn in Chapter 7.

External Connectivity to/from the Stable Core: To quantify the connectivity between $SP(\tau)$ and the rest of the overlay, we examined whether peers within $SP(\tau)$ have a higher tendency to connect to each other rather than peers outside the core. To quantify any potential tendency, we generate a control graph by randomizing the



(a) Percentage of top-level peers with uptime at least x



(b) Percentage of top-level peers with uptime at least x (zoomed in)

FIGURE 9.1: Number of stable peers and their external connectivity for different τ

connections between peers. That is, given a snapshot, $G(V,E)$, we randomly generate a graph $G'(V,R)$ using the same set of peers (V) such that the degree of each peer is unchanged, *i.e.*, $(|R_{ij}\ \forall\, j| = |E_{ij}\ \forall\, j|)\ \forall\, i$. The randomized version gives us a control for the number of edges internal to $SP(\tau)$ that arise purely as a result of the degree distribution of the graph. We can then compare the number of edges internal to $SP(\tau)$ in the snapshot with the number in the randomized version as follows:

$$\frac{|E_{ij}\ \forall\, i,j \in SP| - |R_{ij}\ \forall\, i,j \in SP|}{|R_{ij}\ \forall\, i,j \in SP|}$$

This captures the percentage increase in internal edges compared to the expected value, and is plotted as a function of τ in Figure 9.2. The figure demonstrates that the longer a peer remains in the network, the more biased its connectivity becomes towards peers with the same or higher uptime. The characteristics of internal and external connectivities for $SP(\tau)$ imply that the longer a peer remains in the overlay, the more likely it establishes connections to peers with equal or higher uptimes, *i.e.*, the more biased its connectivity becomes toward peers with higher uptime. Since connections for all participating peers exhibit the same behavior, connectivity of the overlay exhibits a biased "onion-like" layering where peers with similar uptime (a

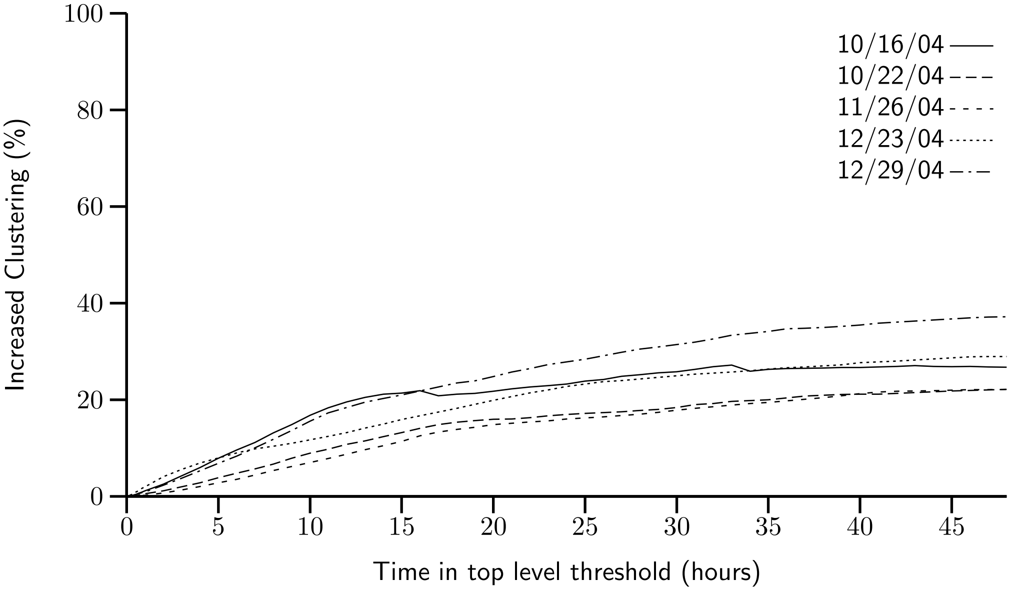


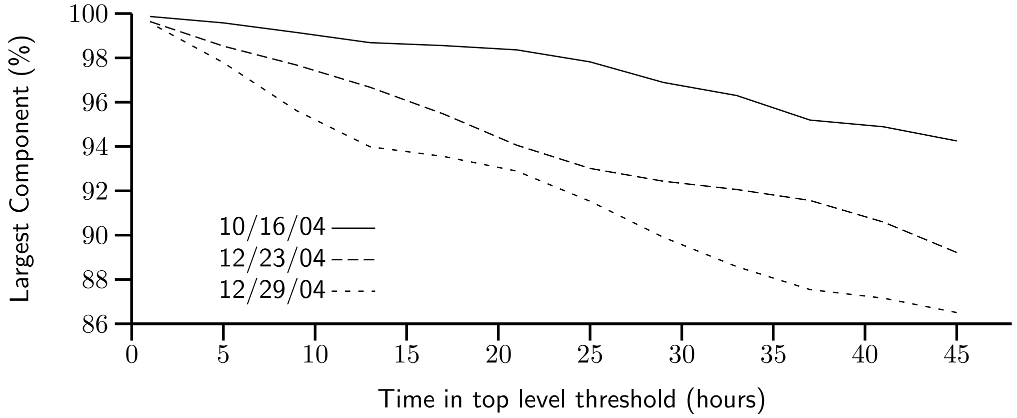
FIGURE 9.2: Percentage of increased clustering among stable nodes, relative to a randomized topology for 5 different snapshots

layer) have a tendency to be connected to peers with the same or higher uptime (internal layers of the onion). Since the size of $SP(\tau)$ decreases with τ, this means that internal layers are both smaller and more clustered.

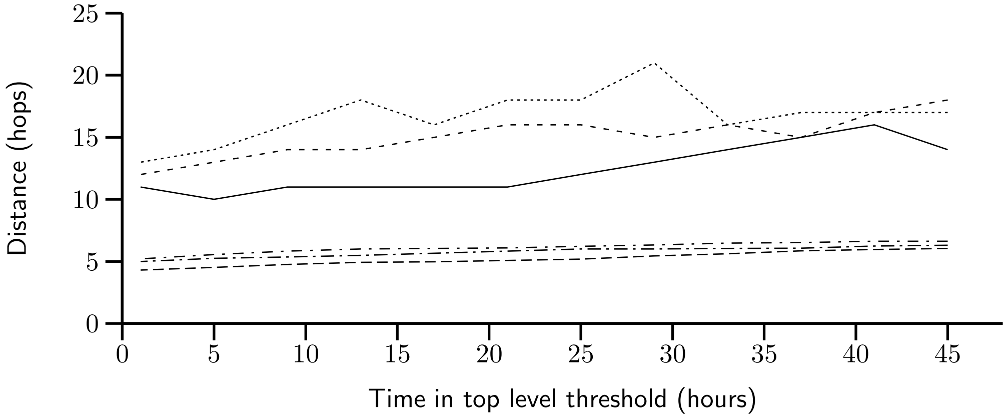
Internal Connectivity Within the Stable Core: To study different angles of connectivity among ultrapeers within $SP(\tau)$, we focus only on the connections of the overlay where both end points are inside $SP(\tau)$, *i.e.*, we remove all edges to peers outside $SP(\tau)$. We call this the *stable core* overlay or $SC(\tau)$. The first question is: *how much connectivity is there between the peers in $SC(\tau)$?* Figure 9.3a depicts the percentage of ultrapeers within $SC(\tau)$ that are in the largest connected component, as a function of τ. This figure demonstrates that while the fraction of connected peers slightly decreases with τ over long time scales, a significant majority (86%–94%) of peers within $SC(\tau)$ remain connected in one large component. The minor drop in the percentage of connected peers is due to the exponential decrease in number of peers within $SC(\tau)$, which in turn reduces the number of edges among peers, and thus affects the opportunity for pairwise connectivity.

The second question is: *how clustered and dense is the connected portion of the core overlay?* Figure 9.3b shows the diameter and characteristic (mean) path length among fully connected peers in the largest component of the stable core overlay. Interestingly, both the mean path length and the diameter of the stable core overlay remain relatively stable as τ increases, despite the dramatic drop in number of edges. Furthermore, the mean path length for the stable core overlay, even when it has a very small population (only 10% of top-level peers for τ=45h), is around 5 hops, very close to the mean path length for the entire top-level overlay (4.17–4.23 from Table 8.3). Finally, Figure 9.3c depicts the evolution of the clustering coefficient for the stable core overlay as τ increases, along with the clustering coefficient for the entire top-level overlay in the reference snapshot. This figure shows two important points: *(i)* peers within the stable core overlay are more clustered together than the entire top-level overlay on average, and, more importantly, *(ii)* connectivity among peers within the stable core overlay becomes increasingly more clustered with τ.

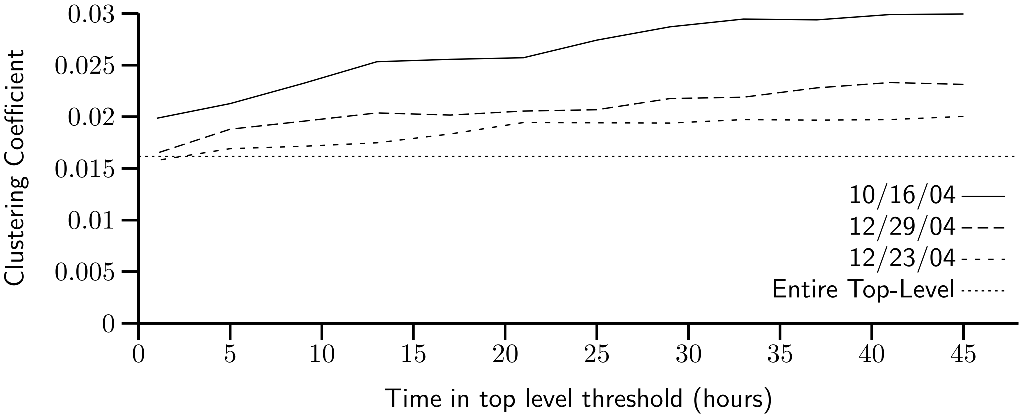
Implications of Stable and Layered Core Overlay: The onion-like connectivity of the unstructured overlay implies that all peers within the core do not depend on



(a) Percentage of peers in the stable core that are part of the core's largest connect component



(b) Diameter (top) and characteristic path length (bottom) of the largest connected component of the stable core



(c) Clustering coefficient within the largest connected component of the stable core

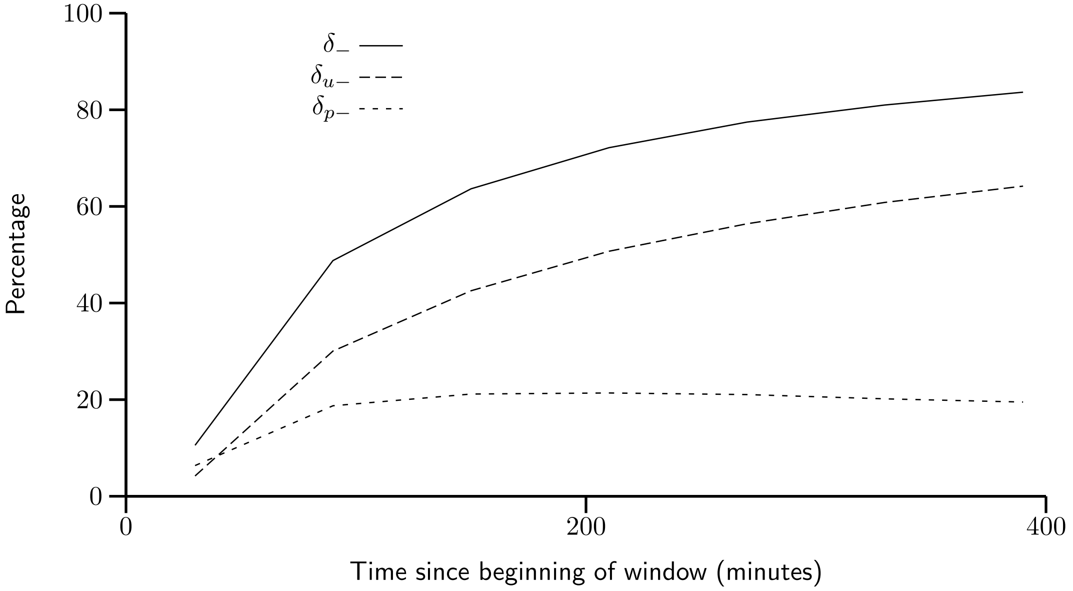
FIGURE 9.3: Different angles of connectivity with the stable core

peers outside the core for reachability. In other words, the core overlay provides a stable and efficient backbone for the entire top-level overlay that ensures connectivity among all participating peers despite the high rate of dynamics among peers outside the core.

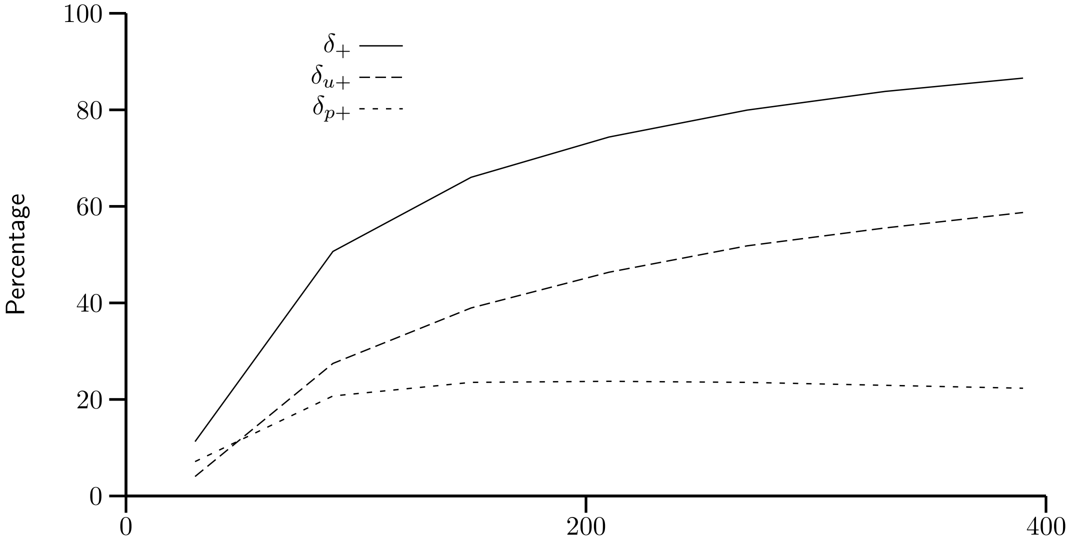
9.1.1 Examining Underlying Causes

A key question is: *how does this onion-like layered connectivity form?* To address this issue, we quantify the contribution of user-driven and protocol-driven changes to the edges of the overlay. We can distinguish protocol-driven versus user-driven changes in edges between two snapshots of the overlay as follows: if at least one of the endpoints for a changing edge has arrived (or departed) between two snapshots, that change is user-driven. Otherwise, a changing edge is protocol-driven. To answer the above question, we examine a 48-hour slice of back-to-back snapshots from 10/14/2004 to 10/16/2004, using the first snapshot as a reference. Given a slice, we can detect new or missing edges in any snapshot compared to the reference snapshot. Let δ_{p-} and δ_{u-} (δ_{p+} and δ_{u+}) denote the percentage of missing (and new) edges in a snapshot due to protocol-driven (p) and user-driven (u) causes, relative to the reference snapshot. Note that δ_p and δ_u are by definition cumulative since the reference snapshot does not change. Figure 9.4a and 9.4b depict δ_{-}=δ_{p-}+δ_{u-} and δ_{+}=δ_{p+}+δ_{u+}. The top graph (δ_{-}) shows that around 20% and 30% of edges in the overlay are removed due to protocol-driven and user-driven factors during the first 100 minutes, respectively. After this period, almost all removed edges are due to departing peers (*i.e.*, user-driven). Similarly, from the bottom graph (δ_{+}), many edges are added during the first 100 minutes due to both protocol-driven factors and the arrival of new peers. After this period, almost all new edges involve a newly arriving peer (*i.e.*, user-driven).

These results show two important points: First, each peer may establish and tear down many connections to other peers during the initial 100 minutes of its uptime. But peers with higher uptime (*i.e.*, peers inside $SC(\tau)$ for $\tau \geq 100$ min) maintain their connections to their remaining long-lived neighbors, and only add (or drop)



(a) Removed edges



(b) Added edges

FIGURE 9.4: Contribution of user- and protocol-driven dynamics in variations of edges in the overlay

connections to arriving (or departing) peers. This behavior appears to explain the formation of the biased onion-like layering in connectivity within the overlay. Second, user-driven dynamics are the dominant factor in long-term changes of the overlay. Since dynamics of peer participations exhibit rather characteristics in different P2P systems [19], other Gnutella-like overlays are likely to show similar behavior. We plan to conduct further investigations to better understand the underlying dynamics that contribute to this behavior.

9.2 Query Properties

The behavior of queries is the second aspect of the connectivity properties of peer-to-peer overlays examined in this dissertation. The topics will not be covered exhaustively, however the most common query techniques will be examined. As structured overlays use a more complex query mechanism, they will be examined in greater detail.

Unstructured Overlays: For unstructured overlays, the basic search operation is flooding within the scope of a certain time-to-live (TTL). The searcher sends out a message to each of its neighbors, who relay it to each of their neighbors, etc. The search ends when it is TTL hops from its origin. Modern Gnutella clients used a more advanced mechanism called Dynamic Querying [21], which uses flooding as its basic underlying mechanism.

To understand flooding behavior, we must examine the number of peers reached as a function of TTL, as a distribution over all possible starting peers. As another angle, we can examine the distribution of path-lengths between pairs of peers. This topic was covered in Chapter 8 (pg. 153), while examining other connectivity properties of the Gnutella network.

Structured Overlays: In structured overlays (DHTs), each peer has an overlay address and a routing table. When a peer performs a query for an identifier, the query is routed to the peer with the closest overlay address. To guarantee good query performance (typically $O(\log |V|)$ hops), every DHT scheme enforces rules for how peers select neighbors. However, these performance bounds have only been proven in the case when peers do not arrive and depart. To show robustness to peer dynamics, researchers have been limited to simulations and small-scale experiments, which may not accurately model real-world peer dynamics. In addition to affecting query performance, peer dynamics can also result in inconsistent routing tables so that the same query may lead to different peers, depending on where in the overlay the query originates.

For these reasons, a study that examines the query performance over a widely-deployed DHT is necessary to examine the scalability and performance of DHTs in the real world.

There are two classes of solutions to cope with the effects of churn on DHTs: *(i) DHT-based*: DHTs can incorporate various techniques to actively improve their resiliency to churn by increasing the degree of redundancy or the frequency of updates for the routing table at each peer; *(ii) Client-based*: Alternatively, a client operating over an inaccurate DHT can improve its lookup efficiency by conducting lookups in parallel and cope with lookup inconsistencies by active replication of content.

Previous studies have examined both DHT-based [5, 27] and client-based [24, 69] solutions as well as the interactions and trade-offs between them [29]. All of the previous studies have used either simulation, analysis, or small-scale experiments to study these issues. However, the characteristics of real-world user dynamics are not well understood, making it unclear how well simulation-based analysis of DHTs represents real-world behavior. Section 9.2.7 discusses the related work in more detail.

This section presents a measurement-based characterization of routing table inaccuracy and its impact on lookup performance in a widely-deployed DHT, namely *Kad*. Kad is an open, Kademlia-based [24] DHT with more than 1 million concurrent users that has been recently deployed by the popular eMule[2] file-sharing application to improve efficiency of search in the face of a growing user population. Section 9.2.1 presents an overview of Kademlia and Kad.

---

[2] eMule began as an open-source alternative for the eDonkey unstructured network.

In Section 9.2.2, we begin our study by comparing the theoretically predicted performance with empirical measurements. Surprisingly, performance is better in practice, suggesting oversights in the theory. In Section 9.2.3, we establish an analytical framework to quantify the effect of routing table richness on lookup performance and show that Kademlia's k-buckets improve lookup performance. In Section 9.2.4, we gather additional empirical data about Kad's routing tables and demonstrate that observed performance is in agreement with our new theoretical predictions.

In Section 9.2.5, we examine two classes of parallel lookup techniques to improve lookup efficiency over Kad. Toward this end we developed a new tool called *kLookup*, which emulates a lookup from *any* source peer to *any* destination ID without requiring local access to the designated peers for these IDs. Furthermore, leveraging the iterative lookup scheme in Kad, kLookup enables us to empirically examine different parallel lookup techniques and identify major design trade-offs. Finally, in Section 9.2.6 we characterize the frequency of inconsistent lookup results in Kad. We then explore how the degree of replication improves lookup consistency.

Our main contributions can be summarized as follows:

- **Analytical Framework**: We develop an analytical framework for computing the average performance of lookups for prefix-matching DHTs. This leads to the result that redundancy in routing tables, such as Kademlia's k-buckets, directly improves mean lookup performance by reducing hop count
- **New Tools**: *(i)* kFetch, a tool for extracting the routing table from Kad peers, *(ii)* kLookup, a parameterized tool for performing lookups over Kad using a variety of lookup algorithms
- **Empirical Findings**: *(i)* Validating the predictions of our analytical framework, *(ii)* Locating the sweet spot for the degree of lookup parallelism to improve lookup efficiency, *(iii)* Locating the sweet spot for the degree of replication to overcome routing table inconsistencies

While this study is centered around Kad, our analysis, methodologies, tools and findings are applicable to many other DHTs with minor adjustments. To address the wider applicability of our work, we briefly discuss how some issues can be pursued in the context of other DHTs. Our extensive examination of eMule's source code also revealed several bugs [120–122], some of which were fixed in the next revision.

The material in this section is adapted from material under submission to a journal, which includes material previously presented at a conference [14]. The material was co-authored material with Prof. Reza Rejaie. The experimental work, analysis, and writing are entirely mine. Reza Rejaie provided technical guidance and editorial assistance.

9.2.1 Background

We first present some background on Kademlia, since it forms the basis for the Kad network that we use for our empirical study. Like most DHTs, peers in Kademlia each have an identifier that is assigned either uniformly at random or via a cryptographic hash. To determine the distance between two peers, Kademlia uses a unique "XOR metric", the bitwise XOR of their identifiers. For example, the distance between 0100 and 0111 is 0011 (or 3).

The XOR metric places Kademlia in the general class of prefix-matching DHTs, such as Pastry [26] and Tapestry [123]. At the high-level, prefix-matching DHTs all work in the same way. A *lookup* for a target identifier must locate the peer with the longest matching prefix (measured in bits). We can view the distance between two identifiers as the index of the highest-order mismatched bit. For example, 0100 and 0000 have a distance of 3, because the third bit is the highest-order mismatched bit. A lookup consists of a sequence of *lookup steps* (or hops). Each step must lead to a peer with a longer matching prefix in order to guarantee forward progress. A lookup terminates when no further forward progress can be made. For a network of n peers, most peers will be around $\log_2 n$ bits apart, and the expected number of steps to perform a lookup is $\log_2 n$, assuming one bit of improvement per step. If we can guarantee b bits of improvement at each step, then each lookup requires $\frac{\log_2 n}{b}$ steps. We call b the *symbol size*, and in basic Kademlia $b = 1$. Section 9.2.3 examines the impact of different choices for b on lookup latency (in hops) and route table size.

As IP is also a prefix-matching protocol, we borrow some terminology from IP to describe Kademlia routing tables. Each route in a Kademlia routing table is labeled with a *subnet address* and *mask*. When performing a lookup for a key, the most-specific routing table entry with a matching subnet is used, just as in IP routing. In this paper, the familiar "slash-notation" specifies the number of bits in the mask (*i.e.*, "/3" means an ID must match the highest-order 3 bits of the subnet address). In Kademlia, the routing table is structured to contain one route per address bit, with increasingly specific masks. The subnet addresses are the same as the ID of the peer hosting the routing table. This structure is sufficient to guarantee forward progress. The routing table structure can be viewed as a binary tree, as shown in Figure 9.5a (pg. 181). For example, consider a Kademlia network using 4-bit identifiers[3] and a particular peer with the address 0000. There are route table entries for the following address–mask pairs: 0000/0, 0000/1, 0000/2, 0000/3, 0000/4. Because more-specific routes are always preferred, the routing table entries are effectively for the following address–mask pairs: 1000/1, 0100/2, 0010/3, 0001/4, 0000/4. In other words, the 0000/0 line will only contain 1000/1 addresses since any 0000/1 address would map to one of the more specific entries.

The routing tables in all the Kademlia peers collectively form one large binary tree, with each peer containing a fraction ($O(\frac{\log n}{n})$) of it. During a lookup, each routing step pivots to a different peer which is one bit closer to the target, guaranteeing that the lookup requires at most $O(\log n)$ steps.

For redundancy purposes, each routing table entry (or node in the binary tree) contains a list, called a k-bucket, of k matching contacts. Each contact includes the Kademlia ID, IP address, and port of the remote peer. Thus, each lookup step has a choice of k different contacts for the next step. Section 9.2.3 examines some of the consequences for choosing different values of k. We note that k-buckets could be adapted for use in other types of DHT as well.

Kademlia makes use of parallel routing to speed up lookups, as do EpiChord [124] and Accordion [69]. Issuing α lookup requests at a time avoids long waits while

---

[3] In practice no DHT would use such a small identifier space, but it's more tractable for illustrative purposes.

departed peers time out and also increase the probability of finding low-latency peers. Section 9.2.5 examines using different values of α in Kad.

Kademlia uses iterative routing, where the client is responsible for the entire lookup process. At each step, the client sends a *lookup request* to the next-hop peer and waits for a *lookup reply*. The reply lets the client know what the next hop is. Iterative routing contrasts with recursive routing, where the lookup request is forwarded automatically from one peer to another. While it has been shown that recursive routing typically has lower latency [125], iterative routing has several useful practical properties:

Fate-Sharing:	Lookup messages cannot be lost due to the departure of an intermediate peer holding the lookup request [126].
Debugging:	Iterative routing is easier to debug since information at each step is reported back to the client performing the lookup.
Compartmentalization:	Iterative routing decouples route table maintenance and lookup technique, allowing them to be studied and improved independently in a deployed network. Our tool, kLookup, uses this division to evaluate a variety of lookup techniques directly over the existing Kad network, as shown in Sections 9.2.5 and 9.2.6.
Route Table Extraction:	Iterative routing allows us to download the entire routing table of any peer. We make use of this feature in our tool, kFetch, described in Section 9.2.2.3.

In summary, the key properties of Kademlia (and thus Kad) are as follows: *(i)* routing by prefix-matching, *(ii)* redundancy in routing tables (k-buckets), *(iii)* parallel routing, and *(iv)* iterative routing. Redundancy, parallel routing, and iterative routing could be incorporated into most varieties of DHT. For example, EpiChord is a variant of Chord with parallel routing. Prefix-matching is an intrinsic property of Kademlia's design, which it shares with a number of other DHTs such as Pastry and Tapestry.

Kad is a Kademlia-based DHT network for file-sharing, composed primarily of eMule clients. While Kad is based on Kademlia, Kad uses a slightly different routing table structure, described in detail in Section 9.2.3. Kad has approximately 1 million simultaneous users, plus many more firewalled peers who utilize the Kad DHT for lookups but do not participate in the DHT structure. For each file an eMule client shares, the client computes the hash of each word in the filename and publishes information about itself and the file to the peers responsible for the hashes. When an eMule user enters a keyword search, eMule computes the hash of the first keyword and initiates a lookup for the hash. The lookup returns a set of endpoints to which the client submits the full keyword list. Those peers process the query and return a set of matching results.

9.2.2 Theory versus Practice

One of the purposes of our work is to compare how well DHTs work in practice with theoretical expectations. The theoretical analysis in the Kademlia paper predicts an expected lookup cost of $\frac{\log_2 n}{b}$ hops [24]. In practice, some routing table entries will be out of date or missing, delaying forward progress and increasing the lookup cost. The open question is: *How much worse is practice than theory?*

To answer this question, we must first compute the theoretical performance for Kad by finding b and n. We can obtain the value of b (the symbol size) through source code inspection. The value of n (the population size) we measure empirically, as described below. Once we compute the theoretical performance, we empirically measure the performance in practice and compare.

9.2.2.1 Estimating Network Size

Our first step is to estimate the size of the Kad network (n). An obvious approach would be to crawl the Kad network to capture a snapshot of the entire population of peers. However, accurately capturing the *entire* Kad network is challenging. Because peers arrive and depart while the crawler runs, any snapshot will result in an inflated population count as it will record a large number of short-lived peers that are not simultaneously present. The magnitude of this error grows with the length of the crawl. Due to Kad's large size, and the large size of individual routing tables, rapidly capturing a snapshot of the whole network is impractical. However, a crawler can take a snapshot of a subnet quickly (*i.e.*, with little error). Since Kad identifiers are selected uniformly at random, any subset of the ID space (such as a subnet) is a representative sample of the total population. Multiplying the measured size of a subnet by the number of such subnets yields an estimate of the population size. By taking the mean over many such samples, we can get a good estimate for n.

In Chapter 4, we developed a parallel peer-to-peer overlay crawler, called Cruiser. Given a Kad overlay subnet as an input (*e.g.*, 0x5cd/12), Cruiser walks the DHT structure to capture a snapshot of all the active peers with IDs in the specified subnet. For example, it can capture a /10 subnet with roughly 1000 peers in around 3–4 minutes and a /12 subnet with roughly 250 peers in around one minute. During June of 2005, we captured the population size for several hundred randomly selected subnets with Cruiser. Our measurements reveal that the Kad network had a mean population size of approximately 980,000 concurrent peers at that time.

9.2.2.2 Performance in Theory

Close examination of the eMule 0.46a source code reveals that Kad is based on Kademlia uses a 4-bit symbol for the first hop and 3.25-bit symbols for each additional hop. The $\frac{1}{4}$ bit is due to the fact that Kad uses unbalanced subtrees. We also validated our understanding of the source code with empirical observations of its operation. Using $n = 980,000$, we can now compute the lookup cost in Kad:

$$\text{lookup cost in Kad: } 1 + \frac{\log_2 n - 4}{3.25} = 5.89 \text{ hops.}$$

Now that we have found the theoretical performance, we move on to measuring the performance in practice.

9.2.2.3 Performance in Practice

To examine performance in practice, we need to measure the number of hops per lookup from peers to destination IDs. While we could instrument a single peer and search for random destinations, we would prefer to examine the lookup cost from different peers. Towards this end, we develop two new tools: kLookup and kFetch.

kLookup: To examine different lookup strategies, we developed a new tool, called *kLookup*, which emulates a lookup from *any* source peer to *any* destination ID without requiring local access to the source peer. To emulate a lookup from a random source peer, kLookup takes the following steps. First, it uses a local Kad routing table to locate the peer closest to a randomly generated ID (*i.e.*, the source peer), then it extracts the routing table of the source peer using kFetch. Finally, it performs a lookup to the destination ID using the extracted routing table of the source peer. We validated kLookup by comparing tcpdump traces of kLookup and eMule lookup using the same destination ID.

kFetch: To download the routing table needed for kLookup, we developed a new tool called *kFetch*. The routing table of the target peer must be downloaded quickly in order to minimize any error due to ongoing churn (*i.e.*, the routing table changes in real time). There are two challenges to downloading a routing table efficiently: *(i)* the rate of requests (which are UDP messages) must be properly paced to rapidly download the table without causing excessive network congestion, and *(ii)* lookup messages must request the right IDs to extract a peer's routing table with the minimum number of messages. kFetch implements congestion control using a variant of the SACK TCP algorithms to determine the proper rate for issuing requests. kFetch computes the routing table structure of the target peer according to Kad's rules for populating them and generates a query for each k-bucket the peer may have. This strategy could be used to extract the routing table in any DHT that uses iterative routing. In addition, kFetch examines the returned data to determine when a branch of the tree is empty, and will not issue queries for provably empty subtrees.

Measured Performance: Using kLookup, we collected measurements from hundreds of randomly selected (source peer, destination ID) pairs, finding a mean lookup cost of 3.2 hops per lookup. Surprisingly, the performance in practice (3.2 hops) is significantly better than the performance predicted by theory (5.9 hops)! As we expect the theoretical expectation to be a lower-bound on lookup cost, this result suggests there is something wrong with the theory. In the next section, we examine the theory more closely and revise it to account for the effects of k-buckets. Using the revised theory, we return to comparing performance in theory with performance in practice in Section 9.2.4.

9.2.3 Analysis of Kademlia's k-Buckets

In this section, we first establish an analytical framework to examine the effect on lookup performance of adding extra contacts to routing tables. We derive a formula for computing the typical number of hops needed to perform a lookup as a function of the quantity and structure of the extra contacts, and use the formula to explore trade-offs between different methods for increasing the richness of routing tables.

Every DHT has some structure that determines a peer's potential neighbors based on identifiers. For example, in basic Kademlia a peer must have a neighbor with a different high-order ID bit, a neighbor with a matching first bit and a different second bit, a neighbor with the first two bits matching and a different third bit, etc. We call each address–mask pair a *bucket* (following the Kademlia terminology) where each bucket contains address information, called *contacts*, for several neighbors. A bucket with k contacts is called a k-bucket. In the base case, a DHT only contains enough information to perform the lookup in $\log_2 n$ steps. In prefix-matching DHTs such Kademlia, this implies a symbol size of $b = 1$ and one contact per bucket. In general, the expected number of steps required to perform a lookup is given as follows:

$$\text{steps per lookup} = \frac{\log_2 n}{\text{bits improved per step}} \tag{9.1}$$

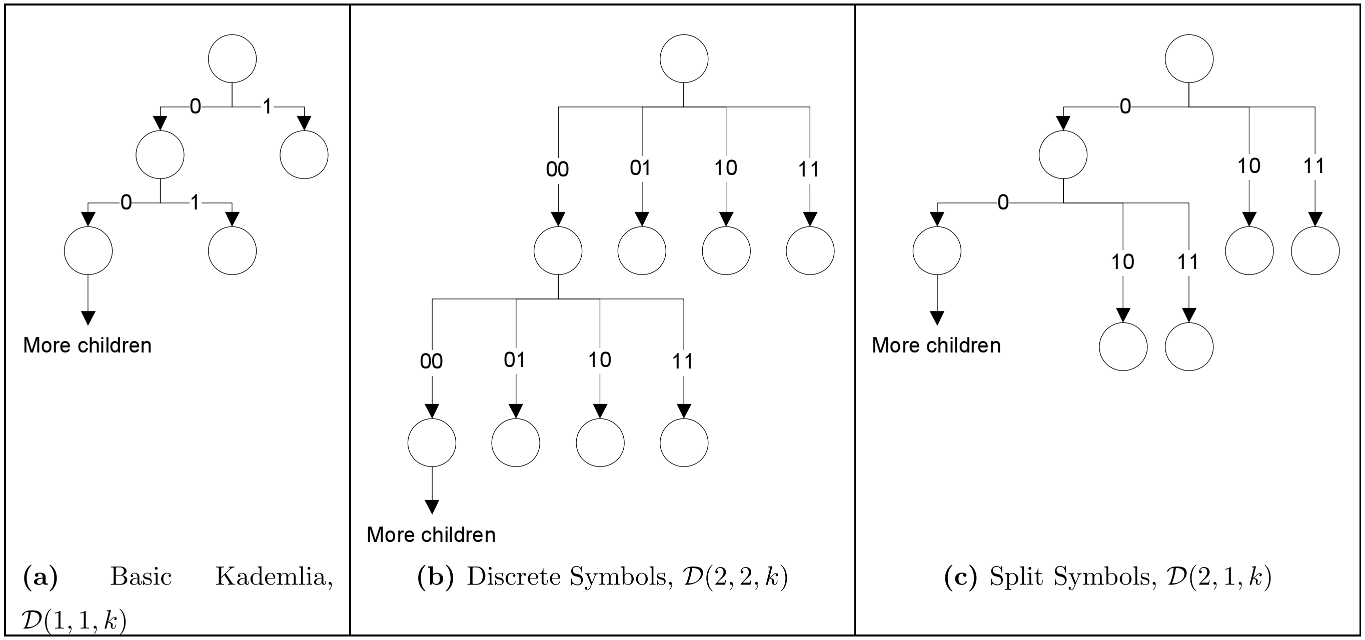
A DHT can enrich the routing table structure beyond this base case by either 1) *adding more buckets* or 2) *adding more contacts per bucket.* By adding more buckets, a DHT can guarantee that a larger number of bits will be improved at each step, thereby decreasing the number of hops for a lookup. For example, Pastry [26] uses a default symbol size of $b = 4$ which guarantees 4 bits will be improved at each step. Tables in Chord can also be enriched in this way [29].

Adding more contacts per bucket is used to guard against churn, an approach employed by DHTs such as Kademlia [24] and Tapestry [123]. By having other contacts handy, a peer can more quickly repair its routing table when a failure is detected. Furthermore, as observed in [24], with heavy-tailed session times, storing backups and only evicting unresponsive peers implicitly leads to a set of peers with good uptime characteristics. Finally, multiple contacts per route allow for the use of parallel routing.

To examine the benefits and costs of the above two approaches for enriching routing tables, we analyze their impact in the context of Kademlia. Our analysis also directly applies to other prefix-matching systems such as Pastry and Tapestry, where we can quantify the improvement at each step in terms of the number of matching bits. For other DHTs that use a different basic geometry, our analysis could be adapted by modifying the formulas to reflect the appropriate distance metric.

There are two different approaches for adding more buckets to a routing table, both of which improve the number of lookup hops from $\log_2 n$ to $\frac{log_2 n}{b}$:

- *Discrete Symbols*: With this approach, illustrated in Figure 9.5b, each interior node points to $2^b - 1$ buckets and an additional interior node. When searching a routing table, a peer begins by checking the first b bits. If all of them match the peer's ID, then it proceeds to the next b bits (*i.e.*, the next interior node). Otherwise, it proceeds immediately to the appropriate bucket. Using Discrete Symbols increases the routing table size from $\log_2 n$ rows of one k-bucket each to $\frac{log_2 n}{b}$ rows of $2^b - 1$ k-buckets each. This is the approach used in Kademlia and Pastry.
- *Split Symbols*: With this approach, illustrated in Figure 9.5c, each interior node points to 2^{b-1} buckets and an additional interior node. When searching a routing table, a peer begins by checking the first single bit. If it matches the peer's ID,



(a) Basic Kademlia, $\mathcal{D}(1,1,k)$

(b) Discrete Symbols, $\mathcal{D}(2,2,k)$

(c) Split Symbols, $\mathcal{D}(2,1,k)$

FIGURE 9.5: Routing table structures

then it proceeds to the next bit (*i.e.*, the next interior node). Otherwise, it examines the next b bits and proceeds to the appropriate bucket. Using Split Symbols increases the routing table size from $\log_2 n$ rows of one k-bucket each to $\log_2 n$ rows of 2^{b-1} k-buckets each. This is the approach used in Kad.

To compare and contrast these approaches for organizing routing table contacts, we create a general framework for analyzing the performance of both Discrete Symbols and Split Symbols. A system in the framework is uniquely characterized by three properties:

Symbol size (b):	the number of bits leading to a leaf node (*i.e.*, a bucket)
Resolution (r):	the number of bits leading to an interior node
Bucket size (k):	the number of contacts per bucket

We define $\mathcal{D}(b, r, k)$ as a system which uses b-bit symbols with r-bit resolution and k-buckets. $\mathcal{D}(1, 1, k)$ is the basic Kademlia approach, $\mathcal{D}(b, b, k)$ is the Discrete Symbol approach, and $\mathcal{D}(b, 1, k)$ is the Split Symbol approach used in Kad. Each routing table has $\frac{log_2 n}{r}$ rows of $2^b - 2^{b-r}$ k-buckets, for a total size of $k(2^b - 2^{b-r})\frac{log_2 n}{r}$ contacts. Normalizing by a factor of $\log_2 n$ yields a normalized size of $k\frac{2^b - 2^{b-r}}{r}$.

Most prior work on DHTs[4] is concerned exclusively with the case where the selected contact will not match any *additional* bits of the target identifier. For example, consider searching for the key 111 in the routing table of peer 000 with the base $b = 1$ system. The peer looks in the bucket with the prefix 1, and returns a contact which we know matches the first bit of the key. However, that contact could be any of the peers 100, 101, 110, or 111. In other words, there's a $\frac{1}{2}$ chance of improving at least 1 extra bit, a $\frac{1}{4}$ chance of improving at least 2 extra bits, and so on. More precisely, the probability of improving at least δ bits is:

$$Pr[X \geq \delta] = \frac{1}{2^\delta} \tag{9.2}$$

Due to these random improvements, the average-case is better than that given in prior work. In particular, the key insight is that large buckets ($k > 1$) improve

---

[4] To our knowledge, the only work on DHTs which has considered the impact of random improvements is Chord [82].

the probability of *randomly* finding a contact with more matching bits since there are more options to choose from. As we will show, the average number of improved bits increases logarithmically with k, making the performance boost of increasing k comparable to the performance boost of increasing b. Random improvements only help if we advance by another interior node; that is, we can only randomly improve in multiples of r bits. Generally, for a k-bucket the probability of improving by at least δ extra r-bit symbols is:

$$F(\delta, r, k) = Pr[X \geq \delta] = 1 - \left(1 - \frac{1}{2^{r\delta}}\right)^k \tag{9.3}$$

and the probability of improving by *exactly* δ extra r-bit symbols is:

$$f(\delta, r, k) = Pr[X = \delta] = F(\delta, r, k) - F(\delta + 1, r, k) \tag{9.4}$$

The key question is: *how many additional bits improve on average due to randomness?* Since we know the probability of improving exactly δ additional symbols ($f(\delta, r, k)$), we can compute the average number of extra bits improved by finding the average value of δ and multiplying by the number of bits per symbol (r) as follows:

$$\text{extra bits improved per step: } m(r, k) = r \sum_{\delta=0}^{\infty} \delta \cdot f(\delta, r, k) \tag{9.5}$$

$$\text{total bits improved per step: } t(b, r, k) = b + m(r, k) \tag{9.6}$$

Note that $m(r, k)$ is actually decreasing in r due to the r term in Formula 9.3. While we were unable to find a simple closed form for $m(r, k)$, it can be computed numerically without difficulty. With 1-bit resolution ($r = 1$), $m(1, k)$ there are asymptotically $\log_2 k + 0.3327$ extra bits of improvement, somewhat exceeding this value for lower k. Significantly, for the base case $\mathcal{D}(1, 1, 1)$ of no additional routing table entries, $m(1, 1) = 1$ indicating one extra bit improves per step. In other words, a basic $\mathcal{D}(1, 1, 1)$ system on average performs a lookup in *half as many hops* as reported by previous work.

For a Discrete Symbol configuration, $\mathcal{D}(b, b, k)$, the number of bits improved on average is $b + m(b, k)$. For a Split Symbol configuration, $\mathcal{D}(b, 1, k)$, the number of bits improved on average is $b + m(1, k)$. *While the Split Symbol approach does use more routing table space for the same values of b and k, it has the advantage that it can leverage a random improvement of a single extra bit.* The Discrete Symbol approach must randomly improve by b extra bits at a time to make use of random improvements.

To compare the different approaches, we first consider the three extreme cases: $\mathcal{D}(1, 1, k)$ (pure Redundancy), $\mathcal{D}(b, b, 1)$ (pure Discrete Symbols), and $\mathcal{D}(b, 1, 1)$ (pure Split Symbols). Figure 9.6a presents the performance of each approach as a function of the normalized routing table size. Split Symbols and Redundancy have nearly identical performance, while Discrete Symbols performs slightly better. For the case of Split Symbols ($\mathcal{D}(b, 1, 1)$), the b-bit symbols guarantee an improvement of b bits in the worst case, plus an additional $m(1, 1) = 1$ bits on average, for a total of exactly $b + 1$ bits

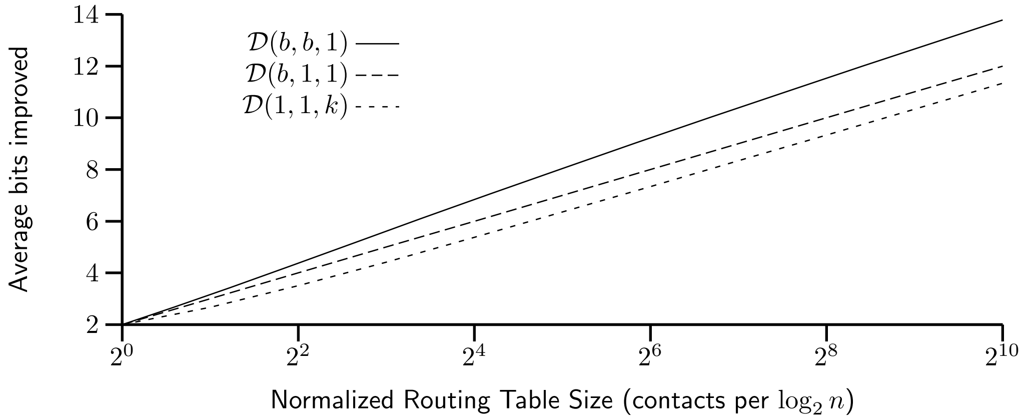
For the case of Discrete Symbols ($\mathcal{D}(b, b, 1)$), the b-bit symbols again guarantee an improvement of b bits in the worst case, plus an additional $m(r, 1)$ bits on average. However, $m(r, 1)$ asymptotically approaches 0 for large r. As a point of reference, for Pastry's typical value of $b = r = 4$, the average improvement is 4.27 bits per step, roughly a 4% reduction in the mean number of lookup hops[5] compared to that reported by the Pastry paper [26].

For the case of large buckets and 1-bit symbols ($\mathcal{D}(1, 1, k)$), there is one guaranteed bit of improvement, plus an additional $m(1, k)$ bits on average, for a total of $1 + m(1, k)$. As a point of reference, for the value of $k = 20$ suggested in the Kademlia paper [24], the average improvement is 5.7 bits per step rather than 1 bit per step, resulting in a 60% reduction in the mean number of hops!

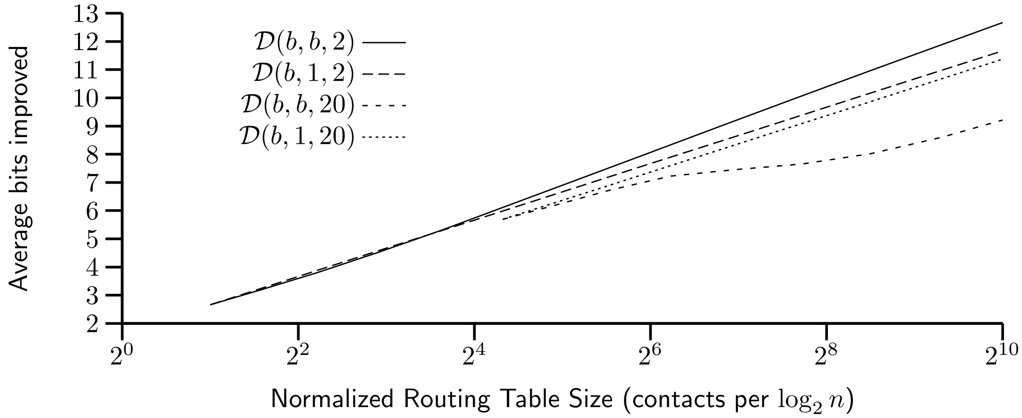
An important question is: *Can performance be improved by using a mixture of large buckets and large symbols?* The short answer is "No". Figures 9.6b and 9.6c plot several other permutations of $\mathcal{D}(b, r, k)$. Figure 9.6b holds k constant and varies b, while Figure 9.6c holds b constant and varies k. For small values of k (*e.g.*, 2) with varying b, both Discrete Symbols and Split Symbols have performance in between

---

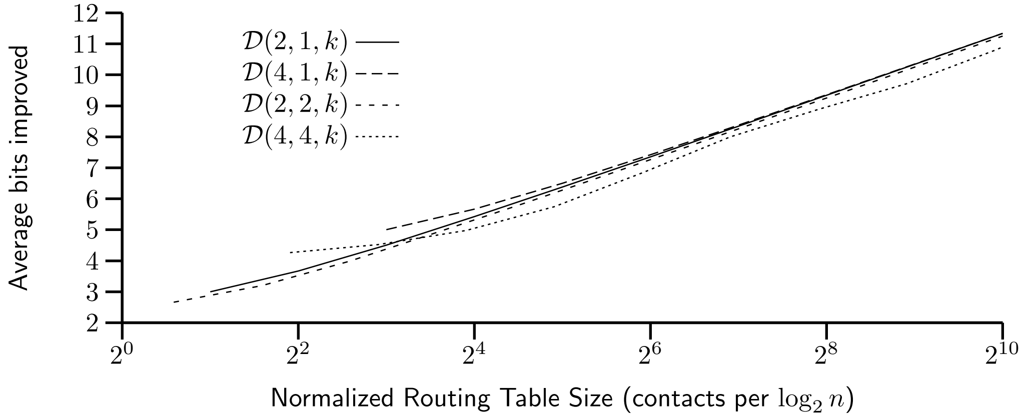
[5] The expected number of hops is equal to $\frac{\log_2 n}{\mathcal{B}}$ where $\mathcal{B}$ is the average number of bits of improvement.



(a) Comparison of pure Redundancy, pure Discrete Symbols, pure Split Symbols



(b) Comparison of approaches with constant k and varying b



(c) Comparison of approaches with varying k and constant b

FIGURE 9.6: Relative performance of different routing table structures

their regular performance and $\mathcal{D}(1,1,k)$. For moderate values (*e.g.*, 20) of k, the performance of Split Symbols is virtually identical to $\mathcal{D}(1,1,k)$, while the performance of Discrete Symbols plummets (as seen in Figure 9.6b). Because Discrete Symbols cannot make good use of randomness, the k-redundancy imposes a cost with little benefit on lookup performance.

In summary, increasing the symbol size (b) offers a constant-factor improvement to performance, while using k-buckets offers a comparable average-case improvement. Moreover, k-buckets offer other advantages as follows:

- Reduced implementation complexity
- Lower maintenance bandwidth; fewer restrictions on acceptable contacts allows for more contacts to be acquired passively
- Better resistance to churn by accumulating high-quality contacts

While our framework is motivated by our study of Kad, it applies to any prefix-matching DHT and could be extended to other DHTs that can accommodate different symbol or bucket sizes. In the following section, we use the formulas we have developed to compute a lower bound on the average lookup hops in Kad and empirically examine how close our predicted model is to the actual performance.

9.2.4 Accuracy of Routing Tables in Kad

In the previous section, we demonstrate that theoretical predictions of lookup performance cannot depend only on the network size (n) and symbol size (b), but must also incorporate the resolution (r) and bucket size (k). In this section, we return to comparing theory with practice, beginning with an exploration of the resolution (r) and bucket size (k) in Kad. We begin by determining these values by source code inspection[6] and then empirically study the impact of churn on routing table accuracy.

---

[6] There is no written specification that describes the Kad protocol so our explanations are based on our reading of the source code.

9.2.4.1 Predicting Kad Performance

Close examination of the eMule 0.46a source code reveals that Kad is based on Kademlia with a bucket size of 10 contacts ($k = 10$) and 3.25-bit Split Symbols, meaning Kad is a $\mathcal{D}(3.25, 1, 10)$ system. The $\frac{1}{4}$ bit is due to the fact that Kad uses unbalanced subtrees. Each interior node has branches with labels 0, 1000, 1001, 101, 110, and 111. The 0 branch leads to the next interior node; the other branches lead to k-buckets. The average improvement per step is $3.25 + m(1, k)$ bits. We also validated our understanding of the source code with empirical observations of eMule's operation. According to Formula 9.6 the mean number of improved bits per step is 6.98 in Kad. As a special case, Kad's root node has a full 16 branches, so it improves at least the 4 most significant bits and 7.73 bits on average on the first step. To account for this, we revise Formula 9.1 as follows:

$$\text{steps per lookup in Kad: } 1 + \frac{\log_2 n - t(4, 1, 10)}{t(3.25, 1, 10)} \tag{9.7}$$

Thus, the expected number of hops in Kad is $1 + \frac{\log_2(n) - 7.73}{6.98} = 2.74$. However, this assumes that every bucket has a full set of 10 valid entries. In the next subsection, we empirically examine the contents of buckets in Kad.

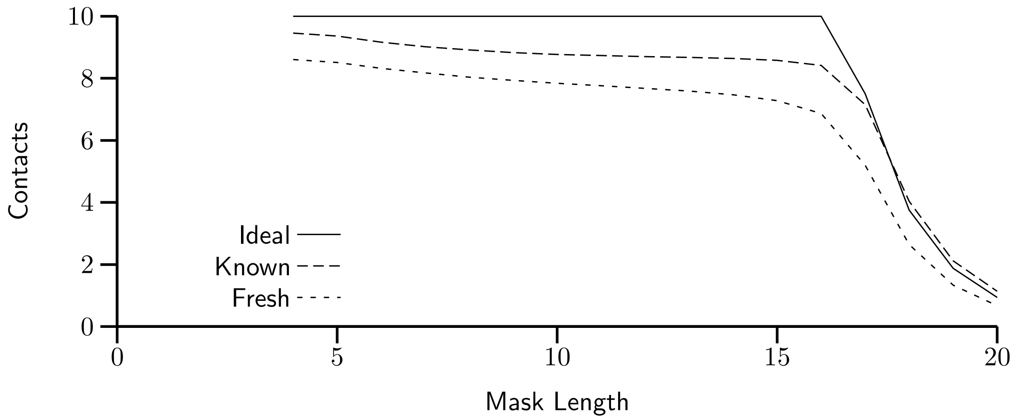
9.2.4.2 Characteristics of Kad Tables

Using kFetch, we retrieved the routing tables of approximately 80,000 randomly selected Kad peers in June 2005 and examined two properties of their k-buckets: *(i) completeness* describes how full a bucket is and *(ii) freshness* describes the number of contacts in the routing table that are still active (*i.e.*, do not point to departed peers). To locate a peer at random, kFetch generates a random Kad identifier, then performs a Kad lookup to locate the peer closest to that Kad identifier. kFetch identifies stale entries in the routing table by actively probing (*i.e.*, sending a lookup request to) each contact in the routing table, concurrently with continuing to download the routing table.

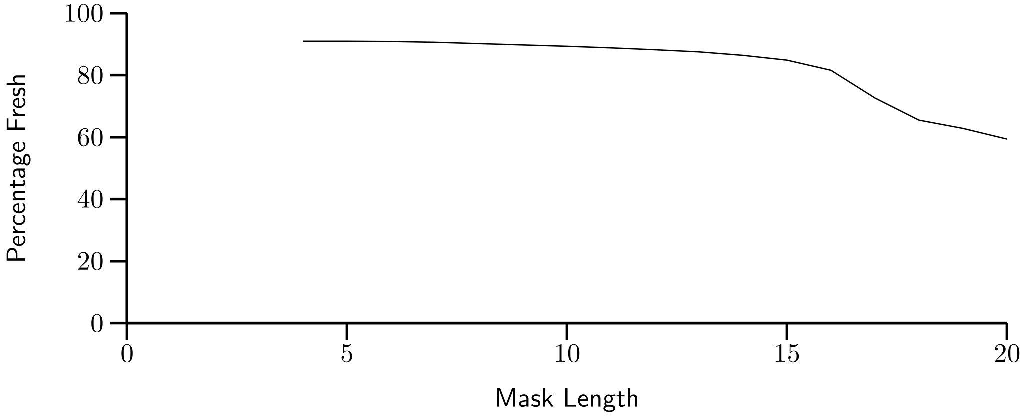
Figure 9.7a shows the mean number of contacts ("Known") in each routing table bucket as a function of the bucket's subnet mask. It also shows what fraction of these contacts are fresh (*i.e.*, the contact responded to our ping). The "Ideal" line indicates the average number of contacts we would expect to be in each bucket if the routing tables were perfectly up to date, *i.e.*, $\min\left(10, \frac{n}{2^x}\right)$ where x is the number of bits in the address mask and n is the population size. All three curves (Ideal, Known, and Fresh) decrease steeply as the mask length exceeds 16 bits, due to the limited number of matching contacts in the system. For shorter masks, on average each bucket has one or two empty slots and contains one stale contact. The mean number of empty slots is slightly higher as the mask length increases.

In Figures 9.7b and 9.7c, we examine the number of fresh contacts in each bucket normalized by the total number in each bucket and by the expected (ideal) number, respectively. Figure 9.7b shows the mean number of fresh contacts as a fraction of the number of contacts actually present. This shows that around 90% of entries are fresh for masks up to length around 16, then the fraction of fresh entries decreases, *i.e.*, the number of stale entries increases. This undesirable increase in stale entries is because the current implementation of eMule doesn't ping peers in buckets which are not at least 70% full. In fact, in Figure 9.7c, where we examine the number of contacts relative to the ideal number, above /17 there are actually more stale contacts than active peers anywhere in that subnet, causing the normalized value to exceed 100%! Peers gradually accumulate stale contacts in buckets which are expunged too slowly. As a consequence, virtually every lookup in Kad necessarily ends with timeouts to stale peers even though the closest active peer has already been contacted! This is a direct result of eMule's policy of not expiring contacts in mostly-empty buckets. As this routing table maintenance problem can trivially be corrected in the eMule code, throughout this paper we emulate the correct behavior as follows. After a lookup completes, we compute the latency as the time from the start of the lookup until kLookup receives a packet from the closest *responsive* peer.

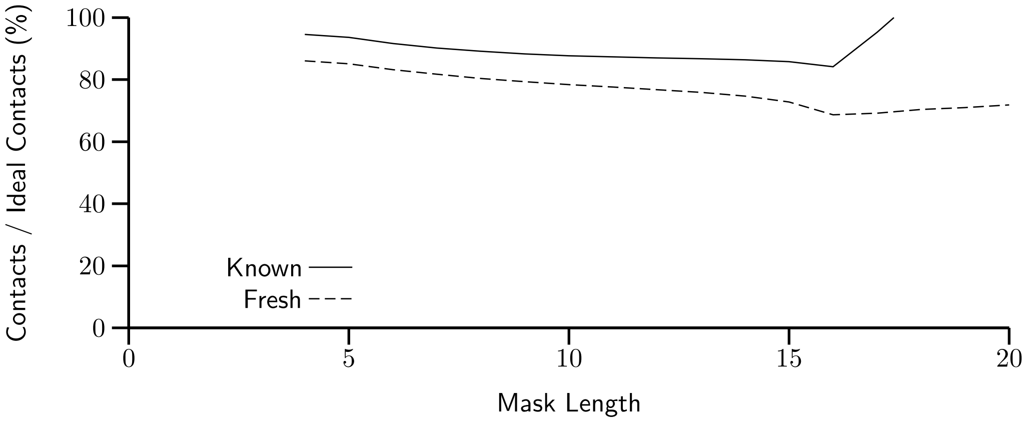
From Figure 9.7a, we see that on average there are 1.5 empty slots plus 1 stale contact per bucket. We could plug $k = 10 - 1.5 - 1 = 7.5$ into our formula, but first we must validate that most buckets are close to the average state. If the variance is very high (*e.g.*, if 85% of buckets had 10 entries and the other 15% were completely empty), then using the average would introduce considerable error. Towards this



(a) Mean number of contacts in a routing table bucket



(b) Fraction of contacts in a bucket that are fresh (i.e., valid)



(c) Mean contacts per bucket, normalized by the ideal value

FIGURE 9.7: Properties of routing table buckets in Kad, as a function of how specific the bucket's mask is

end, Figures 9.8a and 9.8b present the CDF of the number of contacts and fresh contacts across all observed buckets for masks /4, /8, and /12. They show that for both completeness and freshness, nearly all buckets are close to the average value. Therefore, we may use the average value for the purposes of our computations without introducing considerable error.

Using an average of 1.5 empty slots plus 1 stale contact per bucket, we have an effective bucket size of $k = 7.5$. This increases the expected hop count slightly from 2.7 to $\frac{\log_2(n)-7.33}{6.58} + 1 = 2.91$ hops, according to Formula 9.7. This is still significantly better than the previously predicted value of 6.30 hops. Comparing with observed performance in practice (3.2 hops), 2.91 hops is consistent with our expectation that theory provides a lower bound for the performance in practice. Additionally, we see that the performance in practice is only slightly worse than the performance in theory.

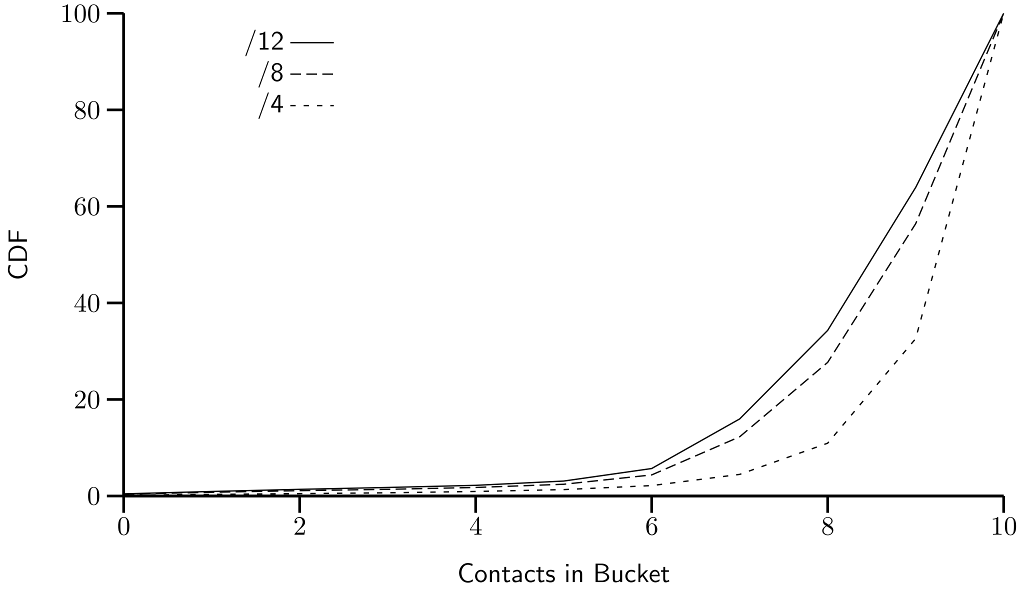
Note that we are unable to change the routing tables in the entire Kad network. Therefore, we explore only client-based alternatives to improve lookup performance in Kad and evaluate different techniques to improve the efficiency and consistency of lookup in the following two sections.

9.2.5 Improving Lookup Efficiency

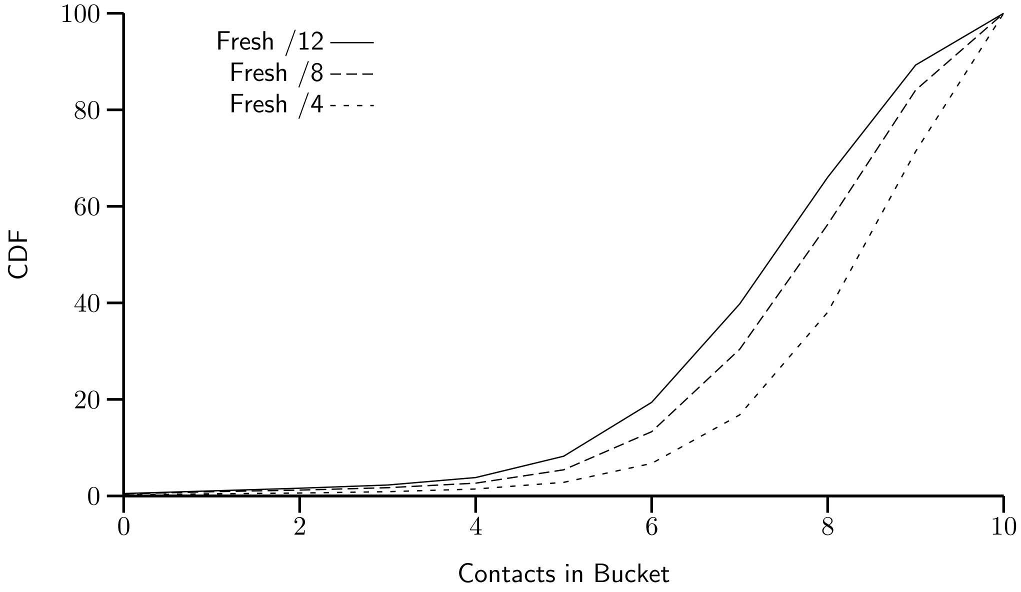
We turn our attention to client-based approaches to improve the performance of iterative lookup over a DHT that has inaccurate routing tables. While incomplete buckets will degrade performance as described in the previous section, stale contacts can dramatically increase latency by causing timeouts to occur. Since the timeout interval is typically set to at least a few round-trip times, it can easily exceed the desired time for the entire lookup.

9.2.5.1 Parallel Lookup

To improve performance despite inaccurate routing tables, clients can perform parallel lookup. While parallel lookup has traditionally been used exclusively with iterative DHTs, Jinyang Li *et al.* [69] also present a technique for performing parallel lookup on a recursive DHT.



(a) Distributions of Bucket Completeness



(b) Distributions of Bucket Freshness

FIGURE 9.8: Distributions of completeness and freshness in Kad's buckets

In a parallel lookup, a client simultaneously manages multiple lookup requests to different peers and performs the lookup process based on the information obtained from all requests, reducing the problem of hitting stale contacts, and improving lookup performance at the cost of greater network overhead (*i.e.*, a larger number of requests per lookup). Parallel lookup has two other significant advantages. First, lookup requests facilitate populating or passively updating the routing tables, which in turn reduces the bandwidth requirement for explicit updates, as shown in [29]. Second, during each step of the lookup process, parallelism increases the number of contacts searched, increasing the probability of finding a contact closer to the target (*i.e.*, with more matching bits) and thus decreasing the number of hops needed to reach the target. We examine the following two classes of parallel lookup techniques:

1. *Strict Parallel Lookup*: In this approach, a client begins a lookup by sending lookup requests to the α best known contacts. Similar to the window-based congestion control in TCP, a client restricts the number of requests in-flight to α. A new request is issued only when a pending request times out or a response is received. The resulting overhead is limited to a factor of α. The downside of the strict approach is that when a client sends a packet to a departed contact, it must wait for a timeout to occur before giving up. In the meantime, the degree of parallelism is effectively reduced by one. However, a timeout is typically set to at least a few round-trip times which is on the order of the desired time for the entire lookup. Thus, in the strict approach, α roughly determines the number of timeout events a client can experience without incurring a significant latency penalty. Kademlia uses this approach.

2. *Loose Parallel Lookup*: Parallel lookup can be performed in a looser fashion by allowing more than α requests in flight. In this approach, a client can issue a lookup request to a contact that is among the top α contacts as soon as such a contact is identified, even if this lookup request increases the number of pending requests beyond α. For example, if $\alpha = 3$, the lookup begins by sending 3 lookup requests. If the first response contains 3 better contacts (which is likely), 3 more requests are sent immediately. While this approach appears to be significantly

more expensive than strict parallel lookup, it incurs only modest additional overhead since later responses from the same step are less likely to contain better contacts (*i.e.*, each time a packet is sent, the bar has been raised). The advantage of this looser approach is the ability to quickly abandon lookups that are likely to time out. This approach is used by eMule.

9.2.5.2 Evaluating Parallel Lookup

We evaluated the performance of both types of parallel lookup techniques under varying degrees of parallelism. Using kLookup, we captured several hundred lookups for different values of α for both strict and loose parallelism. Each lookup used a unique, randomly-selected source and a unique, randomly-selected destination. In our evaluation, we examine three metrics:

- **Hops**: The number of hops from the source to the destination
- **Latency**: The time from the start of the lookup to when a response is received by the final destination, which is a function of the number of hops and the time spent waiting for responses and timeouts
- **Messages Sent**: The overhead used to perform the lookup

As we mentioned earlier, increasing α can reduce the number of lookup hops by providing more opportunities to randomly improve extra bits. Figure 9.9 shows

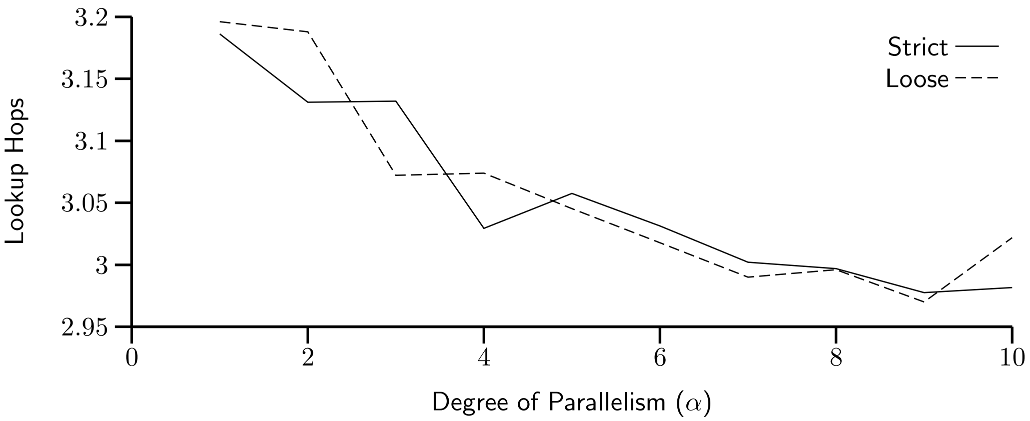


FIGURE 9.9: Effect of parallel lookup on hop count

that the mean number of hops decreases slightly as α increases,[7] providing empirical support.

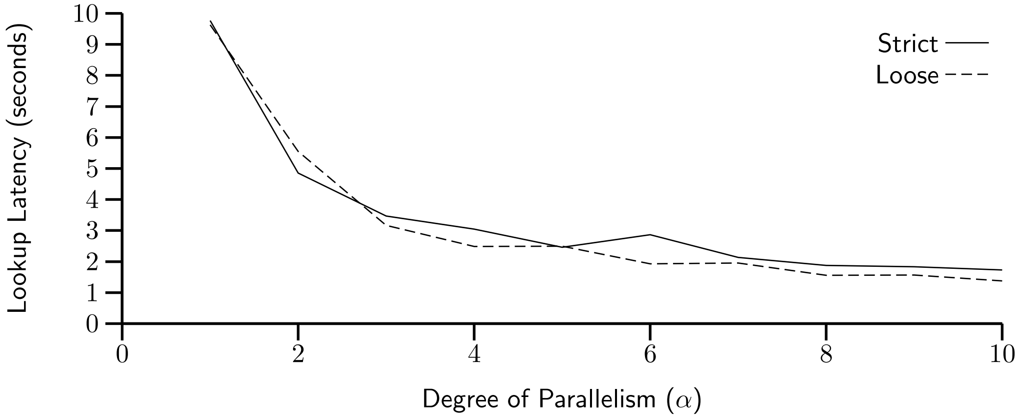
The key question is: *how much latency is introduced to lookup by timing out due to stale contacts?* Figure 9.10a compares the latency of the two approaches for several values of α. The first observation is that the latency for $\alpha = 1$ is very high—close to 10 seconds. Using a value of $\alpha = 3$ dramatically reduces the latency, with diminishing returns for larger α. Second, Figure 9.10a reveals that the loose approach is just barely quicker than the strict approach for constant α. The greatest advantage of loose parallelism is that it is significantly less likely to get stuck waiting for timeouts to occur. However, as we show in Section 9.2.3, few contacts in Kad are stale. This explains why loose parallelism does not show much performance improvement for this network.

To examine the communication overhead of parallel lookup, Figure 9.10b shows the number of packets sent as a function of α for the two approaches. In both cases, the overhead increases roughly linearly with α, with the loose approach generating roughly twice as many messages as the strict approach. Given that for fixed α the performance of strict and loose parallelism are quite similar, strict parallelism is the better choice for the current Kad network. To directly compare the two, Figure 9.10c factors out α by plotting the lookup hops as a function of the overhead. This figure shows that asymptotically the performance of strict and loose parallelism are surprisingly similar. A large number of messages represents the lower bound on lookup hops: no amount of increased parallelism of any kind will significantly improve performance. At the lowest overhead (in messages sent), the two perform the same since the two approaches result in identical behavior for the special case $\alpha = 1$. However, the sweet-spot for strict parallelism ($\alpha = 3$) is significantly better than the sweet-spot for loose parallelism.

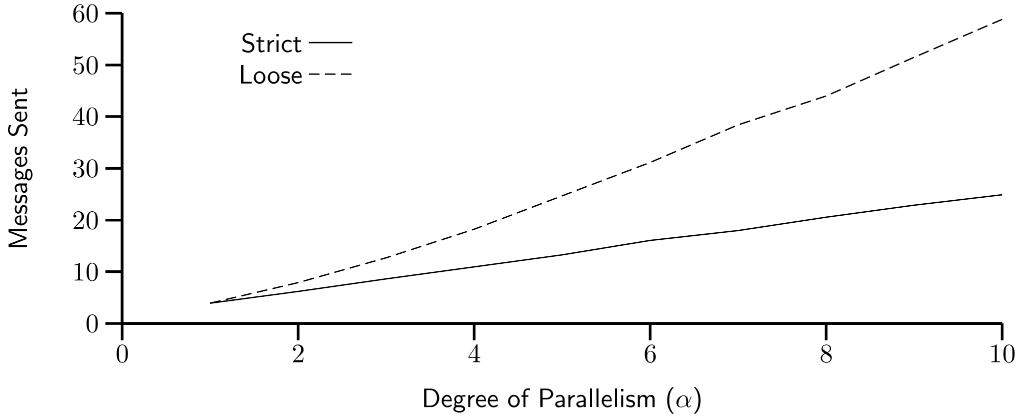
In summary, these observations show that strict parallelism with $\alpha = 3$ is a good choice for the current Kad network. Higher values of α and loose parallelism substantially increase overhead without much change in performance.

---

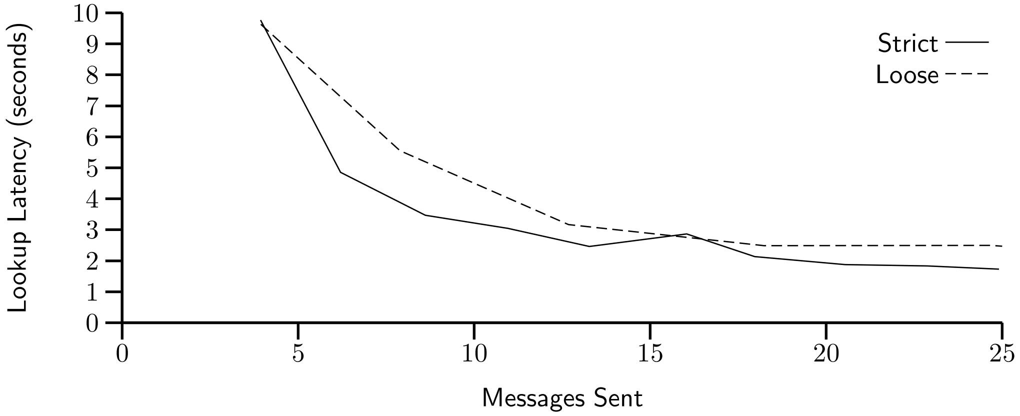
[7] This figure is noisy due to the narrow y-axis range. The general downward trend is nevertheless visible.



(a) Mean lookup latency as a function of α



(b) Mean packets sent as a function of α



(c) Lookup latency as a function of packets sent

FIGURE 9.10: Strict versus loose parallel lookup

Comparing with eMule: As part of creating kLookup, we also attempted to exactly reimplement eMule 0.46a's lookup algorithm. We validated this mode of kLookup by extending tcpdump to decode Kad packets and performing lookups for the same key using kLookup and eMule itself to verify their similarity. In the process of implementing eMule's lookup algorithm, we discovered a few bugs [120–122] which significantly degrade its efficiency.[8]

As part of our study, we wanted to compare the performance of eMule's current lookup algorithm with and without the bugs, in the hope that it will be of use to the eMule developers. Again, we examine performance in terms of hops and latency, and overhead in terms of the number of messages. These experiments are based on more than one-thousand experiments using kLookup from unique, randomly-selected sources and destinations. With the bugs fixed, eMule's lookup algorithm is $\alpha = 3$ with loose parallelism.

Figure 9.11a presents a CDF of the number of hops to perform a lookup. With the bugs present, the mean value is 3.59 hops per lookup. Without bugs, the cost of hops drops to 3.08 hops per lookup. Figure 9.11b shows the latency of the two versions. In both cases, there is a significant tail (not shown) out to around 70 seconds. We see that the fixed version improves by around 1 second in most cases. The most striking difference however is in the overhead, as shown in 9.11c. The fixed version uses roughly half as many messages on average.

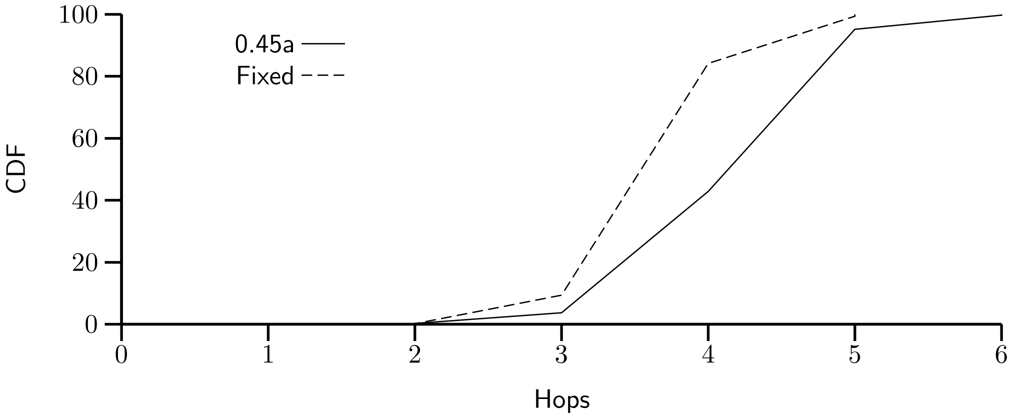
9.2.6 Improving Lookup Consistency

Ideally, each peer in a DHT is responsible for a certain part of the DHT identifier space and lookups for any identifier should lead to the responsible peer. In practice, peer churn causes two types of inaccuracies in routing tables:

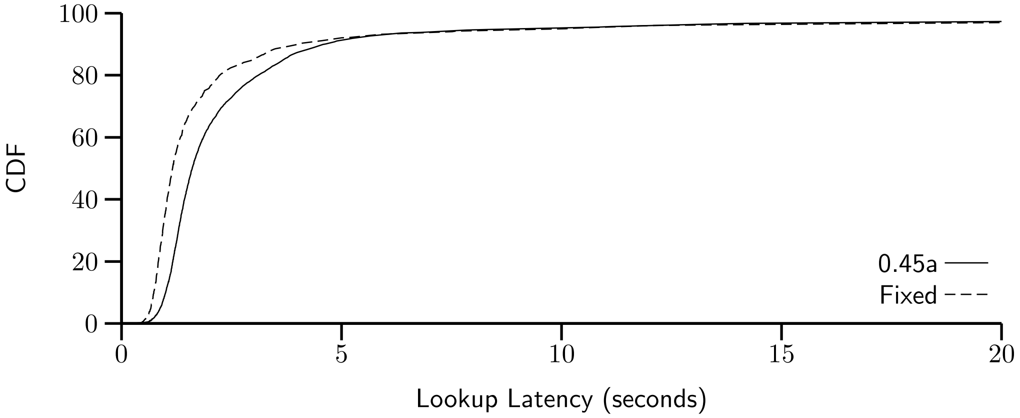
1. Peers may not yet have pointers to a recently arrived peer.
2. Peers may have stale pointers to a recently departed peer.

---

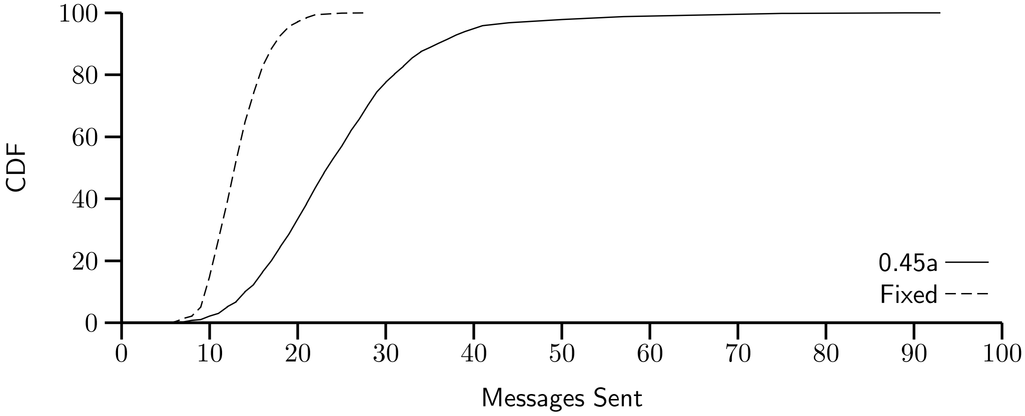
[8] Our results are based on eMule version 0.46a, the most recent version available at the time of our study. We have been corresponding with the eMule developer team regarding these discoveries, and at least some of the reported bugs were corrected in 0.46b.



(a) Hops



(b) Latency



(c) Overhead

FIGURE 9.11: Performance improvement gained by fixing the bugs in eMule 0.45a

When routing tables are incorrect, it is possible for some parts of the identifier space to be unreachable for some peers. The extent of these problems is determined by how frequently the DHT validates its pointers, known as route stabilization [82], compared to the rate of churn in the system. One approach to minimize these problems is to increase the frequency of route stabilization. However, this significantly increases the bandwidth required for route maintenance.

9.2.6.1 Content Replication

An alternative approach is to map each identifier to the set of the c closest peers in the identifier space, rather than to only the single closest peer. The publishing operation performs a regular lookup, then searches the surrounding area to find the closest c peers. The search operation does the same, and as long as the two find any peer in common the search will succeed. Kademlia [24] takes this approach as a basic principle; however, it can be used in almost any DHT. For example, DHash [31] implements this technique over Chord. The parameter c must be chosen based on knowledge of the degree of routing table inaccuracy, to guarantee with high likelihood that multiple lookups will be able to find peers in common.

The key question is: *what is the right value of c to guarantee a certain level of reliability p?* In the following subsection, we use empirical techniques to answer this question for Kad.

9.2.6.2 Evaluating Lookup Consistency

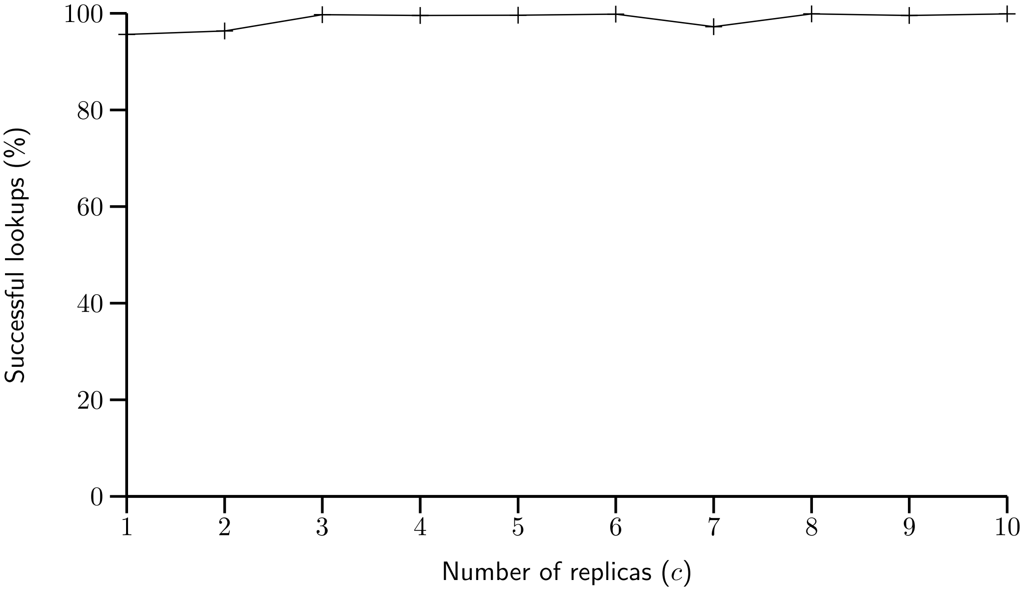
To explore lookup consistency, we extended kLookup to locate the c closest points after its regular search has completed. To get an empirical measure for p, we use kLookup to perform 50 lookups to the same key, each from a different and random starting point in the Kad network. The first lookup emulates a publish operation which returns a set of peers to publish on. The following lookups to the same key emulate query operations, returning a set of peers in response to the actual query. Computing the fraction of queries that successfully find one of the target peers yields an empirical measure of the consistency, p, for that experiment. We perform the lookups as concurrently as possible to limit the effects of peer departure and arrival. For these experiments, we used strict parallelism with $\alpha = 3$. We conducted this experiment 20 times for each value of c in the interval $[1, 10]$ (*i.e.*, 1000 lookups per value of c: 20 experiments and 50 lookups per experiment).

We observed that for $c = 1$, the consistency is only 89%, meaning that 11% of the time queries fail to find the same "closest" peer as a publisher. To explore how many replicas are needed, Figure 9.12a plots reliability (p) as a function of the number of copies (c). With three copies ($c = 3$), the reliability is over 99.9% across the twenty 50-lookup trials. For $c = 2$, the consistency is in between, at around 96%.

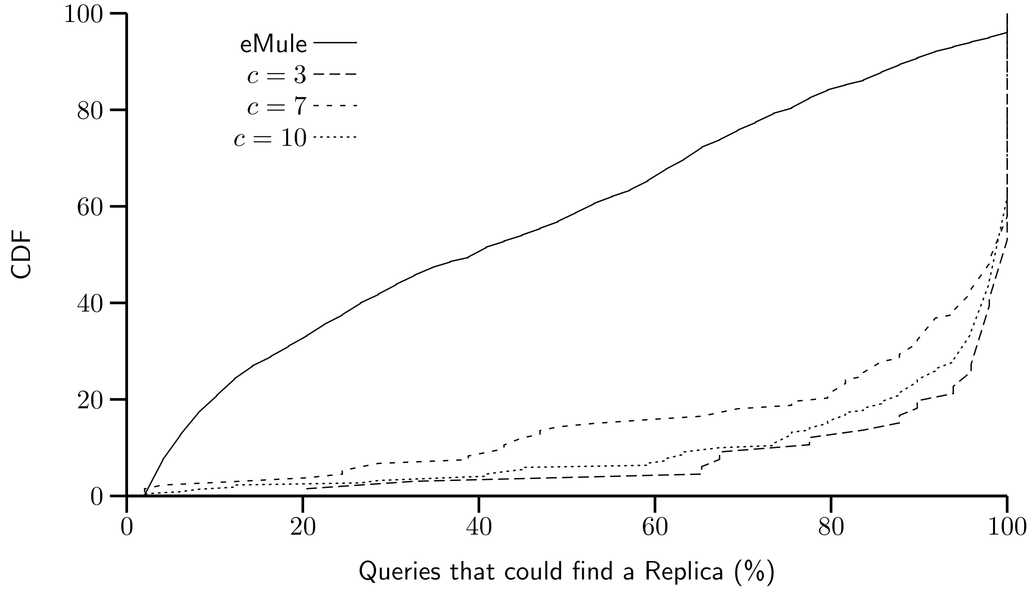
The above values are for finding *any* of the replicas. However, another issue regarding consistency is how effectively *all* of the replicas can be found. If one replica can always be located, but the others cannot be, then lookups will fail if the one easy-to-locate replica becomes unavailable. Therefore, for each replica we compute the *number* of lookups that found the replica, and plot the results as a CDF in Figure 9.12b. An ideal curve would be a vertical line at $x = 100\%$, indicating that every query found every replica. The Figure shows that the performance for the nearby-replication method is indeed good, with roughly 50% of queries able to find every replica, and 80–90% of queries able to find 80% of the replicas.

In summary, our results show that locating the three closest nodes after finding the closest peer is an effective way to cope with routing table inconsistencies. More importantly, we show that even routing table inconsistencies can be a considerable problem in practice with more than 11% of lookups failing when no replication is used.

Comparing with eMule: Currently, eMule uses a fuzzy replication algorithm which selects several peers as part of the endpoint set that are not necessarily the closest. In addition to our experiments for different values of c, we also conducted more than 60 experiments using eMule's algorithms for publishing and lookup. We found that eMule's approach produces 19 copies on average and queries succeed 99.9% of the time. While robust, this is 6.3 times more replicas than simply using $c = 3$. Figure 9.12b shows the CDF of the percentage of all replicas each lookup found. The



(a) Lookup consistency as a function of c



(b) Lookup consistency CDF

FIGURE 9.12: Lookup consistency

performance is substantially worse than the nearest-c approach, with many replicas being found by only a few queries. For example, 50% of replicas could be found less than one-third of the time, compared to just 3% for $c = 3$. Additionally, some replicas were not found by *any* queries.

9.2.7 Related Work

Early work on DHTs focused on introducing new DHTs [23, 24, 26, 82] that each achieved $O(\log n)$ lookup hops using $O(\log n)$ state per peer. Initially, it was difficult to directly compare the performance of these DHTs, as each DHT has several tunable parameters, which might cause them to perform better or worse under different loads. For example, under low churn a DHT with a large routing table will perform better since it can achieve faster lookups and route maintenance is inexpensive. The same DHT will perform poorly under heavy churn.

Several studies [5, 28, 29, 104, 127, 128] have attempted to address the issue of DHT performance under churn, in most cases using a simple Poisson model for session length. However, several measurement studies of peer-to-peer systems [45, 48, 59, 64, 65] show that session times are dramatically different from Poisson. In this study, we conduct experiments using the real Kad network, *i.e.*, under real churn.

Gummadi *et al.* [129] showed that DHTs can be broken into two components: geometry (or structure) and lookup strategy. Some DHT geometries provide greater routing flexibility than others in terms of neighbor selection or route selection. For example, in CAN a peer's neighbors are precisely defined by the geometry, while in Chord there are 2^{i-1} options for the i^{th} neighbor, providing Chord substantially more flexibility in selecting neighbors. Their results show that more flexible systems, such as Chord and Kademlia, can achieve better performance. We utilize their division between geometry and lookup to study the lookup behavior in light of the geometry of the deployed Kad network.

Jinyang Li *et al.* [29] developed a performance-versus-cost framework (PVC) for comparing different DHTs. Their key observation is that for a given bandwidth usage, there is a minimum lookup latency that can be achieved over the entire space of DHT parameters, and vice versa. In PVC, they simulate each DHT using a wide variety of parameters and plot the best lookup latency each DHT can achieve within a given bandwidth constraint. This allows them to compare how different DHTs make the performance-versus-cost trade-off under a given load. They show that using large routing tables with infrequent stabilizations and parallel lookup achieves a better balance than other approaches, culminating in their later development of the Accordion DHT [69]. However, PVC can only draw conclusions about how well the DHTs respond to the simulated workload. While their work is useful for drawing inferences about design trade-offs, our work is aimed at optimizing tunable parameters in a DHT that is already deployed.

In summary, prior work on DHTs has been driven by analysis, simulation, and limited experiments. In each case, a model is used to approximate or estimate real-world behavior. This paper presents experiments on a deployed DHT that has approximately one million real users and develops tools and techniques for improving lookup performance.

9.2.8 Summary and Future Work

This paper examines lookup performance over the Kad DHT network. We analytically derive new formulas for the expected hop count, taking into account random improvements, and demonstrate that Kademlia's use of k-buckets leads to significantly better performance than previously reported. We present new tools, kFetch and kLookup, to characterize the accuracy of routing tables in Kad, examine the impact of routing table accuracy on efficiency and consistency of the lookup operation, and experimentally verify our analysis. Furthermore, we explore two types of parallel lookup techniques and their impact on lookup efficiency and also examine the degree of replication needed to cope with routing inconsistency. While some of our empirical results are specific to Kad, our analysis applies to other prefix-matching DHTs such as Pastry and Tapestry and could be modified to handle other DHT geometries.

In our future work, we plan to measure the bandwidth eMule uses for route maintenance and study ways to maintain higher quality routing information at lower cost.

We also plan to use our recent measurement-based characterization of churn in peer-to-peer systems (Chapter 8) to determine the number of replicas needed to guarantee the availability of a piece of data within the network. This will include a mathematical analysis of the trade-off between republishing the data more frequently to a few peers versus publishing infrequently to many peers, followed by empirical experiments to validate our findings.

CHAPTER 10

Summary of Contributions

My dissertation lays an important foundation for understanding peer-to-peer systems. The main contributions fall into the following categories: *(i)* tools and techniques for collecting *accurate* measurements of properties of peer-to-peer system, *(ii)* empirical results which may be used to evaluate models and improve our understanding of existing systems, and *(iii)* useful models based on the empirical observations. Two basic types of measurement techniques are presented and carefully evaluated for accuracy: a technique for capturing global snapshots (in Chapter 4 and a technique for gathering unbiased samples (in Chapter 5). These techniques were developed into tools, called Cruiser and `ion-sampler`, respectively, which provide the majority of the data examined in later chapters of this dissertation.

Using the empirical data gathered using these new techniques, this dissertation characterizes the properties of peer-to-peer systems along two axes, forming four groups: static peer properties, dynamic peer properties, static connectivity properties, and dynamic connectivity properties. Overall, this dissertation provides a tremendous amount of information about properties of existing peer-to-peer systems, including raw measurements, insights into underlying causes, and models that can be used for simulation and analysis.

Each of the chapters in this dissertation includes many additional details, useful for understanding and simulating peer-to-peer networks. Finally, the measurement techniques developed for this dissertation will be useful for future measurement studies of other properties of peer-to-peer systems. We conclude this dissertation with Table 10.1, which summarizes the characterization contributions in each of the four types of properties.

Property	Known Before	Contribution
Static Peer Properties		
Geographic distribution	—	Gnutella is North America–centric, with a strong European minority, regardless of time of day.
Free riders	Conflicting (25%–68%)	More accurate measurements show around 13%.
File sizes	Almost all files are small, but large files make up a large fraction of all bytes	Confirmed
File types	Mostly audio files, but video files make up a large fraction of all bytes	Confirmed. Video files have made further gains in popularity since the previous study.
File popularity	Most files are unpopular, but some files are extremely popular	Confirmed. Additionally, the distribution changed very little over an 11-month measurement period.

TABLE 10.1: Summary of characterization contributions. Fields marked with a “—” are previously unstudied to the best of our knowledge.

Property	Known Before	Contribution
Dynamic Peer Properties		
Inter-arrival times	—, assumed exponential	Weibull provides a better fit. May be due to composition of many exponential distributions.
Session lengths	Conflicting, often described as heavy-tailed or Pareto	BitTorrent data is not consistent with Pareto. Weibull provides a better fit. Gnutella and Kad data were inconclusive.
Lingering after completion	—	Some peers in BitTorrent remain hours, days, or weeks after completion, leading to the high percentage of seeds.
Peer uptime	—	Most sessions are short, but most active peers have a long uptime.
Remaining uptime	—	Most sessions are short, but most active peers have a long remaining uptime.
Uptime predictability	—	Remaining uptime is correlated with uptime so far, but with high variance.
Session length predictability	—	Consecutive sessions are strongly correlated in Gnutella and Kad, but not in BitTorrent.
Correlations in availability	—	Availability over consecutive days is strongly correlated in Gnutella and Kad, but not in BitTorrent.

TABLE 10.1: (cont.)

Property	Known Before	Contribution
Static Connectivity Properties		
Top-level degree distribution	Power-law	Exhibits a strong mode near 25–30, and not power-law. Power-law may be a result of measurement artifacts. Degree is correlated with uptime.
Leafs per ultrapeer	—	Modes near 30 and 45 for LimeWire and BearShare, respectively.
Ultrapeers per leaf	—	Almost all in the range 1–3.
Shortest path lengths	Nearly all shortest paths are short (5 or less)	Confirmed, 99.5% have length 5 or less.
Eccentricity	—	Eccentricity is in the range 6–12.
Small world	Gnutella has a relatively high clustering coefficient.	Confirmed.
Resilience	Gnutella is resilient to random peer removals, but not highest-degree-first removals.	More accurate measurements show Gnutella is resilient to both types of removals.

TABLE 10.1: (cont.)

Property	Known Before	Contribution
Dynamic Connectivity Properties		
Stable core	—	As a result of peer churn, an emergent effect is an “onion-like” bias where peers tend to be neighbors with other peers with similar uptime.
DHT stale contacts	—	In Kad, each 10-bucket on average has 1.5 empty slots and 1 stale contact.
DHT query performance	Limited experiments on toy systems	First study on a widely deployed system. Query performance is better than predicted by earlier theoretical work. We revise the theory to properly calculate average performance.
DHT parallel lookup	Limited experiments on toy systems	For Kad, strict parallelism with three parallel lookups is best.
DHT redundancy to ensure lookup consistency	Limited experiments on toy systems	For Kad, storing three copies near the target provides excellent consistency.

TABLE 10.1: (cont.)

INDEX

adjacent
  definition, 15
Anderson–Darling, 121, 123
availability, 136–138
  definition, 23

Barabási–Albert, 54–56
BearShare, 146, 151
BitTorrent, 66, 103, 108–110, 116, 118–131, 133–141, 206
  history, 8, 9
  related work, 18, 20, 105
blocks
  definition, 8
bootstrapping
  definition, 27
bucket
  definition, 179
bucket size, 186, 187, 190
  definition, 182

CCDF, *see* complementary cumulative distribution function
CDF, *see* cumulative distribution function
centralize, 20
clustering coefficient, 15, 16, 158, 160, 166, 167, 207
  definition, 15
complementary cumulative distribution function
  definition, 9
completion time, 126
  definition, 126
connection
  definition, 14
contacts
  definition, 179
crawl, 19–20
crawler, 19, 31, 34, 35, 37–40, 42, 59, 83, 84, 86, 87, 89, 100, 106, 107, 109, 134, 148, 177
  definition, 31
  related work, 26
Cruiser, 31–43, 54, 71, 72, 75, 79, 82, 86, 89, 106–108, 112, 117, 148, 159, 177, 204
  use with Kad, 177
cumulative distribution function
  definition, 9

D
  definition, 61
degree, 3, 8, 16, 33, 39, 48, 51–54, 56, 57, 59, 65–69, 72–74, 76, 85, 95, 111, 144, 147, 148, 150–153, 155, 159, 165, 171, 172, 192, 198, 202
  definition, 14
degree distribution, 16, 26, 27, 47, 68, 71–73, 146–153, 160, 165, 207
DHCP, 117
DHT, *see* distributed hash table
Discrete Symbols, 180, 182, 184, 186
  definition, 180
distance, 155
distorted, 3, 27, 39, 71, 148
  definition, 31
Distributed Hash Table
  history, 8
distributed hash table, 2, 3, 27, 30, 46, 77, 78, 103, 106, 107, 112, 136, 138, 143, 161, 170, 171, 174–176, 179, 180, 186, 190, 196, 198, 201, 202, 208
  history, 8, 9
distributions, 11–13

  - exponential, *see* exponential distribution
  - log-normal, *see* log-normal distribution
  - memoryless, *see* exponential distribution
  - Pareto, *see* Pareto distribution
  - power-law, *see* Pareto distribution
  - shifted Pareto, *see* shifted Pareto distribution
  - subexponential, *see* subexponential distribution
  - Weibull, *see* Weibull distribution
  - Zipf, *see* Zipf distribution
- downtime, 134–136
  - definition, 23
- dynamic connectivity properties, 4, 161–203
  - related work, 27–29
- dynamic peer properties, 4, 102–142
  - related work, 23–26, 104–106
- Dynamic Querying, 8, 153, 170
  - definition, 7–8

E
  - definition, 14
- edge
  - definition, 14
- edge distortion, 40, 42
  - definition, 39
- eDonkey, *see* eDonkey 2000
- eDonkey 2000, 82, 90, 91, 95, 107
  - history, 8, 9
  - related work, 18, 83
- eMule, 107, 171, 176–178, 187, 188, 193, 196, 197, 199, 202
  - history, 9
- Erdös–Rényi, 15, 16, 54, 55
- exponential distribution, 11, 63, 113, 123, 126
  - definition, 11

- FastTrack, *see* Kazaa
- FIFO, 60, 66
- file descriptors, 37–38
- file popularity, 95–98
- file properties, 81–101
  - related work, 83–84
- file snapshot
  - definition, 84
- file transfers
  - related work, 25–26
- file types, 98–100
- firewall, *see* NAT
- fitting, 10, 11, 25, 26, 95, 103, 105, 120, 121, 123, 125, 206
  - definition, 13
- flooding, 7, 29, 153, 170
- flow control, 28
- free riders, 90–92
  - definition, 90

- GDF, *see* Gnutella Developer Forum
- geographic location, 79–81
- Gnucleus
  - bootstrapping, 27
- Gnutella, 32, 42, 43, 46, 54–56, 59, 66, 71, 75, 79, 81, 82, 84–86, 89–92, 98, 100, 101, 103, 106, 107, 109, 110, 113–115, 117–120, 124–126, 128–137, 139–141, 143, 144, 146, 147, 149, 152–155, 157–160, 162, 170, 205–207
  - history, 6–8
  - related work, 18, 22, 24, 26–28, 84, 105
- Gnutella Developer Forum, 7
- goodness-of-fit tests, 13, 14, 53, 121, 123
  - definition, 13
- graph theory, 14–16
- Gtk-Gnutella
  - bootstrapping, 27
  - flow control, 28
- GWebCache, 27, 66

heavy-tailed, 12, 105, 118, 125, 126, 206
  definition, 11
History, 66
hop, 9, 27, 59, 65, 72, 73, 75, 155, 157, 170, 172, 173, 176, 177, 179, 180, 183, 184, 186, 187, 190, 192, 193, 196, 201, 202
  definition, 14

IID
  definition, 13
inter-arrival interval, 28, 105, 106, 118, 120–123
  definition, 23
interocular trauma test
  definition, 14
intrusion detection, 38–39
ion-sampler, 3, 45, 46, 69, 71, 75, 78, 82, 86, 93, 95, 100, 204
IOTT, *see* interocular trauma test
iterative routing, 175, 178

Kad, 78, 106–108, 110, 112, 113, 115–122, 124, 125, 128–141, 171–173, 175–178, 182, 186–188, 190, 194, 196, 198, 201, 202, 206
  definition, 171
  history, 9
  related work, 18
Kademlia, 171, 173–177, 179, 180, 182, 187, 192
  history, 9
Kazaa, 107
  history, 8
  related work, 18, 24, 83, 84
kFetch, 178, 187, 202
  definition, 178
kLookup, 172, 178, 188, 193, 196, 198
  definition, 178
Kolmogorov–Smirnov, 61, 73
KS, *see* Kolmogorov-Smirnov

latency, 193
Law of Large Numbers, 3
leaf peers, 7, 27, 33, 40, 41, 87–89, 91, 92, 107, 145–147, 151, 152, 155, 162, 182, 207
  definition, 7
lifetime
  definition, 23
LimeWire, 1, 28, 146, 151, 207
  bootstrapping, 27
  flow control, 28
lingering time, 126
  definition, 126
log-normal distribution, 125
  definition, 11
lookup consistency, 196–201
lookup efficiency, 190–196
lookup reply
  definition, 175
lookup request
  definition, 175
lookup
  definition, 173

Maximum Likelihood Estimator, 121
measurement techniques, 17–20
Metropolis–Hastings method
  definition, 52
MLE, *see* Maximum Likelihood Estimator
modeling, 9–11
MRW, *see* Metropolized Random Walk
MRWB, *see* Metropolized Random Walk with Backtracking
Mutella
  bootstrapping, 27
  flow control, 28

Napster, 1
  history, 6
  related work, 18, 22, 24, 83, 105
NAT, 38, 87, 116, 117
neighbors

  definition, 15
Network Address Translation, *see* NAT
node
  definition, 14
node distortion, 40–42
  definition, 39
Nullsoft
  history, 6

Oracle, 66
overlay network
  definition, 2
Overnet, 25, 105
  history, 9
  related work, 18, 24

parallel lookup, 172, 190–196, 202
  loose
    definition, 192
  strict
    definition, 192
Pareto distribution, 12, 25, 27, 63, 92, 144, 147
  definition, 12
  shifted, *see* shifted Pareto distribution
participate, 19
passive monitoring, 17–19, 21, 104
peer
  definition, 14
peer-to-peer, 6
  definition, 1
peers
  definition, 1
pieces
  definition, 8
power
  definition, 13
power-law, 2, 6, 13, 16, 24, 26, 49, 54, 55, 63, 93, 95, 143, 144, 147, 148, 160, 207
  definition, 12
power-law distribution, *see* Pareto distribution
power-law graph, 16, 26
  definition, 16
probe, 20, 105
properties, 3
  dynamic connectivity, *see* dynamic connectivity properties
  dynamic peer, *see* dynamic peer properties
  static connectivity, *see* static connectivity properties
  static peer, *see* static peer properties

query properties, 170–203

random graph
  definition, 15
random graph model
  definition, 15
random walks, 51–52
reachability, 153–157
recursive routing, 175
related work, 17–30
  dynamic peer properties, 104–106
  file properties, 83–84
  query properties, 201–202
  sampling, 47–49
remaining uptime, 23, 24, 118, 130–133, 136, 206
  definition, 23
resilience, 158–159
resolution
  definition, 182

sampling, 44–78
  from structured systems, 77–78
  number of samples, 76–77
  related work, 47–49
SC, *see* stable core
scale-free
  definition, 12

scale-free graph, *see* power-law graph
search
  related work, 28–29
seeding time, *see* lingering time
session length, 4, 23, 57–59, 63, 65, 102–105, 111–113, 118, 123–126, 128, 130, 136–137, 141, 201
  definition, 23
shifted Pareto distribution, 125
shifted Pareto distribution, 63, 93
  definition, 12
short-circuit effect
  definition, 29
Skype, 1
  related work, 18
small world, 55, 144, 157–158, 160
small-world graph, 84
  definition, 15
Soft state, 66
SP, *see* stable peers
Split Symbols, 182, 184–186
  definition, 180
stable core, 161–170
  definition, 166
stable peers, 163, 165, 166
  definition, 163
static connectivity properties, 4, 143–160
  related work, 26–27
static peer properties, 4, 79–101
  related work, 22–23
stationary distribution
  definition, 52
strongly regular graph
  definition, 16
structured overlay, *see* distributed hash table
subexponential distribution
  definition, 11
swarming, 8
  definition, 8
symbol size, 176, 179, 180, 182, 186
  definition, 173

tail index
  definition, 125
terminology, 5
test statistic
  definition, 13
time-to-live, *see* TTL
timeout, 32, 34, 35, 43, 59, 66, 69, 192
tracker, 25, 66, 108, 109, 111, 114, 116, 120, 128
  definition, 20
TTL, 7, 8, 153–155, 170
  definition, 7
two-tier, 8, 30, 33, 43, 82, 146, 147, 151, 155, 157, 159
  definition, 7

ultrapeer, 7, 8, 145, 207
  definition, 7
undirected graphs
  definition, 14
unreachable, 35–37, 87, 98, 148, 198
uptime, 4, 86, 105, 118, 128–132, 141, 150, 163–166, 168, 206–208
  definition, 23

V
  definition, 14
vertex
  definition, 14

Watts–Strogatz, 16, 54–56
Weibull distribution, 63, 105, 121, 123, 125, 126, 142
  definition, 11

Zipf distribution, 12, 25, 28, 83, 84, 95, 104
  definition, 12
  file popularity, 22
  query popularity, 28

Bibliography

[1] C. Shirky, "What is P2P ... and what isn't," *O'Reilly Network*, Nov. 2000. [Online]. Available: http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

[2] "P2P networks," 2006. [Online]. Available: http://www.slyck.com/

[3] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P dying or just hiding?" in *Proc. Globecom*, Dallas, TX, Nov. 2004.

[4] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy, "Transport layer identification of P2P traffic," in *Proc. International Measurement Conference*, Taormina, Italy, Oct. 2004.

[5] S. Rhea, D. Geels, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. USENIX*, 2004, pp. 127–140.

[6] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Comput.*, vol. 6, no. 1, 2002.

[7] Clip2.com, Inc., "Gnutella: To the bandwidth barrier and beyond," Nov. 2000.

[8] L. A. Adamic, R. M. Lukose, B. Huberman, and A. R. Puniyani, "Search in power-law networks," *Phys. Rev. E*, vol. 64, no. 46135, 2001.

[9] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and replication in unstructured peer-to-peer networks,” in *Proc. International Conference on Supercomputing*, 2002.

[10] D. Stutzbach and R. Rejaie, “Capturing accurate snapshots of the Gnutella network,” in *Proc. Global Internet Symposium*, Miami, FL, Mar. 2005, pp. 127–132.

[11] D. Stutzbach, R. Rejaie, and S. Sen, “Characterizing unstructured overlay topologies in modern P2P file-sharing systems,” in *Proc. Internet Measurement Conference*, Berkeley, CA, Oct. 2005, pp. 49–62.

[12] D. Stutzbach and R. Rejaie, “Characterizing the two-tier Gnutella topology,” Extended Abstract in *Proc. SIGMETRICS*, Banff, AB, Canada, June 2005.

[13] ——, “Evaluating the accuracy of captured snapshots by peer-to-peer crawlers,” Extended Abstract in *Proc. Passive and Active Measurement Workshop*, Boston, MA, Mar. 2005, pp. 353–357.

[14] ——, “Improving lookup performance over a widely-deployed DHT,” in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

[15] S. Zhao, D. Stutzbach, and R. Rejaie, “Characterizing files in the modern Gnutella network: A measurement study,” in *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 2006.

[16] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, “Sampling techniques for large, dynamic graphs,” in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

[17] A. Rasti, D. Stutzbach, and R. Rejaie, “On the long-term evolution of the two-tier Gnutella overlay,” in *Proc. Global Internet Symposium*, Barcelona, Spain, Apr. 2006.

[18] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "On unbiased sampling for unstructured peer-to-peer networks," in *Proc. Internet Measurement Conference*, Rio de Janeiro, Brazil, Oct. 2006.

[19] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. Internet Measurement Conference*, Rio de Janeiro, Brazil, Oct. 2006.

[20] "Gnutella developer forum," 2005. [Online]. Available: http://www.the-gdf.org/

[21] A. Fisk, "Gnutella dynamic query protocol v0.1," Gnutella Developer's Forum, May 2003. [Online]. Available: http://www.the-gdf.org/index.php?title=Dynamic_Query_Protocol

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 01.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001.

[24] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proc. International Workshop on Peer-to-Peer Systems*, 2002.

[25] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," in *Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation*, San Francisco, CA, Mar. 2004, pp. 99–112.

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329–350.

[27] S. S. Lam and H. Liu, “Failure recovery for structured P2P networks: Protocol design and performance evaluation,” in *Proc. SIGMETRICS*, New York, NY, June 2004.

[28] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi, “A statistical theory of Chord under churn,” in *Proc. International Workshop on Peer-to-Peer Systems*, Ithaca, NY, Feb. 2005.

[29] J. Li, J. Stribling, F. Kaashoek, R. Morris, and T. Gil, “A performance vs. cost framework for evaluating DHT design tradeoffs under churn,” in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[30] R. Cox, A. Muthitacharoen, and R. T. Morris, “Serving DNS using a peer-to-peer lookup service,” in *Proc. International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002.

[31] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with CFS,” in *Proc. ACM Symposium on Operating Systems Principles*, Banff, AB, Canada, Oct. 2001.

[32] M. Castro, P. Drushel, A. Kermarrec, and A. Rowstron, “Scribe: A large-scale and decentralized application-level multicast infrastructure,” *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, Oct. 2002.

[33] G. E. P. Box and N. R. Draper, *Empirical Model-Building and Response Surfaces.* Wiley, Jan. 1987, ch. 13.1, p. 424.

[34] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*, 2nd ed. Wiley-Interscience, 1994, vol. 1.

[35] ——, *Continuous Univariate Distributions*, 2nd ed. Wiley-Interscience, 1995, vol. 2.

[36] M. E. Crovella and A. Bestavros, “Self-similarity in world wide web traffic: Evidence and possible causes,” *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 835–846, 1997.

[37] G. K. Zipf, *The Psychobiology of Language.* New York, NY: Houghton-Mifflin, 1935.

[38] D. Friedman and S. Sunder, *Experimental Methods A Primer for Economists.* New York, NY: Cambridge University Press, 1994, p. 92.

[39] P. Erdös and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[40] D. J. Watts, *Small Worlds: The Dynamics of Networks between Order and Randomness.* Princeton, NJ: Princeton University Press, 1999.

[41] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[42] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs," in *Proc. Symposium on Theory of Computing*, Portland, OR, 2000, pp. 171–180.

[43] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of Internet content delivery systems," in *Proc. Symposium on Operating Systems Design and Implementation*, 2002, pp. 315–327.

[44] L. Plissonneau, J.-L. Costeux, and P. Brown, "Analysis of peer-to-peer traffic on ADSL," in *Proc. Passive and Active Measurement Workshop*, Boston, MA, Mar. 2005, pp. 69–82.

[45] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. ACM Symposium on Operating Systems Principles*, 2003.

[46] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit, "Are file swapping networks cacheable? Characterizing P2P traffic," in *Proc. International Web Caching Workshop*, 2002.

[47] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the Kazaa network," in *Proc. IEEE Workshop on Internet Applications*, 2003.

[48] S. Sen and J. Wang, “Analyzing peer-to-peer traffic across large networks,” *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 219–232, Apr. 2004.

[49] Q. He and M. Ammar, “Congestion control and message loss in Gnutella networks,” in *Proc. Multimedia Computing and Networking*, Santa Clara, CA, Jan. 2004.

[50] P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, and E. Zegura, “Bootstrapping in Gnutella: A measurement study,” in *Proc. Passive and Active Measurement Workshop*, Apr. 2004.

[51] A. Klemm, C. Lindemann, M. Vernon, and O. P. Waldhorst, “Characterizing the query behavior in peer-to-peer file sharing systems,” in *Proc. Internet Measurement Conference*, Taormina, Italy, Oct. 2004.

[52] E. P. Markatos, “Tracing a large-scale peer to peer system: an hour in the life of Gnutella,” in *Proc. CC Grid*, 2002.

[53] K. Sripanidkulchai, “The popularity of Gnutella queries and its implications on scalability,” Jan. 2001. [Online]. Available: http://www-2.cs.cmu.edu/~kunwadee/research/p2p/paper.html

[54] E. Adar and B. A. Huberman, “Free riding on Gnutella,” *First Monday*, vol. 5, no. 10, Oct. 2000.

[55] D. Erman, D. Ilie, A. Popescu, and A. A. Nilsson, “Measurement and analysis of BitTorrent signaling traffic,” in *Proc. Nordic Teletraffic Seminar*, Oslo, Norway, Aug. 2004.

[56] J. Liang, R. Kumar, and K. W. Ross, “The Kazaa overlay: A measurement study,” *Computer Networks Journal (Elsevier)*, 2005.

[57] J. Liang, R. Kumar, Y. Xi, and K. W. Ross, “Pollution in P2P file sharing systems,” in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[58] F. S. Annexstein, K. A. Berman, and M. A. Jovanovic, “Latency effects on reachability in large-scale peer-to-peer networks,” in *Proc. Symposium on Parallel Algorithms and Architectures*, Crete, Greece, 2001, pp. 84–92.

[59] J. Chu, K. Labonte, and B. N. Levine, “Availability and locality measurements of peer-to-peer file systems,” in *Proc. ITCom: Scalability and Traffic Control in IP Networks II Conferences*, July 2002.

[60] S. Saroiu, P. K. Gummadi, and S. D. Gribble, “Measuring and analyzing the characteristics of Napster and Gnutella hosts,” *Multimedia Systems J.*, vol. 9, no. 2, pp. 170–184, Aug. 2003.

[61] R. Bhagwan, S. Savage, and G. Voelker, “Understanding availability,” in *Proc. International Workshop on Peer-to-Peer Systems*, 2003.

[62] F. L. Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulie, “Clustering in peer-to-peer file sharing workloads,” in *Proc. International Workshop on Peer-to-Peer Systems*, 2004.

[63] S. Guha, N. Daswani, and R. Jain, “An experimental study of the Skype peer-to-peer VoIP system,” in *Proc. International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, USA, Feb. 2006.

[64] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice, “Dissecting BitTorrent: Five months in a torrent’s lifetime,” in *Proc. Passive and Active Measurement Workshop*, Apr. 2004.

[65] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, “The BitTorrent P2P file-sharing system: Measurements and analysis,” in *Proc. International Workshop on Peer-to-Peer Systems*, Ithaca, NY, Feb. 2005.

[66] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, “Measurements, analysis, and modeling of BitTorrent-like systems,” in *Proc. Internet Measurement Conference*, Berkeley, CA, Oct. 2005.

[67] M. Yang, Z. Zhang, X. Li, and Y. Dai, "An empirical study of free-riding behavior in the Maze P2P file-sharing system," in *Proc. International Workshop on Peer-to-Peer Systems*, Ithaca, NY, 2005 2005.

[68] F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in P2P protocols," in *Proc. International Workshop on Web Content Caching and Distribution*, 2003.

[69] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek, "Bandwidth-efficient management of DHT routing tables," in *Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation*, Boston, MA, May 2005.

[70] D. Leonard, V. Rai, and D. Loguinov, "On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks," in *Proc. SIGMETRICS*, 2005.

[71] S. Jiang and X. Zhang, "Floodtrail: An efficient file search technique in unstructured peer-to-peer systems," in *Proc. Globecom*, San Francisco, CA, Dec. 2003.

[72] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "AOTO: Adaptive overlay topology optimization in unstructured P2P systems," in *Proc. Globecom*, San Francisco, CA, Dec. 2003.

[73] ——, "A distributed approach to solving overlay mismatching problem," in *Proc. International Conference on Distributed Computing Systems*, 2004.

[74] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-aware topology matching in P2P systems," in *Proc. IEEE INFOCOM*, 2004.

[75] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proc. International Conference on Distributed Computing Systems*, 2002.

[76] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proc. IEEE INFOCOM*, 2004.

[77] Q. Lv, S. Ratnasamy, and S. Shenker, "Can heterogeneity make Gnutella scalable?" in *Proc. International Workshop on Peer-to-Peer Systems*, 2002.

[78] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed caching and adaptive search in multilayer P2P networks," in *Proc. International Conference on Distributed Computing Systems*, 2004.

[79] H. Dämpfling, "Gnutella web caching system: Version 2 specifications client developers' guide," June 2003. [Online]. Available: http://www.gnucleus.com/gwebcache/newgwc.html

[80] D. Ilie, D. Erman, A. Popescu, and A. A. Nilsson, "Measurement and analysis of Gnutella signaling traffic," in *Proc. IPSI Internet Conference*, Stockholm, Sweden, Sept. 2004.

[81] J. Postel, "Transmission Control Protocol," RFC 793, Sept. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[82] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Trans. Netw.*, 2002.

[83] S. Chib and E. Greenberg, "Understanding the Metropolis–Hastings algorithm," *The Americian Statistician*, vol. 49, no. 4, pp. 327–335, Nov. 1995.

[84] W. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[85] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *J. of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[86] B. Bollobás, "A probabilistic proof of an asymptotic formula for the number of labelled regular graphs," *European J. of Combinatorics*, vol. 1, pp. 311–316, 1980.

[87] M. Jerrum and A. Sinclair, "Fast uniform generation of regular graphs," *Theoretical Computer Science*, vol. 73, pp. 91–100, 1990.

[88] C. Cooper, M. Dyer, and C. Greenhill, "Sampling regular graphs and a peer-to-peer network," in *Proc. Symposium on Discrete Algorithms*, 2005, pp. 980–988.

[89] V. Krishnamurthy, J. Sun, M. Faloutsos, and S. Tauro, "Sampling Internet topologies: How small can we go?" in *Proc. International Conference on Internet Computing*, Las Vega, NV, June 2003, pp. 577–580.

[90] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J.-H. Cui, and A. G. Percus, "Reducing large Internet topologies for faster simulations," in *Proc. IFIP Networking*, Waterloo, Ontario, CA, May 2005.

[91] M. P. H. Stumpf, C. Wiuf, and R. M. May, "Subnets of scale-free networks are not scale-free: Sampling properties of networks," *Proc. of the National Academy of Sciences*, vol. 102, no. 12, pp. 4221–4224, Mar. 2005.

[92] A. A. Tsay, W. S. Lovejoy, and D. R. Karger, "Random sampling in cut, flow, and network design problems," *Mathematics of Operations Research*, vol. 24, no. 2, pp. 383–413, Feb. 1999.

[93] A. Lakhina, J. W. Byers, M. Crovella, and P. Xie, "Sampling biases in IP topology measurements," in *Proc. IEEE INFOCOM*, 2003.

[94] D. Achlioptas, A. Clauset, D. Kempe, and C. Moore, "On the bias of traceroute sampling; or, power-law degree distributions in regular graphs," in *Proc. Symposium on Theory of Computing*, Baltimore, MD, May 2005.

[95] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz, "Approximating aggregate queries about web pages via random walks," in *Proc. International Conference on Very Large Databases*, Cairo, Egypt, Sept. 2000, pp. 535–544.

[96] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles, “Methods for sampling pages uniformly from the world wide web,” in *Proc. AAAI Fall Symposium on Using Uncertainty Within Computation*, 2001, pp. 121–128.

[97] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork, “On near-uniform URL sampling,” in *Proc. International World Wide Web Conference*, May 2001, pp. 295–308.

[98] L. Lovász, “Random walks on graphs: A survey,” *Combinatorics: Paul Erdös is Eighty*, vol. 2, pp. 1–46, 1993.

[99] Y. Chawathe, S. Ratnasamy, and L. Breslau, “Making Gnutella-like P2P systems scalable,” in *Proc. ACM SIGCOMM*, 2003.

[100] V. Vishnumurthy and P. Francis, “On heterogeneous overlay construction and random node selection in unstructured P2P networks,” in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

[101] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: Densification laws, shrinking diameters and possible explanations,” in *Proc. KDD*, Chicago, IL, Aug. 2005.

[102] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama, “Distributed uniform sampling in unstructured peer-to-peer networks,” in *Proc. Hawaii International Conference on System Sciences*, Kauai, HI, Jan. 2006.

[103] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating latency between arbitrary Internet end hosts,” in *Proc. Internet Measurement Workshop*, Marseille, France, Nov. 2002.

[104] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the evolution of peer-to-peer systems,” in *Proc. Principles of Distributed Computing*, Monterey, CA, July 2002.

[105] Free Peers, Inc., “BearShare network statistics,” Oct. 2005. [Online]. Available: http://www.bearshare.com/stats/

[106] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the Internet," in *Proc. Internet Measurement Conference*, Taormina, Italy, Oct. 2004.

[107] "slyck.com," 2005. [Online]. Available: http://www.slyck.com

[108] Lime Wire LLC, "Crawler compatability," Gnutella Developer's Forum, Jan. 2003. [Online]. Available: http://www.the-gdf.org/index.php?title=Communicating_Network_Topology_Information

[109] "eMule," 2005. [Online]. Available: http://www.emule-project.net

[110] A. Bharambe and C. Herley, "Analyzing and improving BitTorrent performance," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2005-03, Jan. 2005.

[111] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet service providers fear peer-assisted content distribution?" in *Proc. Internet Measurement Conference*, Berkeley, CA, Oct. 2005, pp. 63–76.

[112] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proc. ACM SIGCOMM*, 2002.

[113] A. Kumar, J. Xu, and E. Zegura, "Efficient and scalable query routing for unstructured peer-to- peer networks," in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[114] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid search schemes for unstructured peer-to-peer networks," in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[115] M. Jovanovic, F. Annexstein, and K. Berman, "Modeling peer-to-peer network topologies through "small-world" models and power laws," in *Proc. TELFOR*, Nov. 2001.

[116] B. Yang, P. Vinograd, and H. Garcia-Molina, "Evaluating GUESS and non-forwarding peer-to-peer search," in *Proc. IEEE International Conference on Distributed Systems*, 2004.

[117] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *Proc. International Conference on Data Engineering*, Mar. 2003.

[118] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *Proc. IEEE INFOCOM*, 2003.

[119] R. H. Wouhaybi and A. T. Campbell, "Phenix: Supporting resilient low-diameter peer-to-peer topologies," in *Proc. IEEE INFOCOM*, 2004.

[120] D. Stutzbach, "Csearch::jumpstart() deletes wrong contacts," June 2005. [Online]. Available: http://forum.emule-project.net/index.php?showtopic=81094

[121] ——, "Kad lookups start in wrong place," June 2005. [Online]. Available: http://forum.emule-project.net/index.php?showtopic=81113

[122] ——, "M_best not initialized and updated properly," June 2005. [Online]. Available: http://forum.emule-project.net/index.php?showtopic=81020

[123] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[124] B. Leong, B. Liskov, and E. D. Demaine, "EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management," in *Proc. nternational Conference on Networks*, Nov. 2004.

[125] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation*, Berkeley, CA, 2004.

[126] D. D. Clark, “Design philosophy of the DARPA Internet protocols,” *ACM SIGCOMM Computer Commun. Rev.*, vol. 25, no. 1, Jan. 1995.

[127] J. Li, J. Stribling, T. M. Gil, R. Morris, and F. Kaashoek, “Comparing the performance of distributed hash tables under churn,” in *Proc. International Workshop on Peer-to-Peer Systems*, 2004.

[128] R. Mahajan, M. Castro, and A. Rowstron, “Controlling the cost of reliability in peer-to-peer overlays,” in *Proc. International Workshop on Peer-to-Peer Systems*, 2003.

[129] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The impact of DHT routing geometry on resilience and proximity,” in *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.