Measurement and Analysis of Internet Content Delivery Systems

Stefan Saroiu

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2004

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Stefan Saroiu

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of Supervisory Committee:

_____

Steven D. Gribble

_____

Henry M. Levy

Reading Committee:

_____

Steven D. Gribble

_____

Henry M. Levy

_____

Michael B. Jones

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform".

Signature_____

Date_____

University of Washington

Abstract

Measurement and Analysis of Internet Content Delivery Systems

by Stefan Saroiu

Co-Chairs of Supervisory Committee:

Professor Steven D. Gribble
Computer Science and Engineering

Professor Henry M. Levy
Computer Science and Engineering

In recent years, the Internet has experienced an enormous increase in the use of specialized content delivery systems, such as peer-to-peer file-sharing systems (e.g., Kazaa, Gnutella, or Napster) and content delivery networks (e.g., Akamai). The sudden popularity of these systems has resulted in a flurry of research activity into novel peer-to-peer system designs. Because these systems: (1) are fully distributed, without any infrastructure that can be directly measured, (2) have novel distributed designs requiring new crawling techniques, and (3) use proprietary protocols, surprisingly little is known about the performance, behavior, and workloads of such systems in practice. Accordingly, much of the research into peer-to-peer networking is uninformed by the realities of deployed systems. This dissertation remedies this situation. We examine content delivery from the point of view of four content delivery systems: HTTP Web traffic, the Akamai content delivery network, and the Kazaa and Gnutella peer-to-peer file sharing networks. Our results (1) quantify the rapidly increasing importance of new content delivery systems, particularly peer-to-peer networks, and (2) characterize peer-to-peer systems both from an infrastructure and workload perspective. Overall, these results provide a new understanding of the behavior of the modern Internet and present a strong basis for the design of newer content delivery systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Three people in graduate school had a profound impact on my work, my learning, my research, and myself overall. The first one is Steve Gribble. I consider myself truly lucky to have the chance to work and learn from him. There are just so many things that I owe to him. Thank you Steve! The second one is Hank Levy. His guidance, advice, elegance, and humor are just unmatchable. I learned a tremendous amount from him, but first I learned to always aim high. The third one is Krishna Gummadi. Krishna taught me countless lessons, of which two stand out. First, there is always a deeper level of understanding when facing a problem. Second, I should never condemn another's judgement because it differs from my own's; we may both be wrong.

There is an overwhelming number of people that I need to thank and acknowledge for their support, guidance, and help. Mike Jones first showed me what research is. John Zahorjan taught me how to always try to find simple and brilliant ideas. David Wetherall, Alon Halevy, and Brian Bershad patiently listened and provided feedback to my incoherent thoughts. Ed Lazowska was always available to chat about all things big and small. Andy Collins, Richard Dunn, David Ely, Robert Grimm, Eric Lemar, Ratul Mahajan, Marianne Shaw, Neil Spring, Mike Swift, and Andrew Whitaker taught me what it means to be a systems student. John Dunagan, Nick Harvey, Jared Saia, Marvin Theimer, and Alec Wolman were terrific collaborators. Alec Wolman and Geoff Voelker wrote beautiful code. Karim Filali, Sorin Lerner, Todd Millstein, and Deepak Verma tolerated me and my interruptions. David Richardson, Ville Aikas, Art Dong, Eric Schwimmer, and the other people at the UW C&C department generously provided assistance in collecting data.

Finally, Delia, Nini, and my parents gave me their unconditional love.

Wow! This was a blast!

Chapter 1

INTRODUCTION

The Internet supports a variety of services, including content delivery [36]. Traditionally, content delivery systems have been based on the client-server architecture, as is the case with the World Wide Web. However, in recent years, the Internet content delivery landscape has changed due to the immense popularity of a new content delivery application: peer-to-peer file-sharing.

Because peer-to-peer file-sharing systems are novel, little is known about their performance, behavior, and workloads in practice. The consequences of these systems' design choices are poorly understood both in the industry and in academia. The sudden popularity of peer-to-peer systems has also resulted in a flurry of research activity [129, 121, 116, 144]; however, little of this research is informed by the realities of deployed systems.

Characterizing peer-to-peer system behavior is difficult for several reasons. First, these systems are fully distributed, without any infrastructure that can be directly measured and monitored. Second, they have novel distributed designs, requiring new and highly scalable crawling and measurement techniques. Finally, they use proprietary naming, encoding mechanisms, and protocols.

Accordingly, the emergence of peer-to-peer file-sharing has created a new research agenda: to understand the nature of modern content delivery and the impact of this new technology on the Internet at large. Current peer-to-peer research further amplifies the importance of this research agenda. We need to understand the forces that drive Internet content delivery today in order to design and build tomorrow's Internet content delivery systems.

This dissertation presents an in-depth analysis of content delivery in today's Internet, based on two measurement studies. The first study examines the workloads of four content

delivery systems: the World Wide Web [18], the Akamai content delivery network [4], and the Kazaa [75] and Gnutella [61] peer-to-peer file-sharing systems. The second study characterizes the infrastructure of two popular peer-to-peer file-sharing systems: Gnutella and Napster [102]. To accomplish this, we also develop novel measurement tools and techniques, including a scalable network tracing system, a crawler for Gnutella, and a fast bandwidth measurement tool.

1.1 Technological Trends Responsible for the Rise of Multimedia Workloads

A new Internet content delivery scenario has recently enjoyed immense popularity levels: millions of people world-wide can download audio and video files over the Internet. In this section, we examine the three technological trends responsible for this recent phenomenon: (1) the development of efficient audio and video file formats; (2) the exponential growth of hard drive storage capacity; and (3) the adoption of *broadband*.

In recent years, a wave of efficient algorithms for encoding audio and video files have emerged, such as MPEG-1 Layer 3 (or MP3), MPEG-2, Windows Media Audio, RealAudio, RealVideo, QuickTime, and AVI. Three reasons are commonly cited for the explosion of these formats [24]. First, these are open standards; their specification is available to anyone (sometimes for a fee). Second, several high-quality encoders and decoders have been made publicly available. Third, these standards have efficient compression rates while preserving a high quality of their content. The result of the proliferation of these algorithms is that it is now possible to cheaply encode music, movies, and television shows.

The second trend responsible for the rise of multimedia workloads is the exponential growth of hard drive storage capacity (see Figure 1.1). For several decades, disks have been experiencing 60% year-over-year improvements in cost and density [66]. The price per megabyte of storage has been decreasing at a rate of 50% per year [66]; today, a quarter terabyte disk costs less than $150. Big and inexpensive disks have enabled home users to store increasing amounts of audio and video content on their personal computers.

Broadband (i.e., fast and always-on Internet connection) is the last ingredient necessary for creating the right environment for downloading audio and video files over the Internet.

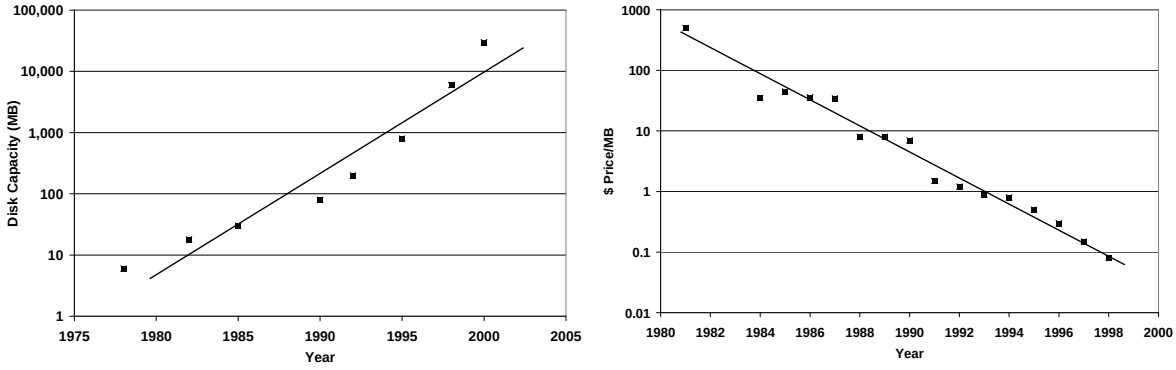

Figure 1.1: Left: Yearly growth of disk drives capacity. Right: Yearly cost of storage (as dollars per megabyte). These statistics are presented in [66].

Downloading audio and video files over broadband is much faster than over "old-fashioned" dialup connections. For example, a typical song can be downloaded in 30 seconds, and a typical encoded movie in less than two hours over a typical cable modem.

The number of users with broadband Internet connections has recently grown at phenomenal rates. From 2001 to 2002, the number of U.S. Internet users with broadband connectivity increased by 60%, while the number of people with Internet dialup connections fell by 10%. In the United States, more than a third of all Internet users owned broadband connections in 2003 [143]. In April of 2002, Nielsen/NetRatings estimated that the majority of total time spent on the Internet came from people connected to broadband networks [143]. Figure 1.2 illustrates the recent rapid growth in the number of broadband users world-wide.

The combination of the first two technological trends, efficient audio and video file formats and large-sized disk drives, enabled home users to create personal collections of audio and video files. Broadband connectivity allowed these users to transfer audio and video content over the Internet. The convergence of these three technological trends created a new content delivery scenario: users could now have the intent to transfer and share their audio and video file collections.

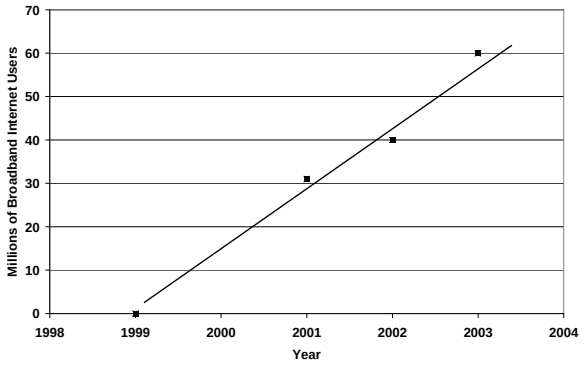

Figure 1.2: The estimated number of broadband Internet users world-wide [143]. These numbers are compiled from two different reports: one by the In-State/MDR research group and one by the Organization for Economic Cooperation and Development (OECD).

1.2 Historical Reasons Responsible for the Rise of Peer-to-Peer File-Sharing Systems

In the previous section, we examined the reasons why multimedia workloads became possible. In this section, we present the reasons why peer-to-peer file-sharing systems were the first to exhibit these workloads, as opposed to more traditional content delivery systems, such as the Web or content delivery networks (CDNs).

The newly emerged multimedia workloads created a tremendous latent demand for a suitable content delivery system. Although commercial Web services were in place, they failed to recognize and meet this demand for several reasons. First, the recording and motion picture industry was afraid to let digital media loose on the Internet without a strong digital rights managements technology (DRM), and initial DRM efforts [65] never materialized in practice. Second, a wave of electronic copyright infringement lawsuits had a chilling effect on initial multimedia Web services efforts [104]. These events resulted in a lack of commercial content delivery services for multimedia workloads.

In the absence of an alternative, peer-to-peer file-sharing systems flourished on the Internet. These systems allowed users to share their audio and video file libraries online. Instead of purchasing multimedia content from a commercial service, users could now transfer con-

tent from other users. Moreover, peer-to-peer was a natural architecture for file-sharing; these systems lack any dedicated infrastructure, relying on voluntary cooperation among users. These systems were essentially "free" to operate and maintain. These convenient properties led millions of people world-wide to participate in peer-to-peer file-sharing systems.

1.3 Qualitative Differences between Peer-to-Peer and Web-based Content Delivery Systems

In the previous section, we examined the reasons why peer-to-peer file-sharing systems became the *de facto* delivery technology for multimedia workloads. In this section, we focus on the differences between peer-to-peer and Web-based content delivery systems.

On the surface, both the Web and peer-to-peer systems appear similar: they both deliver content to end-users. In practice, they differ substantially. For example, their workloads are different: the Web is primarily an interactive system, whereas peer-to-peer file-sharing is primarily a batched system. Their infrastructures are also different: the Web relies on dedicated and well-engineered centralized servers, whereas peer-to-peer relies on volunteer endhosts.

Multimedia usage in peer-to-peer systems has led to substantial differences between P2P and Web workloads. Multimedia content is driven by different forces than those driving Web content. For example: (1) multimedia files are immutable, whereas Web objects are not; (2) their transfer times are much longer than those of Web objects; and (3) their popularities rise and fall over time unlike Web object popularities. An audio file cannot remain a top choice among users for very long periods of time, in contrast to Web objects, which do stay popular for long periods of time (e.g., `www.google.com`).

The lack of a dedicated infrastructure in peer-to-peer systems has led to substantial differences between P2P's architecture and the Web's architecture. For example: (1) peer-to-peer systems need to contend with a potential lack of availability of the underlying constituent hosts; (2) peer-to-peer hosts are likely to exhibit a high degree of heterogeneity with respect to their bandwidths, latencies, and the degree of content shared; and (3) peer-

to-peer systems need to rely on volunteer endhosts; this introduces the problem of "greedy" hosts – hosts that never contribute, but only benefit from the system.

Although these differences between the Web and peer-to-peer file-sharing systems are substantial, they have not been quantified in practice and their implications are largely unknown. The main goal of our dissertation is to remedy this situation.

1.4 Contributions

At a high-level, this dissertation makes three contributions:

1. We explore the workload characteristics of current Internet content delivery systems from the vantage point of the University of Washington (UW). We measure and analyze the workloads of the World Wide Web, the Akamai content delivery network, and the Kazaa and Gnutella peer-to-peer systems, answering three questions:

   (a) In a short period of time, peer-to-peer file-sharing systems have attracted large user populations. These populations use the Internet to transfer content that is larger and larger in size. These trends suggest that new content delivery systems are likely to consume an increasing amount of network resources, especially bandwidth. *What is the bandwidth impact of peer-to-peer file-sharing systems?*

   (b) Although Web-based systems and peer-to-peer file-sharing systems share the same goal of delivering content to end-users, they deliver substantially different types of content. *What content types dominate Internet content delivery traffic?*

   (c) The characteristics of multimedia content, such as the large object sizes and the popularity trends that rise and fall over time, directly shape peer-to-peer file-sharing traffic. *How is bandwidth consumption distributed across objects?*

   We find that the Internet has undergone substantial change in content delivery systems usage in the Internet. Second, we find that the balance of HTTP traffic has changed dramatically over the last several years, with video and audio accounting for a large fraction of traffic, despite the small number of requests involving those data types.

Third, we find that a small number of multimedia objects are responsible for a disproportionately high fraction of the bandwidth consumed by peer-to-peer file-sharing systems. Finally, we find that multimedia object sizes, rather than their widespread usage, are responsible for the large bandwidth consumption of P2P traffic.

2. We explore the infrastructure characteristics of two modern peer-to-peer file-sharing systems: Napster and Gnutella. In particular, using a two-step process, we measure and characterize the properties of peer-to-peer hosts that form the infrastructure for these systems. In the first step, we gather detailed snapshots of these systems in order to collect a large fraction of their entire host population. Second, we probe the discovered hosts in order to measure their network-level characteristics, such as their bandwidths, latencies, and the amount of time they participate in the system. In the context of our study, we ask and answer two questions:

   (a) One of the premises of the peer-to-peer architecture is that peers voluntarily join the system in order to cooperate, exchange resources, and derive benefits from the system. Another premise is that the participants of these systems behave altruistically; there is no notion of self-interest or individual greediness. *How well-behaved are peers in practice?*

   (b) The intended behavior of peer-to-peer system designs is to have single and uniform roles for their peers, without the client-server role demarcation existing in the Web or in content delivery networks. *Are peers uniform in practice, or do we see evidence of "client-like" and "server-like" peers?*

   We find that there is a significant amount of heterogeneity in peer-to-peer systems. Network properties, such as bandwidth and latency, as well as host properties, such as availability and the degree of sharing vary between three and five orders of magnitude across the system. Second, we find clear evidence of client-like or server-like behavior in a significant fraction of these populations. Third, we find that peers tend to deliberately misreport information if there is an incentive to do so.

3. We design, implement, and deploy a measurement infrastructure for modern Internet content delivery systems. To characterize these workloads, we develop a network tracing system. This measurement tool is installed at the Internet border of the University of Washington (UW) and it observes all Internet traffic that flows through the border. To characterize peer-to-peer architectures, we developed a suite of highly scalable tools, including a crawler for Gnutella's overlay network and a fast bandwidth measurement tool.

1.5 Overview of the Dissertation

The rest of this dissertation is organized as follows:

- Chapter 2 provides background information about the architectures of several modern Internet content delivery systems.
- Chapter 3 describes the design and implementation of the measurement infrastructure we use to characterize current content delivery systems.
- Chapter 4 examines the characteristics of workloads of current content delivery systems.
- Chapter 5 examines the characteristics of the infrastructure of current peer-to-peer systems.
- Chapter 6 provides an overview of related research.
- Chapter 7 presents avenues of future research directions.
- Finally, Chapter 8 summarizes the contributions of this dissertation and concludes.

Chapter 2

BACKGROUND

This chapter provides a technical overview of five Internet content delivery systems that we study in this dissertation: the Web, Akamai, Napster, Gnutella, and Kazaa. The main goal of these five systems is the same: to deliver content on the Internet. These systems have well-defined notions of clients, servers, and objects. They all follow the same basic premise: clients fetch objects from servers. Although these systems' goals and basic premises are the same, their architectures are vastly different. In the remainder of this chapter, we provide the background information needed to understand these five Internet content delivery systems, their protocols, and architectures. Our goal is not to produce a "reference manual" of these systems, but instead to provide enough context so that one can understand how these systems can be measured.

2.1 The World Wide Web

First proposed by Tim Berners-Lee in 1989, the World Wide Web is a client-server content delivery system, in which a client connects to a server, sends a request for content, and downloads it. The Web is centralized: a single server is responsible for delivering content to many users over the Internet. Figure 2.1 is an illustration of many Web clients contacting a single Web server. In this figure, each arrow represents a client's request for an object located on the server.

There are three main semantic components of the Web: a naming infrastructure, a document representation language, and an RPC protocol [81]. The Web uses Uniform Resource Locators (URLs) [19, 56] to name content. An example of a URL is: `http://sprobe.cs.washington.edu/download.htm`. A URL consists of three parts: the protocol for communicating with the server (e.g., *http*), the server's name (e.g., *sprobe.cs.washington.edu*), and the name of a file placed on that server (e.g., *download.htm*). Web objects are mutable since

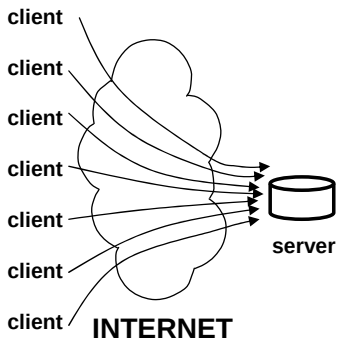

Figure 2.1: The layout of a Web client-server architecture on the Internet.

the content addressed by a URL can change over time.

The Hypertext Markup Language (HTML) is the standard representation for hypertext documents in ASCII format. HTML allows content publishers to format their content, reference images or other objects, and embed hypertext links to other content. The Hypertext Transport Protocol (HTTP) is the RPC protocol for transporting content in the Web. HTTP is human-readable, its protocol messages being encoded in ASCII format. HTTP is layered on top of the Transmission Control Protocol (TCP) [69], a reliable byte-stream transport protocol.

There are two major versions of the HTTP protocol: version 1.0 and 1.1. A significant difference between these two versions is the use of persistent network connections. In HTTP 1.0, a client opens a new network connection for each new request. In HTTP 1.1, requests sent to the same server can share the same network connection. While the introduction of persistent connections in HTTP 1.1 reduces the consumption of network resources by Web traffic, they add additional complexity to Web measurements.

Figure 2.2 shows an example of an HTTP request and its corresponding response. In this example, the URL of the file being fetched is `http://sprobe.cs.washington.edu/sprobe-0.3.tgz`. The Web server responds with an object whose content type is *application/x-tgz-compressed* and whose size is 213,560 bytes. The object requested, *sprobe-0.3.tgz*, immediately follows the HTTP response.

```
GET /sprobe-0.3.tgz HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Referer: http://sprobe.cs.washington.edu/download.htm
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
Host: sprobe.cs.washington.edu
Connection: Keep-Alive
Cookie: SITESERVER=ID=47db74844000504cf900fc8b7cc35775

HTTP/1.1 200 OK
Date: Mon, 30 Aug 2004 22:44:24 GMT
Server: Apache/1.3.31 (Unix)
Last-Modified: Wed, 29 May 2002 05:47:51 GMT
ETag: "158015-34238-3cf46b87"
Accept-Ranges: bytes
Content-Length: 213560
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: application/x-tgz-compressed
Content-Language: en
[content......]
```

Figure 2.2: An example HTTP request (top) and response (bottom).

Because the HTTP protocol is simple and humanly readable, creating traffic filters to match HTTP headers is an easy task. Most Web traffic can be further identified by port numbers. Typically, Web servers use port 80 to listen for incoming HTTP requests. Unless explicitly set differently, the default behavior of a Web browser is to send HTTP requests to the server's port 80.

2.2 Akamai

Akamai [4] is a commercial content delivery network. It consists of thousands of content distribution servers world-wide whose roles are to deliver content to nearby clients. Web servers sign up with Akamai to replicate and serve a portion of their local content from the Akamai's servers. The benefits from using Akamai are two-fold. First, Akamai servers act as proxy caches: they reduce client latencies, server load, and network traffic. Second, Akamai's proxy caching service is available to all Internet clients world-wide, as opposed to proxy caches that only serve a limited client population.

Akamai uses DNS-based name redirection to route client requests to Akamai servers.

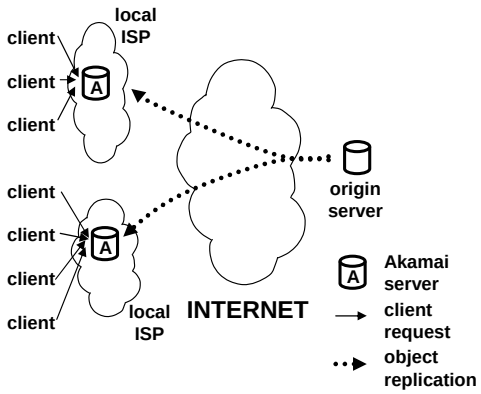

Figure 2.3: The architecture of the Akamai content delivery network. Clients download content from nearby Akamai servers.

Upon receiving a DNS request, an Akamai nameserver returns the address of an Akamai content server located near the client issuing the request. Because of the transparent nature of DNS name resolution, Akamai's client redirection mechanism does not require any modifications to client software, server protocols, or Web applications. Figure 2.3 is an illustration of Akamai's architecture: Web client's requests are routed to nearby Akamai servers instead of the origin Web server.

Like the Web, Akamai delivers HTML objects, it uses URLs to name objects, and it uses HTTP to transport content. This makes Akamai traffic look similar to Web traffic. One way to distinguish Akamai traffic from Web traffic is to identify whether the server delivering content is an Akamai server or a regular Web server. We used this approach to identify Akamai traffic in our measurement studies (see Section 3.1.4 for our complete methodology for tracing Akamai workloads).

2.3 Peer-to-Peer File-Sharing Systems

All Internet peer-to-peer file-sharing systems have similar goals: to facilitate the location and exchange of files (typically images, audio, or video) among a large group of independent users connected through the Internet. In these systems, files are stored on the computers

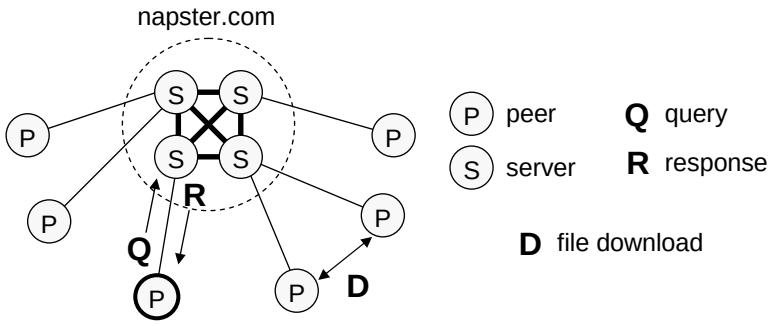

Figure 2.4: The architecture of Napster. Peers send their queries to the Napster central cluster of servers, in a client-server manner. Files are transferred among peers, in a peer-to-peer manner.

of the individual users or peers, and exchanged through a direct connection between the downloading and uploading peers, often over HTTP. All peers in this system are symmetric: they have the ability to function both as a client and a server. This symmetry distinguishes peer-to-peer systems from other content delivery system architectures. Although the process of exchanging files is similar in peer-to-peer systems, their architectures differ substantially. In this section, we provide overviews of three peer-to-peer systems: Napster, Gnutella, and Kazaa.

2.3.1 Napster

In this section, we describe the architecture of Napster during the time of our measurements, in May 2001. Since then, Napster has been ordered to shutdown by the legal courts, it has been sold, and it has reopened under a new name: Napster 2.0.

Napster [102] has a hybrid client-server and peer-to-peer architecture. In Napster, searching for files is done in a client-server fashion. Transferring content is done using a peer-to-peer approach: a peer downloads content directly from another peer. Figure 2.4 illustrates the architecture of Napster.

In Napster, a large cluster of dedicated central servers maintains an index of all files currently shared by on-line peers. Each peer connects to one of the central servers and,

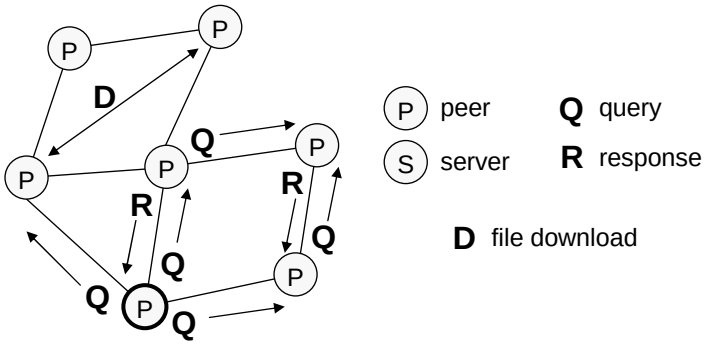

Figure 2.5: The architecture of Gnutella. Peers form an overlay over which queries are flooded. Files are transferred among peers.

once connected, it uploads information about all the content it shares. Queries are also sent to the central servers. Napster servers cooperate to process each query and to return a list of matching files and locations. Once the query results are received, the requesting peer directly initiates a file download from another peer found in the query result set.

In addition to maintaining an index of shared files, the centralized servers also monitor the state of each peer in the system, keeping track of metadata such as the peers' reported connection bandwidth and the duration that the peer has remained connected to the system. This metadata is returned with the query results. In this way, the requesting peer has useful information to distinguish between the quality of different download sites. We also used this metadata to collect peer information in our measurement studies (see Section 5.2 for our complete methodology for tracing Napster).

2.3.2 Gnutella

In this section, we describe the architecture of Gnutella during the time of our measurements, in May 2001. This corresponds to the Gnutella protocol version 0.4. Since 2001, the Gnutella protocol has been upgraded to version 0.6 and it includes complexities that are not discussed in this dissertation.

Gnutella [61] is another example of a file-sharing peer-to-peer system. Unlike Napster, there are no centralized servers in Gnutella. Instead, Gnutella peers form an "overlay

Table 2.1: Message types in the Gnutella protocol ver. 0.4.

Type	Description	Information
Ping	Probe for other peers	None
Pong	Response to a ping	IP+port of the responding peer; # of files and total KB of shared data
Query	Search request	Search filter
Query Hit	Successful response to a query	IP+port+network bandwidth of the responding peer; # of results and the result set
Push	File download request for peers behind firewalls	IP+port of the requester; identifier and file index of the peer behind firewall

network" by forging point-to-point connections with a set of neighbors. To locate a file, a peer initiates a controlled flood of the overlay network by sending a query packet to all of its neighbors. Upon receiving a query packet, a peer checks if any locally stored files match the query. If so, the peer sends a query response packet back towards the query originator. Whether or not a file match is found, the peer continues to flood the query through the overlay. Once content is found, downloading content is done using the HTTP protocol. Figure 2.5 illustrates an example of a Gnutella overlay network.

The Gnutella protocol [37] consists of five basic message types showed in Table 2.3.2. To help maintain the overlay as the users enter and leave the system, the Gnutella protocol includes ping and pong messages that help peers to discover other nodes. Pings and pongs behave similarly to query/query-response packets: any peer that sees a ping message sends a pong back towards the originator and forwards the ping onwards to its own set of neighbors. Ping and query packets thus flood through the network; the scope of flooding is controlled with a time-to-live (TTL) field that is decremented on each hop. Peers occasionally forge new neighbor connections with other peers discovered through the ping/pong mechanism. Note that it is possible to have several disjoint Gnutella overlays simultaneously coexisting in the Internet; this contrasts with Napster, in which peers are always connected to the same cluster of central servers.

Gnutella uses the HTTP protocol to transfer files. However, unlike the Web, each Gnutella connection first starts with a Gnutella protocol handshake exchange. For example, for the Gnutella protocol version 0.4, each Gnutella flow initially starts with an exchange

```
GET /get/1128/Noam Chomsky - Outwits A Young Capitalist.mp3 HTTP/1.0
Host: 66.16.6.33:6346
User-Agent: Gnucleus 1.8.6.0
Connection:
Keep-Alive
Range: bytes=0-3653915

HTTP 200 OK
Cache-Control: no-cache
Connection: Keep-Alive
Server: Gnucleus 1.5.2.0
Content-Type: audio/mpeg
Content-Length: 3653916
Content-Range: bytes 0-3653915/3653916
[content......]
```

Figure 2.6: An example Gnutella request (top) and response (bottom).

of "GNUTELLA CONNECT/0.4" and "GNUTELLA OK" messages. After this handshake takes place, Gnutella starts using HTTP to transfer files. Figure 2.6 shows an example of the headers for a simple Gnutella file download request and the corresponding response.

2.3.3 Kazaa

Kazaa [75] is another example of a file-sharing peer-to-peer system. Like Gnutella, Kazaa is based on a "pure" peer-to-peer architecture. However, unlike Gnutella, Kazaa introduces the notion of supernodes. Supernodes are voluntarily conscripted peers, that typically have better network connectivity and longer availabilities. Each peer connects to exactly one supernode and uploads metadata about its shared content. This allows the supernode to maintain an index of all files that its peers share. The role of supernodes is reminiscent of Napster servers; but, in contrast with Napster, Kazaa supernodes are not dedicated servers; instead, they are volunteer peers. Figure 2.7 illustrates the architecture of Kazaa.

To find a file, a user sends a query with keywords to its supernode. For each match in the local neighborhood of nodes, the supernode returns the IP address and the metadata corresponding to the match. Each supernode also maintains connections with other supernodes, creating an overlay network among the supernodes. When a supernode receives a query, it may forward the query to one or more of the supernodes to which it is connected.

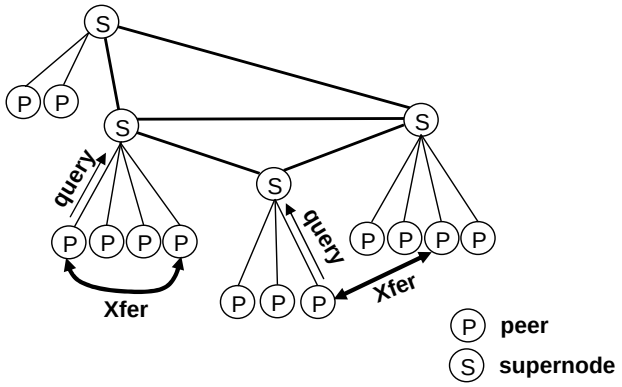

Figure 2.7: The architecture of Kazaa. Each peer is connected to exactly one supernode. Supernodes form an overlay network. Files are transferred among peers.

Like Gnutella, Kazaa uses the HTTP protocol to transfer files. Unlike Gnutella, Kazaa does not use an initial handshake to signal that the connection is not a Web connection, but a Kazaa connection. Kazaa file-transfer traffic includes Kazaa-specific HTTP headers (e.g., "X-Kazaa-IP"). Figure 2.8 shows an example of the headers for a simple Kazaa request and the corresponding response.

Kazaa supports "file swarming," which is the ability to transfer non-overlapping fragments of the same file from multiple peers simultaneously. With file swarming, an object can be downloaded in pieces (often called "chunks") from several sources in parallel. When a user disconnects, the Kazaa client software suspends all ongoing file transfers; these transfers are resumed once the user re-connects. As a result, file downloads in Kazaa may be long-lived, spanning days or even weeks. Occasionally, a client may download only a subset of the entire object (either because the user gives up or because the object becomes permanently inaccessible in the middle of a request).

To support file swarming and suspend/resume file transfers, the Kazaa protocol must determine unambiguously whether two files stored by two different peers are the same or not. File names and file descriptors (e.g., artist name, album name, and other user-entered text) cannot serve as unique object identifiers, as the same object can have multiple names or file descriptors. Instead, Kazaa uses a hash of the object's content as a unique object

```
GET /10809/Beatles - Yesterday.mp3 HTTP/1.1
Host: 123.52.193.31:1214
UserAgent: KazaaClient Oct 18 2002 01:57:14
X-Kazaa-Username: JohnDoe
X-Kazaa-Network: KaZaA
X-Kazaa-IP: 192.168.1.191:1214
X-Kazaa-SupernodeIP: 206.158.106.142:1715
Connection: close
X-Kazaa-XferId: 119345
X-Kazaa-XferUid: ytCzDgo+2sTohMl2+1Y2jYkCY6NwCA==

HTTP/1.1 200 OK
Content-Length: 2015232
Accept-Ranges: bytes
Date: Mon, 13 Sep 2004 02:15:50 GMT
Server: KazaaClient Jul 15 2002 20:37:36
Connection: close
Last-Modified: Tue, 15 Oct 2002 15:36:45 GMT
X-Kazaa-Username: JaneDoe
X-Kazaa-Network: KaZaA
X-Kazaa-IP: 13.52.193.31:1214
X-Kazaa-SupernodeIP: 198.7.216.79:2577
X-KazaaTag: 5=123
X-KazaaTag: 21=128
X-KazaaTag: 6=Beatles
X-KazaaTag: 8=Help
X-KazaaTag: 4=Yesterday
X-KazaaTag: 3==0GM/G/3Q9eSM81HvjFQbZ1Z0Jt0=
Content-Type: audio/mpeg
[content......]
```

Figure 2.8: An example Kazaa request (top) and response (bottom).

identifier. The content hash field enables the Kazaa client to directly search for content, without issuing a new keyword query [90].

2.4 Summary

In this section, we provided overviews of five content delivery systems: the Web, Akamai, Napster, Gnutella, and Kazaa. While these systems' architectures are different, their goal is the same: to deliver content on the Internet. Each of these five systems has well-defined notions of clients, servers, and objects. These can be identified by examining their protocol headers (e.g., HTTP headers) or by actively probing these systems (e.g., examining the metadata returned by Napster servers). In the next chapter, we describe the measurement techniques we developed to measure these Internet content delivery systems.

Chapter 3

THE DESIGN AND IMPLEMENTATION OF A MEASUREMENT INFRASTRUCTURE FOR CONTENT DELIVERY SYSTEMS

In this chapter, we describe the design and implementation of our measurement infrastructure for characterizing Internet content delivery systems. Section 3.1 describes the design and implementation of a tracing system used to collect workloads of content delivery systems. Section 3.2 presents the architecture and challenges for designing a crawler for a peer-to-peer system. Finally, Section 3.3 describes a technique for measuring the bandwidths of Internet hosts, including peers in a peer-to-peer system.

3.1 Collecting and Analyzing Content Delivery Workloads

Characterizing Internet content delivery systems is important to system designers, system developers, content publishers, and Internet service providers. System designers and developers can learn how their designs and software perform in practice. They can make informed decisions about which components to optimize and how to fine-tune configuration parameters. Content publishers can tailor their content to suit the performance characteristics of the content delivery system. Internet service providers can learn about the relative importance of network applications based on the amount of network resources they consume.

Collecting and analyzing content delivery workloads is not an easy task. The scale and diversity of the Internet make it impossible to know if the traffic characteristics observed at one location are applicable at other locations. Many of these systems are decentralized; there is no single authority or entity that can gather a snapshot of the entire system's activity. Some peer-to-peer systems lack any dedicated infrastructure; there is no hardware component or centralized vantage point that one can instrument and monitor to understand how the system behaves. Finally, some of the these systems' protocols are proprietary or

encrypted. As a result, one must reverse-engineer their protocols to instrument and monitor them.

In this section, we present the design and implementation of a network tracing system used to collect workloads of four content delivery systems: the Web, Akamai, Gnutella, and Kazaa. Our system is an extension of that originally built by Wolman et al. [137, 139, 138, 34] to collect Web workloads. There are two high-level differences between Wolman's system and ours. First, our system identifies and reconstructs peer-to-peer traffic (e.g., Gnutella and Kazaa) and content delivery network traffic (e.g., Akamai), in addition to Web traffic. Second, our system scales to an order of magnitude higher along several different dimensions: network speeds (we trace at speeds of 1Gbps rather than 100Mbps), amount of data (we collect terabytes of data rather than gigabytes), and robustness (we continuously trace for several consecutive months rather than several consecutive days). Our tracing system was in use at the University of Washington Internet border for approximately two years. We monitored Web and Akamai workloads from April 2002 to December 2002, and Gnutella and Kazaa workloads from April 2002 to July 2003.

3.1.1 Hardware Configuration

Figure 3.1 illustrates the UW network border router configuration. Our tracing system uses *passive network monitoring* to observe all packets flowing through our four campus network switches. Each of the campus switches is configured to enable port mirroring. Traffic from the campus subnet routers is load balanced across the four campus subnet switches.

Our trace host is a dual-processor Dell Precision Workstation 530 with 2.0GHz Pentium III Xeon CPUs and a gigabit Ethernet SysKonnect SK-9843 network card; it runs FreeBSD 4.5. The monitoring host has two internal 40 GB SCSI hard disks. For log storage and backup, we built and configured two file servers of approximately 1 TB each. These file servers were in turn connected to our monitoring host via a local gigabit network. Under this configuration, the monitoring host collected online traces on its internal disks. Each morning at 3 a.m., when traffic is at its lowest rate, the monitoring host dumped all its daily logs to the file server via the local gigabit network.

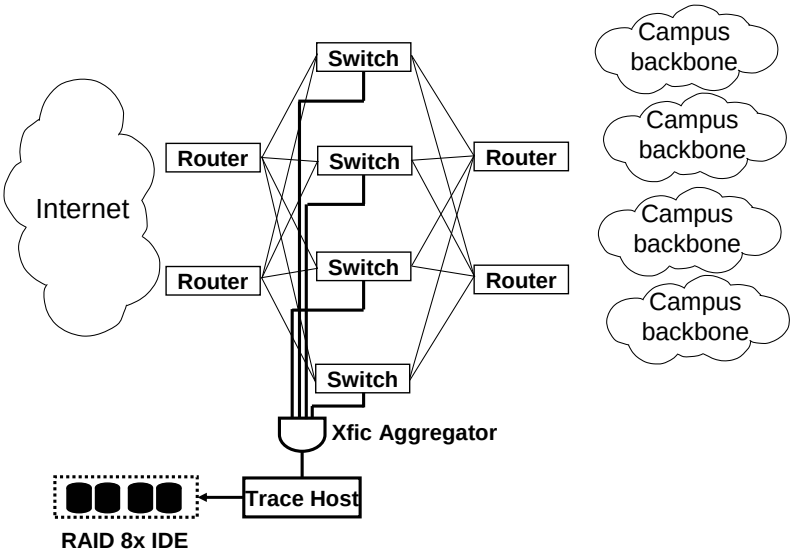

Figure 3.1: The integration of our network tracing system within the University of Washington network border configuration.

3.1.2 The Software Architecture of Our Tracing System

In this section, we briefly describe the software originally developed by Wolman et al. [137, 139, 138, 34] for tracing Web workloads. We illustrate the key points of this software, emphasizing our contributions to scaling the system to modern workloads. A complete description of the original system can be found in [137].

The design principle driving the development of this tracing infrastructure is avoiding kernel modifications: its entire functionality is implemented at user-level. We pay a performance penalty by implementing functionality at user-level; in exchange, we reduce development time by avoiding the extra complexity arising from debugging within the kernel.

Kernel Modifications

As we mentioned above, our goal is to make as few kernel modifications as possible. We found it necessary to make five kernel modifications. First, we increase the size of the receive descriptor rings from 16 to 256 in the Ethernet device driver. These rings maintain memory

descriptors used by the network card driver to DMA data into memory. This increase allows the device to handle larger bursts of packets.

Our second modification is an enhancement of the network card driver. Our network card (SK-9843) has the capability of reporting the number of packets seen on the network wire and the number of packets DMA-ed into memory. Comparing these two values detects any packet losses occurring at the network driver layer. We modify the network driver to export a system call that queries the card and reports these counters' values. We develop an application that runs periodically and compares these two values, validating that no packet losses are occurring.

Our third modification is to reduce the granularity of the timestamping mechanism in the kernel's packet filter module. The default packet timestamping mechanism reads the Intel 8254 programmable timer chip for each received packet [131]. This mechanism was too slow for our network speeds: our monitoring host was spending 60-70% of its CPU timestamping packets using this configuration. Instead, we modify the packet filter module to use the default hardware oscillator whose interrupt rate was increased from 100Hz to 1,000Hz. This reduces our CPU load for handling timer interrupts to about 12%, in exchange for a loss in packet timestamping accuracy. An alternative solution would have been to read the Pentium cycle counter for each packet and use it to calculate the corresponding timestamp. Although cycle counters are fast to read, they may reflect time inaccurately, because the exact clock frequency varies with time as the processor temperature changes [40].

Our fourth modification is adding a shortcut to the kernel's TCP processing stack. Once data is received from the network card's Ethernet processing layer, the data is immediately passed to the packet capture library and all other kernel functionality is aborted. In this way, we avoid any extra performance cost due to the kernel handling all incoming packets and deciding which action to take appropriately.

Our final modification is to change one buffer management constant in the packet-filter module. This constant controls the maximum number of packets queued for a packet-filter. The goal of increasing this constant is to permit the monitoring process to transfer packets from the kernel less frequently, reducing the kernel crossing overhead.

Robustness

We use this system to collect traces of many consecutive months. The biggest challenge when running this system for a long period of time is to ensure that no memory leaks occur. For example, a single memory leak manifesting one in a million packets can be ignored when tracing for several consecutive days only; the same memory leak must be resolved when tracing for several consecutive months.

Over the course of one year of constant tracing we have experienced three different kinds of trace interruptions. First, we ran out of disk space for our logs several times. Second, we had one failure when parsing a multicast packet. This was the first time the University decided to conduct studies using IP Multicast traffic. Our application did not handle these packets properly, and the parsing code crashed the application. Third, we had to deliberately stop our network tracing system for seven days because the area reserved for our monitoring host was recarpeted.

TCP Reconstruction

Once a packet is received, it is passed to a TCP reconstruction module. This module identifies all packets containing TCP segments and reconstructs the TCP connection that those segments belong to. Each TCP connection's first packet is examined to determine whether the connection is an HTTP connection. Once a connection is classified as HTTP, we examine all its subsequent packets to locate other HTTP headers in case of persistent connections. We ignore all subsequent packets of non-HTTP connections.

Our TCP reconstruction module maintains an in-memory data structure that contains the collected information about each single HTTP flow. Once a flow ends, we record this information in our logs and we remove it from the data structure. Determining when flows terminate is challenging. Even if a flow receives a connection ending packet (i.e., a TCP RST or a TCP FIN), it can still receive other packets, due to possible packet reorderings. For these reasons, flows must still be maintained in our data structure even after connection ending packets are encountered. On the other hand, removing flows from our data structure frees up memory for new flows. This is a fundamental trade-off: whether to retain "old"

flows in memory in case new packets arrive or to free up memory to make space for "new" flows. To resolve this tension, we implement the following policy: we remove all terminated connections which are idle for at least three minutes, and we remove all connections which are idle for at least fifteen minutes, even if they were not terminated.

HTTP Reconstruction

Our HTTP reconstruction module identifies and parses HTTP headers. The extracted information is copied into a data structure that is written to the log, once the HTTP flow is garbage-collected from memory. HTTP parsing is a complex task due to a wide variety of implementations that do not properly follow the HTTP protocol. This is especially exacerbated by the Gnutella and Kazaa protocols.

Privacy and Anonymization

It is crucial to protect the privacy of students, faculty, and staff using the University of Washington campus networks. Our goal is to ensure that no information in the trace can be associated with a person or a machine. We reach this goal through a combination of three different approaches. First, we ensure that all information collected is anonymized. There is no knowledge of which client accesses which server, which server is accessed, which URL is accessed, or which file is downloaded. Second, we erase two low-order bits for all UW client IP addresses in our trace before the anonymization step occurs. This step collapses four different UW IP addresses into one single IP address, adding an extra layer of privacy to the UW addresses. Finally, we do not write any non-anonymized information to stable storage.

Our IP address anonymization hashes each IP address octet separately. This ensures that IP address locality is preserved, even after anonymization. This allows us to identify whether IP addresses belong to the same network region within the campus (e.g., campus departments) without compromising the users' privacy.

Our constraint that no non-anonymized data packet should ever reach stable storage has many implications to the overall design of our tracing infrastructure. Most related work

on building high-speed network tracing tools copies the packets from the network interface to stable storage. For our tracing tool, TCP and HTTP connection reconstruction, data anonymization, and data compression are done online, in memory.

Logging and Compression

After anonymization, the collected information is passed to the logging and compression module. This module uses the LZO library for data compression; this library allows fast, on-the-fly compression and decompression [105]. It also has support for quickly recovering from data corruption. Our final traces are very large; over the course of two years of tracing we have accumulated over two terabytes of data. At this scale, bit flips are almost certain; in fact, we encountered one bit flip. LZO reported the corresponding data blob as unreadabable and immediately recovered and continued to decompress the log.

3.1.3 Collecting Web Workloads

We use port numbers to classify HTTP flows as Web flows. If an HTTP request is directed to ports 80 or 8080, we classify both the request and the associated response as Web traffic. Although Akamai traffic also uses these ports, we exclude it from the Web traffic.

3.1.4 Collecting Akamai Workloads

As we outlined in Section 2.2, Akamai traffic is similar to Web traffic. To identify which traffic is served by Akamai servers rather than other Web servers, we compiled a list of IP addresses for Akamai caches. Whenever our monitoring application detects a UW client sending an HTTP request to one of these Akamai IP addresses, we record this traffic as Akamai traffic.

We used the following methodology to collect the IP addresses of Akamai content distribution servers. We started with the BGP tables collected by RouteView [99]. These tables provided BGP views of the entire Internet from 60 different vantage points. For each network address prefix encountered in this table, we selected randomly an IP address within this prefix range and we did a reverse DNS lookup to discover its authoritative name server.

In this manner, we obtained a list of 25,318 authoritative name servers spread throughout the Internet. We sent a recursive DNS query to each server for a name in an Akamai managed-domain (e.g., `a338.g.akamaitech.net`). Because Akamai redirects DNS queries to nearby Akamai servers, we collected a list of 3,966 unique Akamai servers in 928 different networks.

Although this technique discovered a large number of Akamai IP addresses, it had a number of limitations. First, the BGP views provided by RouteView were incomplete, and they provided a coarse-grained view of the Internet address prefix ranges. Second, this methodology did not discover the backup Akamai servers' IP addresses. Third, different IP addresses could reference different interfaces of the same Akamai server. Finally, our list of Akamai IP addresses was not updated while collecting the traces presented in this dissertation.

3.1.5 Collecting Gnutella Workloads

To collect Gnutella traffic, we enhanced our HTTP reconstruction module to also parse and identify Gnutella handshake messages. All Gnutella connections, including the overlay network connections, start with a Gnutella handshake. Once a Gnutella handshake occurs, we label the flow as a Gnutella flow and we parse each of its subsequent packets to extract HTTP headers. If no HTTP headers are found, the connection is considered to be an overlay connection and discarded; otherwise, the connection is a Gnutella HTTP file transfer. In this case, the normal HTTP parsing module extracts and logs information about the transfer.

There are many Gnutella software clients available. Some follow different character sequences than the ones specified by the HTTP RFCs. For example, a popular Gnutella client uses HTTP responses that are malformed. The proper syntax of an HTTP response with error code 200 is "HTTP/1.0 200 OK"; instead, "HTTP 200 OK" was used. To handle these cases, we extend our HTTP parsing module with many different spelling variants for HTTP commands.

3.1.6 Collecting Kazaa Workloads

Like Gnutella, Kazaa uses the HTTP protocol to transfer files. Unlike Gnutella, there are only a few Kazaa software clients available, and they all closely adhere to the HTTP RFCs. As a result, parsing Kazaa HTTP flows is much simpler and less error-prone.

Many Kazaa transfers are partial transfers of a single object. Kazaa allows "swarming" across different peers. Each client downloading an object can request different byte ranges of the object from different clients. Partial transfers add extra complexity to our matching code of requests and responses. Several workload properties cannot be extracted from partial transfers only. For example, measuring the average amount of time a client waits to download an entire file requires an analysis of multiple partial file downloads from multiple peers. Our matching code handles multiple partial downloads: for each downloaded object, we construct of a map of all blocks (32 KB blocks) of the object. For each block, we record the connection over which the block is downloaded, allowing us to reconstruct and to compute all statistics dealing with full object downloads.

Kazaa file-transfer traffic consists of unencrypted HTTP transfers; all transfers include Kazaa-specific HTTP headers (e.g., "X-Kazaa-IP"). These headers make it simple to distinguish between Kazaa activity and other HTTP activity. They also provide enough information to precisely identify which object is being transferred in a given transaction.

3.1.7 Performance Evaluation

Figures 3.2 and 3.3 characterize the network load on our tracing system during the first half of 2003. Figure 3.2 illustrates the traffic exchanged over time between the University of Washington and the rest of the Internet measured in megabits per second. Figure 3.3 shows the packet arrival rate over time during this period. The peak bandwidth during this trace period was 896Mbps. The peak arrival rate was over 250,000 packets per second. In these figures, a gap is visible between April 10, 2003 and April 17, 2003. During this period, we had to deliberately stop our tracing system due to the recarpeting of the area reserved for our monitoring host.

Our network system also recorded the amount of packet loss it observed. There are three

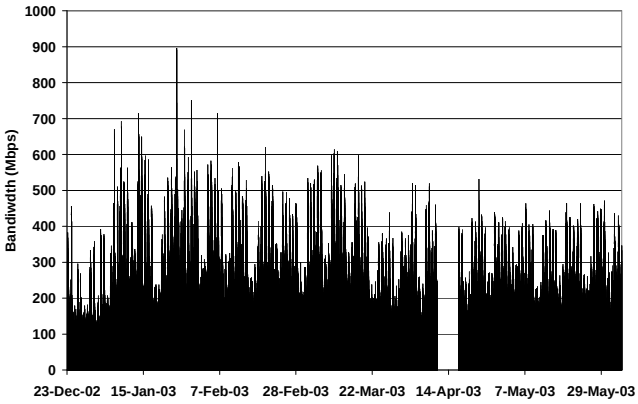

Figure 3.2: Bandwidth consumed between the University of Washington and the rest of the Internet from December 23rd, 2002 to June 6th, 2003.

places where packets can be lost. First, packets can be lost due to the switch port mirroring mechanism. To measure this loss rate we gathered the statistics reported by the network switches. We found this loss rate to stay constant at about 0.000005%. Second, packets can be lost during the DMA from the network card into memory. We measured this loss rate periodically and we always found it to be zero. Finally, packets can be lost when copied from the kernel to the user-level packet capturing module. We found this packet loss rate to be approximately one in a million, or 0.0001%.

3.1.8 Summary

In this section, we presented an overview of our network monitoring system used to characterize content delivery workloads. Our system was deployed at the Internet border of the University of Washington for approximately two years.

We provided a detailed look at the engineering details involved in building and operating a high-speed network tracing tool. Our system made a deliberate effort to maintain the user's privacy. As a consequence of our privacy design constraints, our system performed on-the-fly data anonymization and TCP and HTTP reconstruction. We also described the kernel-level modifications made to increase the performance of the system and to detect low-level packet losses.

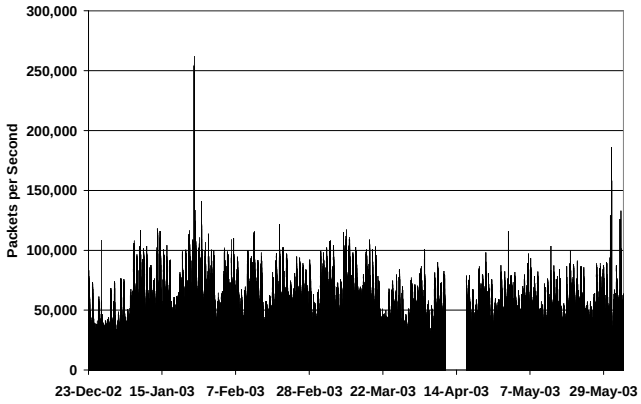

Figure 3.3: Number of packets per second exchanged between the University of Washington and the rest of the Internet from December 23rd, 2002 to June 6th, 2003.

We characterized the performance of our tracing system including the traffic load and the packet loss characteristics. Our system handled peak rates of 896Mbps and 250,000 packets per second.

3.2 Crawling the Gnutella Peer-to-Peer System

The previous section presented our passive tracing system for collecting content delivery workloads. In the remainder of this chapter, we focus on the techniques and tools we used to actively probe peer-to-peer systems. We start by describing the crawler for the Gnutella overlay network.

To capture the global view of the system and its peers, a crawler must gather nearly instantaneous snapshots of the Gnutella overlay network. To discover peers, we used the Gnutella protocol's ping/pong messages. The crawler starts by connecting to several well-known peers, such as `gnutellahosts.com` or `router.limewire.com`. Then, it begins an iterative process of sending ping messages to known peers, adding newly discovered peers to its list of known peers. In addition to the IP address of a peer, each pong message contains metadata about the peer, including the number and total size of files being shared.

There are two modes in which the crawler can operate: a slow mode that preserves the overlay network topology, and a fast mode in which network topology information is lost.

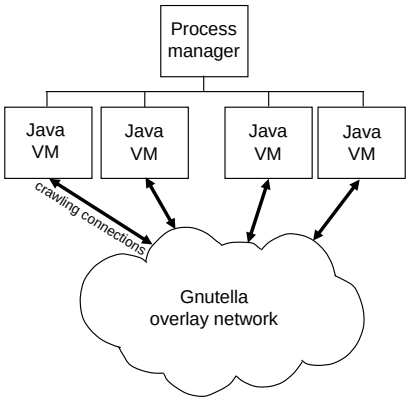

Figure 3.4: The architecture of the Gnutella crawler.

Switching between modes is done based on the TTL value set in the crawler's ping messages. In the slow mode, we set the TTL field to the value two. In this case, the newly discovered peers are neighbors of known peers, and the crawler can infer overlay links and recreate the overlay topology. In the fast mode, we set the TTL field to a large value. In this case, the crawler discovers new peers more rapidly, but the network topology information is lost.

Figure 3.4 illustrates the architecture of our crawler. It consists of a process manager and several crawling Java virtual machines. Many of these crawling machines are located on different hardware nodes, although it is possible to run several virtual machines on the same node. The Process Manager coordinates and controls the crawling virtual machines. It maintains a list of all IP addresses discovered in Gnutella and it dispatches these IP addresses to each virtual machine. When discovering a new IP address, a virtual machine sends it back to the process manager. This mechanism ensures that the set of IP addresses crawled by each crawler is disjoint.

A single crawl lasts approximately two minutes; during this time, we would typically gather between 8,000 and 10,000 unique peers. According to measurements reported by Clip2 [64] at the time we crawled, this corresponds to at least 25% to 50% of the total population of peers in the system at any time. After two minutes, we terminate the crawler, save the crawling results to a file and begin another crawl iteration to gather our next

Gnutella population snapshot. We seed new crawls with previously discovered peers. This has the advantage of making our crawls reach more peers more quickly; however, it has the potential of consistently omitting overlay fragments disconnected from the previously crawled peers. To avoid this problem, after a fixed number of crawls (30 in our case), our crawler discards all previously discovered peers and uses `gnutellahosts.com` and `router.limewire.com` as seeds only.

The Gnutella crawler was written in Java, and ran using the IBM Java 1.18 JRE on Linux 2.2.16. The crawler ran in parallel on a small number of dual-processor Pentium III 700MHz computers with 2GB RAM. Our Gnutella trace spanned eight days (Sunday May 6th, 2001 through Monday May 14th, 2001) and captured 1,239,487 Gnutella peers.

3.3 SProbe: A Fast Tool for Measuring Bottleneck Bandwidth in Uncooperative Environments

A rigorous Internet peers' characterization must include measuring their Internet connections' bandwidths. The precise meaning of "network bandwidth" can be defined according to several metrics, including:

- throughput – the number of transferred bytes over a network path during a fixed amount of time.

- available bandwidth – the maximum attainable throughput of a newly started flow over a network path.

- bottleneck bandwidth – the maximum throughput that is ideally obtained across the slowest link of a network path.

Measuring throughput and available bandwidth is complex in practice, since they depend on many different factors, including latency, loss rate, network path load, and TCP implementation details. Instead, we decided to use the bottleneck link bandwidth as a first-order approximation to the available bandwidth.

In this section, we present the design, implementation, and evaluation of SProbe, a bottleneck bandwidth estimation tool based on the packet-pair technique. SProbe is designed to be fast and scalable. Unlike previous bottleneck bandwidth tools, SProbe can measure bottleneck bandwidth in an *uncooperative* environment, one in which measurement software is only deployed on the local host.

First, we introduce some terminology. A network path is *symmetric* if the bottleneck bandwidths in each direction are equal, otherwise the path is *asymmetric*. In an uncooperative environment, the path direction from the cooperative endhost to the uncooperative endhost is referred to as the *downstream* direction, in contrast to the *upstream* direction, from the uncooperative to the cooperative endhost.

3.3.1 Measuring Bottleneck Bandwidth in the Downstream Direction

To measure downstream bottleneck bandwidth, a large packet pair needs to traverse the path from the local to the remote host. Due to the nature of the uncooperative environment, it is practically impossible to know the time dispersion of the packet pair arriving at the remote host. However, in certain cases, the remote host responds to the received packets. Assuming that the responses do not queue at the bottleneck bandwidth on the reverse path, their time dispersion can be used as an approximation to the initial, large packet pair time dispersion.

SProbe exploits the TCP protocol to perform the sequence of packet exchanges described above. In the TCP protocol, a SYN packet pair sent to an inactive port of the remote machine is answered by a RST packet pair. Regular SYN packets are 40-byte packets, having no payload data. SProbe appends a large payload (1460 bytes) to each sent SYN packet. However, the answered RST packets are small packets (40 bytes), which are unlikely to queue at the bottleneck link on the reverse path. On the local host, SProbe uses the time dispersion of the received RST packet pair as an approximation to the remote host time dispersion of the SYN packet pair. Note that this packet exchange only relies on a correct TCP implementation at the remote host and is sufficient to measure the downstream bottleneck bandwidth. Figure 3.5 illustrates the packet exchange initiated by SProbe.

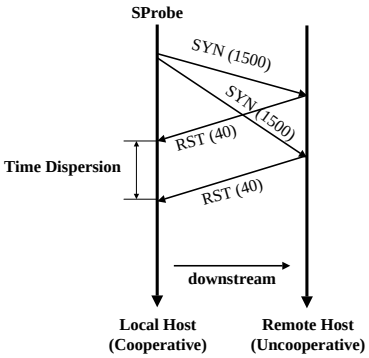

Figure 3.5: The packet exchange initiated by SProbe to estimate the bottleneck bandwidth in the downstream direction.

Our technique has several limitations. Because firewalls silently drop SYN packets to inactive ports, SProbe cannot distinguish hosts behind firewalls from offline hosts. When packet loss occurs, SProbe will timeout after five seconds and terminate with an *unanswered* probe message. It is up to the user or the application invoking SProbe to decide whether to retry the measurement.

Previous active tools send many probing packets to decrease the likelihood of cross traffic, reordering or multi-channel links interference. Unfortunately, this approach has two inherent drawbacks – (1) it is *unscalable* and (2) it is *slow*. Instead, SProbe shapes its probing traffic to reveal information about the conditions of the measured network path. It sends a train of small, 40-byte SYN packets in the middle of which a large packet pair (1500-byte) is placed. The remote endhost answers with a train of RST packets. SProbe analyzes the time dispersion of the RST packets in the train and uses two heuristics that determine the presence of cross traffic.

The Shuffle Heuristic Test: SProbe uses the RST packets' sequence numbers to determine whether the ordering of the received packets has been preserved relative to the ordering of the sent SYN packets. When packets were reordered, SProbe discards the measurement and returns an *unknown* estimate. Note that a multi-channel link is also likely to reorder the train packets.

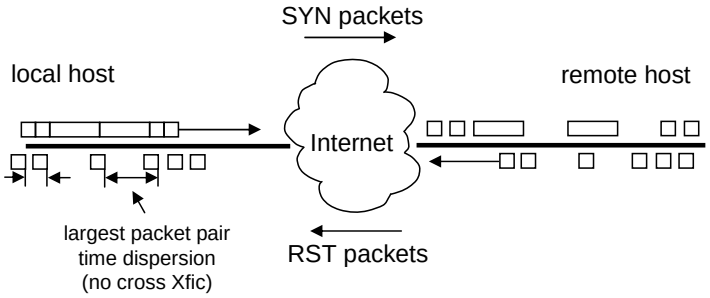

Figure 3.6: The consistent arrival times heuristic test. The time dispersion of the RST packets corresponding to the large SYN packet pair (the middle packets) should be the largest dispersion among packet pairs when no cross traffic is present.

The Consistent Arrival Times Heuristic Test: When the RSTs are not reordered, the time dispersion of the two RSTs in the middle of the train should be larger than the dispersion of any of the smaller 40-byte packet pairs. If not, it is likely that a rogue packet was placed between a 40-byte packet pair. This indicates cross-traffic presence during probing; SProbe discards the measurement and returns an *unknown* estimate. Figure 3.6 illustrates the packet train exchange and the consistent arrival times heuristic test.

3.3.2 Measuring Bottleneck Bandwidth in the Upstream Direction

Unlike the downstream direction, where virtually no cooperation is required from the remote host, the upstream direction requires a low degree of cooperation. In particular, the remote host must be willing to serve a small amount of data. For example, running a Web server or a Gnutella peer is sufficient for an upstream SProbe measurement.

SProbe starts by initiating a TCP connection, advertising a 1500-byte MSS, and sending an application level request for data (e.g. an HTTP GET-request). On receiving the request, the remote server consecutively sends as many data packets as the initial value of its congestion window. Using their arrival time dispersion, SProbe produces an estimate of the upstream bottleneck bandwidth. A previous study shows that most Web servers start with an initial congestion window size of at least two segments [106].

If the initial congestion window size is one, SProbe will timeout after waiting for a packet

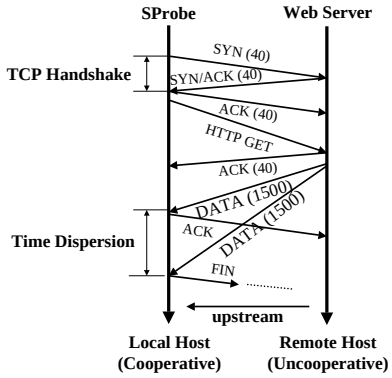

Figure 3.7: The packet exchange initiated by SProbe to estimate the bottleneck bandwidth in the upstream direction.

pair and will acknowledge the packet received. This will cause an increase in the server's congestion window size, and a packet pair is sent. In the case of a packet loss, SProbe acknowledges the first unreceived byte, until the remote host sends a packet pair. Finally, when the packets received are not large, SProbe discards the measurement and reports an *unknown* estimate. This is a violation of the packet pair assumptions, and we believe that, in this case, a measurement tool should return an *unknown* estimate, rather than produce a potentially inaccurate one.

Using this technique, SProbe produces *very fast* estimates. Figure 3.7 shows the typical packet exchange sequence used by SProbe. In general, three round-trip times (RTT) are sufficient for SProbe to produce an estimate.

3.3.3 Evaluation

Previous work has presented careful, controlled experiments evaluating the strengths and weaknesses of the packet pair estimation technique [78, 110, 88]. Rather than performing another investigation into the efficacy of the technique itself, we instead focus on demonstrating the practicality of using SProbe in the context of a large-scale network measurement project. We use SProbe to gather bottleneck bandwidths of 1,180,205 unique Gnutella IP addresses. SProbe's speed and ability to work in uncooperative environments are essential

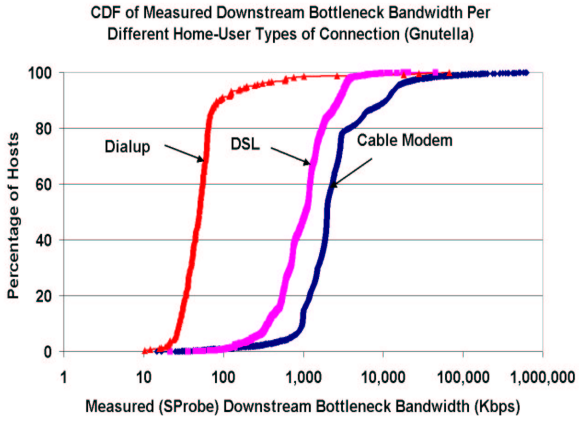

Figure 3.8: Measured downstream bottleneck bandwidths for Gnutella peers, grouped by the types of their Internet connections.

for gathering this large number of measurements.

The typical Gnutella user connects to the Internet using a dialup modem, cable modem, or DSL connection. By resolving a peer's IP addresses into a DNS name, we can often deduce the type of Internet connection used by that peer. For example, if a peer's DNS name contains the word *dialup*, it is extremely likely that the peer uses a dialup modem to connect to the Internet. Unfortunately, the same Internet connection type can imply different network speeds (e.g., a dialup modem's speed can be anywhere between 300bps and 57.6Kbps). In Figure 3.8, three different subsets of peers have been measured:

- **dialup**: 282 peers whose names contain the word *dialup*.
- **DSL**: 2270 peers whose names contain the word *dsl*.
- **cable modem**: 10,259 peers whose names end in *home.com*, *shawcable.net* or *cgocable.net*.

As Figure 3.8 shows, SProbe reported a bottleneck bandwidth of less than 100Kbps for 92% of the dialup modems. Similarly, 91% of the measured DSLs have bottleneck bandwidth estimates between 128Kbps and 3Mbps, which corresponds to the range of speeds supported

by DSL modems. For cable modems, it is less clear what their actual connection speed is, since different vendors report downstream speeds of anywhere between 1Mbps to 10Mbps. Only 78% of the cable modems have a measured bottleneck bandwidth within this range; SProbe reports a bottleneck bandwidth of higher than 10Mbps for 11% of them. These results illustrate SProbe's accuracy.

3.4 Summary and Contributions

In this chapter, we presented the design and implementation of a series of tools for measuring content delivery systems. At a high-level, we made three contributions:

- We provided a detailed look at the engineering details involved in building a high-speed packet monitoring system. Our system is an extension of the system used by Wolman et al. [137, 139, 138, 34] to collect Web workloads. There are two key differences between Wolman et al.'s system and ours:
  - Our system identifies and reconstructs peer-to-peer traffic (e.g., Gnutella and Kazaa) and content delivery network traffic (e.g., Akamai), in addition to Web traffic.
  - Our system scales to an order of magnitude higher along several different dimensions: network speeds (we trace at speeds of 1Gbps rather than 100Mbps), amount of data (we collect terabytes of data rather than gigabytes), and robustness (we continuously trace for several consecutive months rather than several consecutive days).

- We presented the design of a distributed crawler for the Gnutella peer-to-peer system. We used this crawler to crawl the Gnutella overlay network for eight consecutive days in May 2001, capturing 1,239,487 Gnutella peers.

- We presented the design, implementation, and evaluation of SProbe, a bottleneck bandwidth measurement tool designed to be fast and scalable. Unlike previous bottle-

neck bandwidth tools, SProbe can measure bottleneck bandwidth in an uncooperative environment, one in which measurement software is only deployed on the local host.

Chapter 4

WORKLOAD CHARACTERIZATION OF CONTENT DELIVERY SYSTEMS

This dissertation's major goal is to characterize the recently emerged Internet content delivery workloads. For example, file-sharing workloads are likely to be driven by different forces than Web workloads. Similarly, peer-to-peer system designs are likely to directly impact and shape their workloads' characteristics. To understand the current Internet content delivery landscape, we need to understand today's content delivery workloads.

In this chapter, we examine the workloads of four content delivery systems: the World Wide Web, the Akamai content delivery network, and the Kazaa and Gnutella peer-to-peer file sharing systems. To perform this study, we traced all incoming and outgoing Internet traffic at the University of Washington, a large organization with over 60,000 students, faculty, and staff. In this chapter, we analyze a nine-day trace that saw over 500 million transactions and over 20 terabytes of HTTP data. From this data, we provide a detailed characterization and comparison of content delivery systems. Our results quantify: (1) the extent to which peer-to-peer traffic has overwhelmed Web traffic as a leading consumer of Internet bandwidth, (2) the dramatic differences in the characteristics of objects being transferred as a result, (3) the impact of the two-way nature of peer-to-peer communication, and (4) the ways in which peer-to-peer systems are *not* scaling, despite their explicitly scalable design. For example, our measurements show that an average peer of the Kazaa peer-to-peer network consumes 90 times more bandwidth than an average Web client in our environment. Overall, we present implications for large organizations, service providers, network infrastructure, and general content delivery.

The remainder of this chapter is organized as follows. Section 4.1 describes the measurement methodology we used to collect and process our data. In Section 4.2 we give a high-level overview of the workload we have traced at the University of Washington. Sec-

tion 4.3 provides a detailed analysis of our trace from the perspective of objects, clients, and servers, focusing in particular on a comparison of peer-to-peer and Web traffic. Section 4.4 concludes and summarizes our results.

4.1 Methodology

In this section, we provide a brief overview of the traffic collection and analysis infrastructure developed to perform this study. A significantly more detailed description of the tracing system is provided in Section 3.1. We use *passive network monitoring* to collect traces of traffic flowing between the University of Washington (UW) and the rest of the Internet. UW connects to its ISPs via two border routers; one router handles outbound traffic and the other inbound traffic. These two routers are fully connected to four switches on each of the four campus backbones. Each switch has a monitoring port that is used to send copies of the incoming and outgoing packets to our monitoring host.

Our software installs a kernel packet filter [97] to deliver TCP packets to a user-level process. This process reconstructs TCP flows, identifies HTTP requests within the flows (properly handling persistent HTTP connections), and extracts HTTP headers and other metadata from the flows. Because Kazaa and Gnutella use HTTP to exchange files, this infrastructure is able to capture P2P downloads as well as Web and Akamai traffic. We anonymize sensitive information such as IP addresses and URLs, and log all extracted data to disk in a compressed binary representation.

4.1.1 Distinguishing Traffic Types

Our trace captures two types of traffic: *HTTP traffic*, which can be further broken down into Web, Akamai, Kazaa, and Gnutella transfers, and *non-HTTP TCP traffic*, including Kazaa and Gnutella search traffic. If an HTTP request is directed to port 80, 8080, or 443 (SSL), we classify both the request and the associated response as Web traffic. Similarly, we use ports 6346 and 6347 to identify Gnutella HTTP traffic, and port 1214 to identify Kazaa HTTP traffic. A small part of our captured HTTP traffic remains unidentifiable; we believe that most of this traffic can be attributed to less popular peer-to-peer systems (e.g.,

Napster [102]) and by compromised hosts turned into IRC or Web servers on ports other than 80, 8080, or 444. For non-HTTP traffic, we use the same Gnutella and Kazaa ports to identify P2P search traffic.

We will use the following definitions when classifying traffic:

- **Akamai:** HTTP traffic on port 80, 8080, or 443 that is served by an Akamai server.
- **Web:** HTTP traffic on port 80, 8080, or 443 that is not served by an Akamai server; thus, "Web traffic" does not include Akamai traffic.
- **Gnutella:** HTTP traffic sent to ports 6346 or 6347 (this includes file transfers, but excludes search and control traffic).
- **Kazaa:** HTTP traffic sent to port 1214 (this includes file transfers, but excludes search and control traffic).
- **P2P:** the union of Gnutella and Kazaa.
- **non-HTTP TCP traffic:** any other TCP traffic, including protocols such as NNTP and SMTP, HTTP traffic to ports other than those listed above, traffic from other P2P systems, and control or search traffic on Gnutella and Kazaa.

4.2 High-Level Data Characteristics

Table 4.1.1 shows summary statistics of object transfers. This table separates statistics from the four content delivery systems, and further separates inbound data (data requested by UW clients from outside servers) from outbound data (data requested by external clients from UW servers).

Despite its large client population, the University is a net *provider* rather than consumer of HTTP data, exporting 16.65 TB but importing only 3.44 TB. The peer-to-peer systems, and Kazaa in particular, account for a large percentage of the bytes exported and the total bytes transferred, despite their much smaller internal and external client populations. Much

Table 4.1: HTTP trace summary statistics: trace statistics, broken down by content delivery system; *inbound* refers to transfers from Internet servers to UW clients, and *outbound* refers to transfers from UW servers to Internet clients. Our trace was collected over a nine day period, from Tuesday May 28th through Thursday June 6th, 2002.

	WWW		**Akamai**		**Kazaa**		**Gnutella**	
	inbound	outbound	inbound	outbound	inbound	outbound	inbound	outbound
HTTP transactions	329,072,253	73,001,891	33,486,508	N/A	11,140,861	19,190,902	1,576,048	1,321,999
unique objects	72,818,997	3,412,647	1,558,852	N/A	111,437	166,442	5,274	2,092
clients	39,285	1,231,308	34,801	N/A	4,644	611,005	2,151	25,336
servers	403,087	9,821	350	N/A	281,026	3,888	20,582	412
bytes transferred	1.51 TB	3.02 TB	64.79 GB	N/A	1.78 TB	13.57 TB	28.76 GB	60.38 GB
median object size	1,976 B	4,646 B	2,001 B	N/A	3.75 MB	3.67 MB	4.26 MB	4.08 MB
mean object size	24,687 B	82,385 B	12,936 B	N/A	27.78 MB	19.07 MB	19.16 MB	9.78 MB

of this is attributable to a large difference in average object sizes between Web and P2P systems.

The number of clients and servers in Table 4.1.1 shows the extent of participation in these systems. For the Web, 39,285 UW clients accessed 403,437 Internet Web servers, while for Kazaa, 4,644 UW clients accessed 281,026 external Internet servers. For Akamai, 34,801 UW clients download Akamai-hosted content provided by 350 different Akamai servers. In the reverse direction, 1,231,308 Internet clients accessed UW Web content, while 611,005 clients accessed UW-hosted Kazaa content.

Figure 4.1 shows the total TCP bandwidth consumed in both directions over the trace period. The shaded areas show HTTP traffic, broken down by content delivery system; Kazaa and Gnutella traffic are grouped together under the label "P2P." All systems show a typical diurnal cycle. The smallest bandwidth consumer is Akamai, which currently constitutes only 0.2% of observed TCP traffic. Gnutella consumes 6.04%, and Web traffic is the next largest, consuming 14.3% of TCP traffic. Kazaa is the largest contributor in our trace, consuming 36.9% of TCP bytes. These four content delivery systems account for 57% of total TCP traffic, leaving 43% for other TCP-based network protocols (streaming media, news, mail, and so on). TCP traffic represents over 97% of all network traffic at UW. This closely matches published data on Internet 2 usage [70].

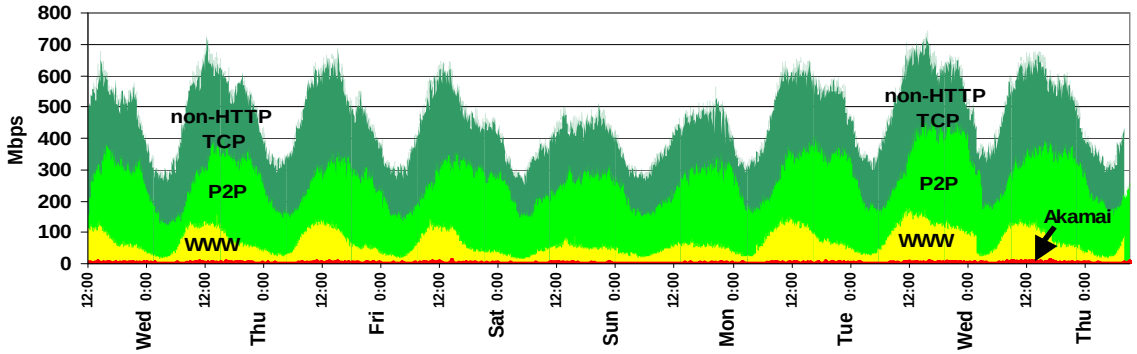

Figure 4.1: TCP bandwidth: total TCP bandwidth consumed by HTTP transfers for different content delivery systems. Each band is cumulative; this means that at noon on the first Wednesday, Akamai consumed approximately 10 Mbps, Web consumed approximately 100 Mbps, P2P consumed approximately 200 Mbps, and non-HTTP TCP consumed approximately 300 Mbps, for a total of 610 Mbps.

Figures 4.2a and 4.2b show inbound and outbound data bandwidths, respectively. From Figure 4.2a we see that while both Web and Kazaa have diurnal cycles, the cycles are offset in time, with Web peaking in the middle of the day and Kazaa peaking late at night. For UW-initiated requests, Web and Kazaa peak bandwidths have the same order of magnitude; however, for requests from external clients to UW servers, the peak Kazaa bandwidth dominates Web by a factor of three. Note that the Y-axis scales of the graphs are different; Web peak bandwidth is approximately the same in both directions, while external Kazaa clients consume 7.6 times more bandwidth than UW Kazaa clients.

In comparison, Figures 4.8(a) and (b) show the inbound and outbound request rates over time, for the Web and Kazaa, respectively. Even though Kazaa accounts for a substantial fraction of bytes transferred, the rate of Kazaa requests is up to two orders of magnitude less than that of the Web. Interestingly, Kazaa inbound and outbound requests have approximately the same rate, while for the Web, UW clients issue many more requests to external Web servers than external clients issue to UW Web servers.

Figure 4.3a and 4.3b show the top 10 content types requested by UW clients, ordered by bytes downloaded and number of downloads. While GIF and JPEG images account for 42% of requests, they account for only 16.3% of the bytes transferred. On the other

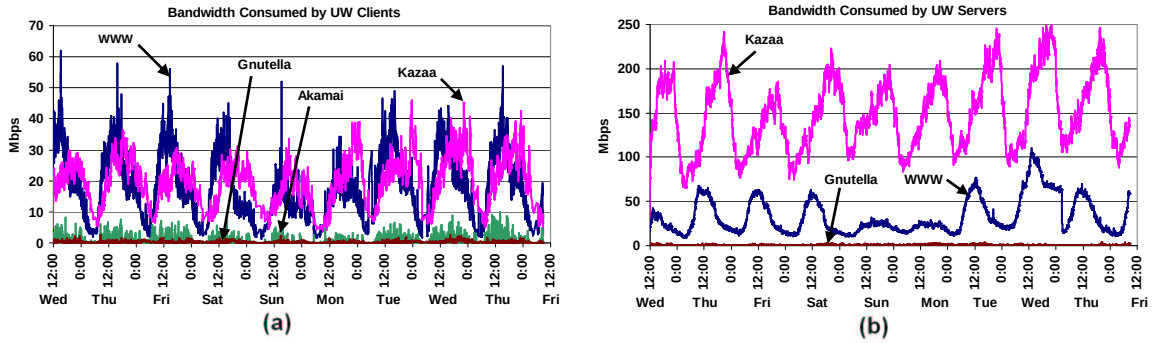

Figure 4.2: UW client and server TCP bandwidth: bandwidth over time (a) accountable to Web and P2P downloads from UW clients, and (b) accountable to Web and P2P uploads from UW servers.

hand, AVI and MPG videos, which account for 29.3% of the bytes transferred, constitute only 0.41% of requests. HTML is significant, accounting for 14.6% of bytes and 17.8% of requests. The 9.9% of bytes labeled "HASHED" in Figure 4.3a are Kazaa transfers that cannot be identified; of the non-hashed Kazaa traffic that can be identified, AVI and MPG account for 79% of the bytes, while 13.6% of the bytes are MP3.

It is interesting to compare these figures with corresponding measurements from the 1999 study of the same population [138]. Looking at bytes transferred as a percent of total HTTP traffic, HTML traffic has decreased 43% and GIF/JPG has decreased 59%. At the same time, AVI/MPG (and Quicktime) traffic has increased by nearly 400%, while MP3 traffic has increased by nearly 300%. (These percentages numbers include an estimate of the appropriate portion of the hashed bytes contributing to all content types).

In summary, this high-level characterization reveals substantial changes in content delivery systems usage in the Internet, as seen from the vantage point of UW. First, the balance of HTTP traffic has changed dramatically over the last several years. In 2002, P2P traffic overtook Web traffic as the largest contributor to HTTP bytes transferred. Second, although UW is a large publisher of Web documents, P2P traffic makes the University an even larger exporter of data. Finally, the mixture of object types downloaded by UW clients has changed, with video and audio accounting for a substantially larger fraction of traffic than three years ago, despite the small number of requests involving those data types.

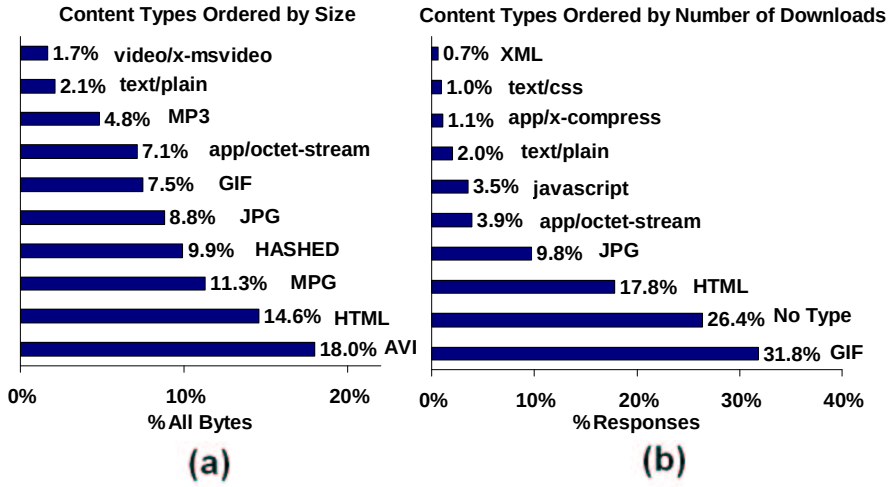

Figure 4.3: Content types downloaded by UW clients: a histogram of the top 10 content types downloaded by UW clients, across all four systems, ordered by (a) size and (b) number of downloads.

4.3 Detailed Content Delivery Characteristics

The changes in Internet workload that we have observed raise several questions, including: (1) what are the properties of the new objects being delivered, (2) how are clients using the new content-delivery mechanisms, and (3) how do servers for new delivery services differ from those for the Web? We attempt to answer these questions in the subsections below.

4.3.1 Objects

Data in Section 4.2 suggests that there is a substantial difference in typical object size between P2P and Web traffic. Figure 4.4 illustrates this in dramatic detail. Not surprisingly, Akamai and Web object sizes track each other fairly closely. The median Web object is approximately 2KB, which matches previous measurement studies [63]. The Kazaa and Gnutella curves are strikingly different from the Web; the median object size for these P2P systems is approximately 4MB – a *thousand-fold* increase over the average Web document size! Worse, we see that 5% of Kazaa objects are over 100MB. This difference has the potential for enormous impact on Internet performance as these systems grow.

Figure 4.5 shows a cumulative distribution of bytes fetched by UW clients for the 1,000

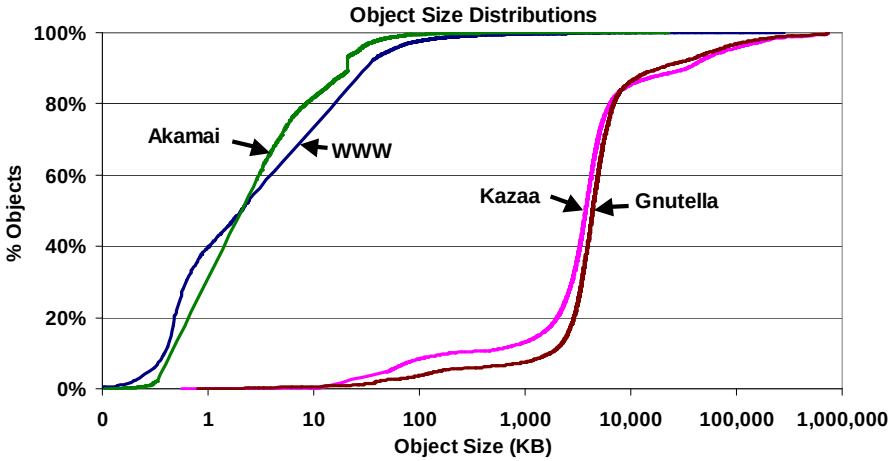

Figure 4.4: Object size distributions: cumulative distributions (CDFs) of object sizes.

highest bandwidth-consuming objects in each of the four CDNs. The Akamai curve rises steeply, with the top 34 objects accounting for 20% of the Akamai bytes transferred; Akamai traffic is clearly skewed to its most popular documents. For Kazaa, we see that a relatively small number of objects account for a large portion of the transferred bytes as well. The top 1,000 Kazaa objects (out of 111K objects accessed) are responsible for 50% of the bytes transferred. For the Web, however, the curve is much flatter: the top 1,000 objects only account for 16% of bytes transferred.

To understand this better, we examined the 10 highest bandwidth-consuming objects for Web, Akamai and Kazaa, which are responsible for 1.9%, 25% and 4.9% of the traffic for each system, respectively. The details are shown in Table 4.3.1. For Web, we see that the top 10 objects are a mix of extremely popular small objects (e.g., objects 1, 2 and 4), and relatively unpopular large objects (e.g., object 3). The worst offender, object 1, is a small object accessed many times. For Akamai, although 8 out of the top 10 objects are large and unpopular, 2 out of the top 3 worst offenders are small and popular. Kazaa's inbound traffic, on the other hand, is completely consistent; all of its worst offenders are extremely large objects (on the order of 700MB) that are accessed only ten to twenty times.

Comparing Kazaa inbound and outbound traffic in Table 4.3.1 shows several differences. The objects that contribute most to bandwidth consumption in either direction are similarly

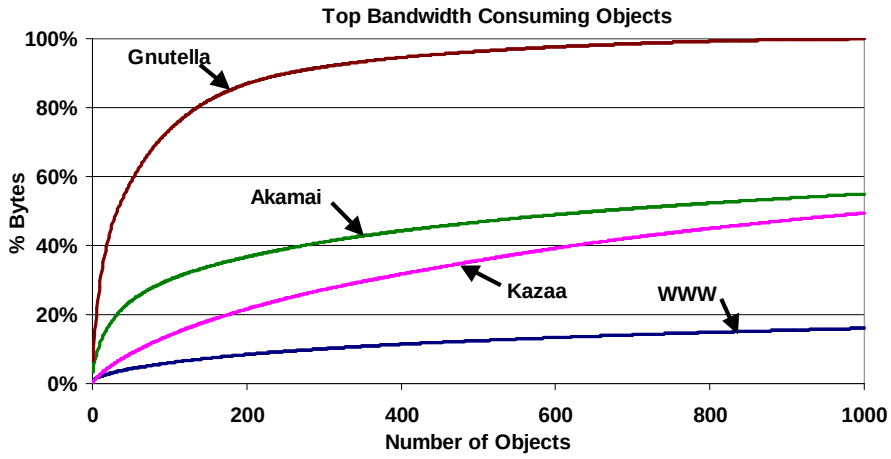

Figure 4.5: Top bandwidth consuming objects: a CDF of bytes fetched by UW clients for top 1,000 bandwidth-consuming objects.

sized, but UW tends to export these large objects more than it imports them. A small number of UW clients access large objects from a small number of external servers, but nearly thirty times as many external clients access similarly-sized objects from a handful of UW servers, leading to approximately ten times as much bandwidth consumption. This suggests that a *reverse* cache that absorbs outbound traffic might benefit the University even more than a forward cache that absorbs inbound traffic.

Figure 4.3 in the previous section showed a breakdown of all HTTP traffic by content type for UW-client-initiated traffic. Figure 4.6 shows a similar breakdown, by bytes, but for each individual CDN. Not surprisingly, the highest component of Web traffic is text, followed closely by images, while Akamai is dominated by images (42% of bytes are GIF and JPEG). In contrast, Kazaa is completely dominated by video (80%), followed by 14% audio; Gnutella is more evenly split with 58% video and 36% audio.

4.3.2 Clients

The previous subsection considered the characteristics of *what* is transferred (the object view); here we consider *who* is responsible (the client view). Because Web and Akamai are indistinguishable from a UW client's perspective, this section presents these two workloads

Table 4.2: Top 10 bandwidth consuming objects: the size, bytes consumed, and number of requests (including the partial and unsuccessful ones) for the top 10 bandwidth consuming objects in each system. For Kazaa, instead of requests, we show the number of clients and servers that participated in (possibly partial) transfers of the object.

	WWW (inbound)			Akamai			Kazaa (inbound)				Kazaa (outbound)			
	obj. size (MB)	GB consumed	# requests	obj. size (MB)	GB consumed	# requests	obj. size (MB)	GB consumed	# clients	# servers	obj. size (MB)	GB consumed	# clients	# servers
1	0.009	12.29	1,412,104	22.37	4.72	218	694.39	8.14	20	164	696.92	119.01	397	1
2	0.002	6.88	3,007,720	0.07	2.37	45,399	702.17	6.44	14	91	699.28	110.56	1000	4
3	333	6.83	21	0.11	1.64	68,202	690.34	6.13	22	83	699.09	78.76	390	10
4	0.005	6.82	1,412,105	9.16	1.59	2,222	775.66	5.67	16	105	700.86	73.30	558	2
5	2.23	3.17	1,457	13.78	1.31	107	698.13	4.70	14	74	634.25	64.99	540	1
6	0.02	2.69	126,625	82.03	1.14	23	712.97	4.69	17	120	690.34	64.97	533	10
7	0.02	2.69	122,453	21.05	1.01	50	715.61	4.49	13	71	690.34	54.90	447	16
8	0.03	1.92	56,842	16.75	1.00	324	579.13	4.30	14	158	699.75	49.47	171	2
9	0.01	1.91	143,780	15.84	0.95	68	617.99	4.12	12	94	696.42	43.35	384	14
10	0.04	1.86	47,676	15.12	0.80	57	167.18	3.83	39	247	662.69	42.28	151	2

combined.

Figure 4.7a shows a cumulative distribution of bytes downloaded by the top 1000 bandwidth-consuming UW clients for each CDN. It's not surprising that the Web+Akamai curve is lower; the graph shows only a small fraction of the 39K Web+Akamai clients, but nearly a quarter of the 4644 Kazaa clients. Nevertheless, in both cases, a small number of clients account for a large portion of the traffic. In the case of the Web, the top 200 clients (0.5% of the population) account for 13% of Web traffic; for Kazaa, the top 200 clients (4% of the population) account for 50% of Kazaa traffic. The next 200 Kazaa clients account for another 16% of its traffic. Clearly, a very small number of Kazaa clients have a huge overall bandwidth impact.

To see the impact more globally, the curves in Figure 4.7b show the *fraction* of the total HTTP bytes downloaded by the most bandwidth-consuming clients for each CDN (the curves are cumulative). This allows us to quantify the impact of a particular CDN's clients on total HTTP traffic. Gnutella clients have almost no impact as consumers of HTTP bandwidth. In contrast, the Kazaa users are the worst offenders: the top 200 Kazaa clients are responsible for 20% of the total HTTP bytes downloaded. In comparison, the top 200 Web+Akamai clients are responsible for only 7% of total HTTP bytes. Further out, the top 400 Kazaa and Web clients are responsible for 27% and 10% of total HTTP traffic,

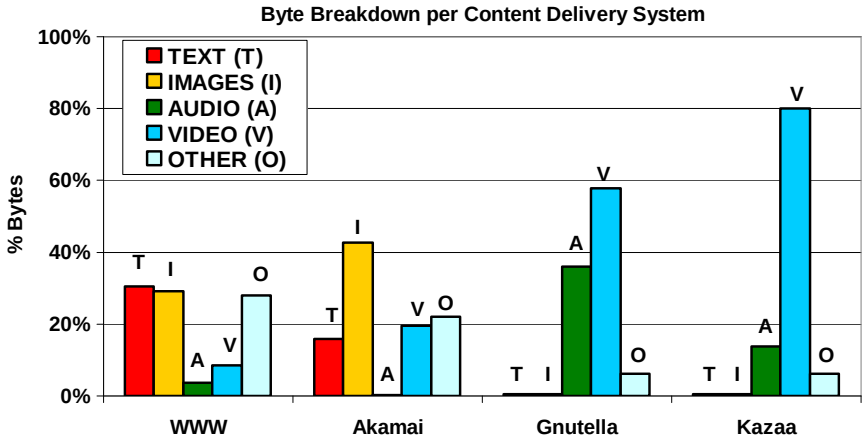

Figure 4.6: Downloaded bytes by object type: the number of bytes downloaded from each system, broken into content type.

respectively.

Given the bandwidth consumed by the Web and peer-to-peer delivery systems, it is interesting to examine the request rates that are creating that bandwidth. Figures 4.8a and 4.8b show the inbound and outbound request rates for Web+Akamai and Kazaa, respectively; notice the nearly two order-of-magnitude difference in the Y axis scales. For Kazaa, the outbound request rate peaks at 40 requests per second, dominating the inbound request rate of 23 requests per second. In contrast, the Web+Akamai inbound request rate peaks at 1100 requests per second, dominating the Web outbound request rate of just under 200 requests per second. At a high level, then, Kazaa has a request rate about two orders of magnitude lower than the Web, but median object size about three orders of magnitude higher than the Web. The result is that overall, Kazaa consumes more bandwidth. Similarly, Web's outbound bandwidth exceeds its inbound bandwidth, despite the opposite trend in request rate; this results from the difference in transfer size in the two directions. While inbound Web documents are largely HTML or images, outbound is dominated by application/octet-streams (possibly UW-supplied software, binary data, and video streams from its TV station or Web-broadcast technical talks).

A perhaps surprising (although now understandable) result of the disparity in Web+Akamai

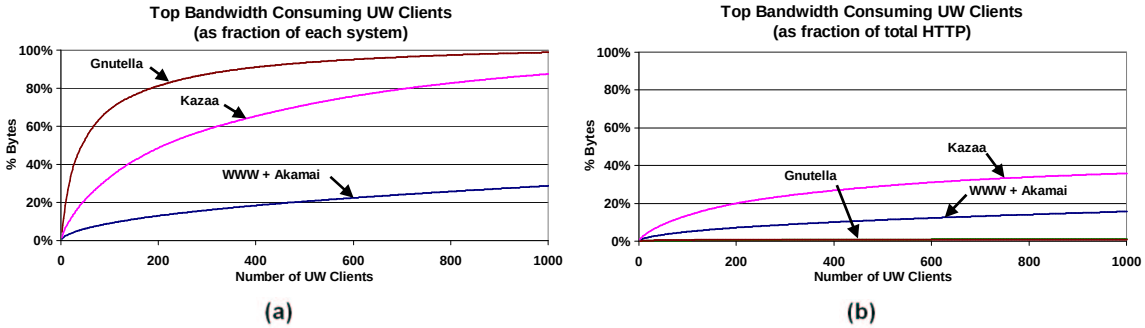

Figure 4.7: Top UW bandwidth consuming clients: a CDF of bytes downloaded by the top 1000 bandwidth-consuming UW clients (a) as a fraction of each system, and (b) as a fraction of the total HTTP traffic.

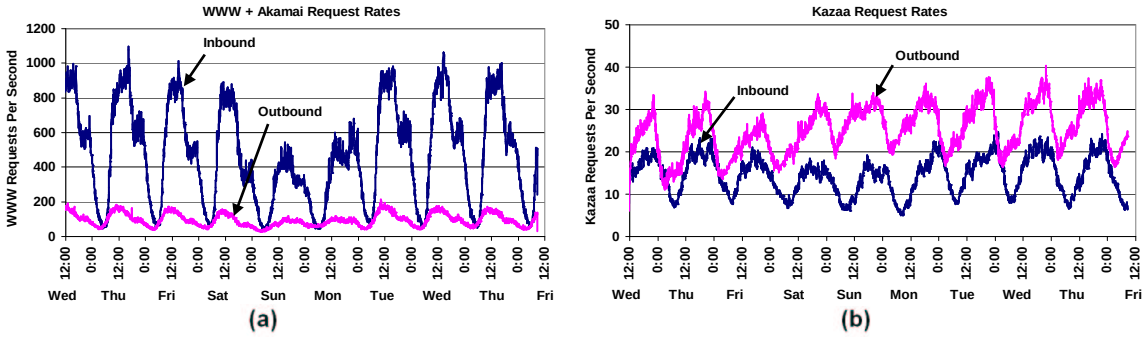

Figure 4.8: Request rates over time: inbound and outbound HTTP transaction rates for (a) the Web + Akamai, and (b) Kazaa.

and Kazaa object sizes and request rates is shown in Figure 4.9, which graphs the number of *concurrent* HTTP transactions active at a time for the two systems. Despite the order-of-magnitude request-rate advantage of Web over Kazaa, the number of simultaneous open Kazaa connections is about twice the number of simultaneous open Web+Akamai connections. While Kazaa generates only 23 requests per second, it is responsible for up to almost 1000 open requests at a time due to its long transfers. Compared to the Web requests, whose median duration is 120 ms, the median duration for Kazaa requests is 130 seconds – a 1000-fold increase that tracks the object size. This fact has important implications for the network infrastructure that must maintain those connections.

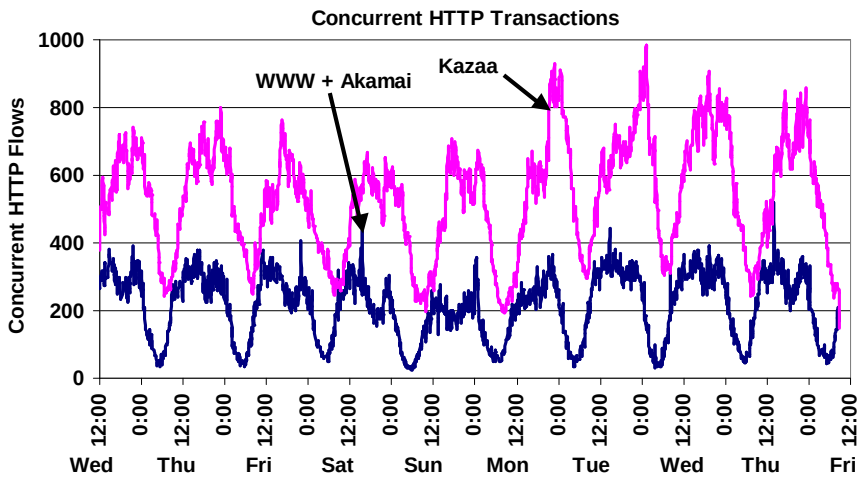

Figure 4.9: Concurrent HTTP transactions: concurrent HTTP transactions for UW Clients.

4.3.3 Servers

This section examines servers: the suppliers of objects and bytes. Figure 4.10a shows the CDF of bytes transferred by UW-internal servers to external clients. Gnutella has the smallest number of content-serving hosts, and all of the bytes are served by the first 10 of those servers. The Web curve is quite steep; in this case, the campus has several major servers that provide documents to the Web, and 80% of the Web traffic is supplied by 20 of the 9821 internal servers we identified. The Kazaa curve rises more slowly, with 80% of the Kazaa bytes being served by the top 334 of the 3888 Kazaa peers we found serving data. One would expect the Kazaa curve to be flatter; an explicit goal of peer-to-peer structures like Kazaa is to spread the load uniformly across the peers. We will look at this issue in more detail with external Kazaa servers.

Figure 4.10b shows the fraction of total HTTP bytes transferred (cumulative) from the top UW servers for the CDNs. The global impact of a small number of internal Kazaa peers is clearly seen on the graph. Again, a small number of Web servers do most of the work for Web, but this is a small part of the total HTTP outgoing traffic; 20 Web servers provide 20% of all HTTP bytes output, and the curve rises very slowly from there. However, from the Kazaa curve we see that 170 Kazaa peers are responsible for over 50% of all HTTP

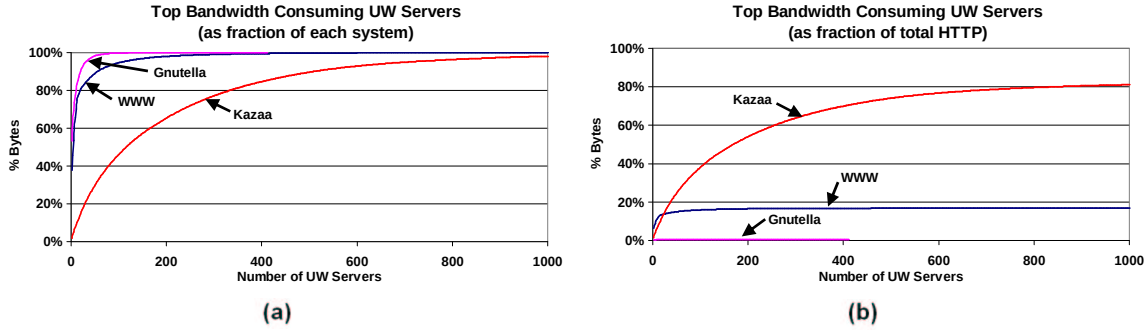

Figure 4.10: Top UW-internal bandwidth producing servers: a CDF of bytes produced by the top 1000 bandwidth-producing UW-internal servers (a) as a fraction of each system, and (b) as a fraction of the total HTTP traffic.

bytes transferred; the top 400 Kazaa peers are creating 70% of all outgoing HTTP traffic.

In the opposite direction (UW-external servers), Figures 4.11a and 4.11b show the cumulative and HTTP-fractional distributions for incoming bytes transferred, respectively, from the top 1,000 external servers to UW clients. The cumulative distribution (Figure 4.11a) shows the Web and Kazaa curves rising very slowly. The Web curve first rises steeply (for the most popular servers), then levels off, with 938 (out of 400,000) external servers supplying 50% of the total Web bytes to UW clients; this closely matches previous findings [138]. The Kazaa curve shows that 600 external Kazaa peers (out of 281,026) supply 26% of the Kazaa bytes to internal peers; this result, however, is somewhat unexpected. Web clients request data from *specific* servers by specifying a URL. A small number of Web servers are highly popular, and the popularity curve has a large tail (Zipf) of very unpopular servers. Peer-to-peer systems, though, are different by design. Clients request documents by name, not by server. Those documents may (and hopefully, will) exist on many peers. The goal of the peer-to-peer overlay structure is to broadly distribute work for both scalability and availability. In Kazaa, large files are downloaded by transferring different fragments of the file from different peers, to spread the load among multiple peers. Overall, one would expect the server load for Kazaa to be *much* more widely distributed among peers than for Web. From Figure 4.11a, this does not appear to be the case.

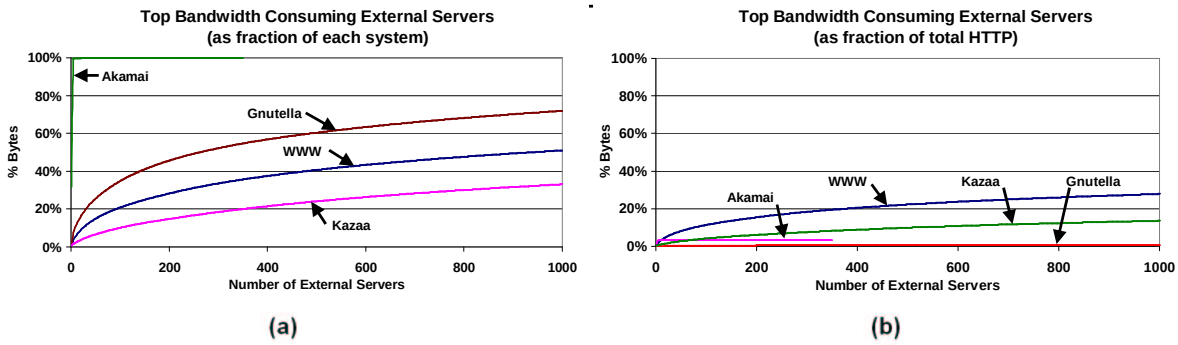

Figure 4.11: Top UW-external bandwidth producing servers: a CDF of bytes produced by the top 1000 bandwidth-producing UW-external servers (a) as a fraction of each system, and (b) as a fraction of the total HTTP traffic.

As a fraction of total HTTP bytes received by UW clients, Figure 4.11b shows that the top 500 external Kazaa peers supply 10% of the bytes to UW, while the top 500 Web servers supply 22% of the bytes. Gnutella and Akamai serve an insignificant percentage of the bytes delivered.

Participation in a P2P system is voluntary, and as a result, servers on P2P system are often less well-provisioned than in the Web or a CDN. In Figure 4.12, we show the response codes returned by external servers in each content delivery system. Figure 4.12a shows that for Akamai and the Web, approximately 70% of requests result in a successful transaction. However, for P2P systems, less than 20% of requests result in a successful transaction. Most P2P requests are met with a "service unavailable" response, suggesting that P2P servers are often saturated.

Figure 4.12b shows that nearly *all* HTTP bytes transferred in Web, Akamai and P2P systems are for useful content. Even though most P2P requests are rejected, the overhead of rejected requests is small compared to the amount of useful data transferred while downloading content.

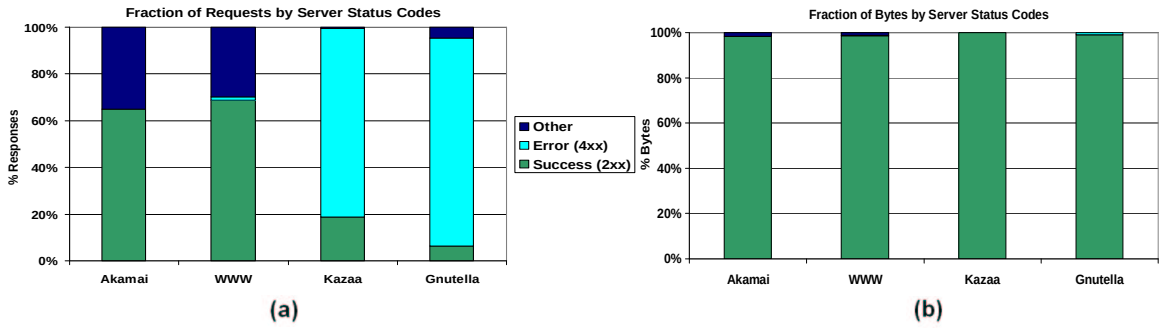

Figure 4.12: Server status codes: response codes returned by external servers; (a) shows the fraction of requests broken down by response code, (b) shows the fraction of bytes broken down by response code.

4.3.4 Scalability of Peer-to-Peer Systems

Our data raises serious questions about whether systems like Kazaa can scale in environments such as the University. We saw that the average Kazaa object is huge, and a small number of peers can consume an enormous amount of total network bandwidth in both directions. Over the period of our trace, an average Web client consumed 41.9 MB of bandwidth; in contrast, an average Kazaa peer consumed 3.6 GB of bandwidth. Therefore, the bandwidth cost of each Kazaa peer is approximately 90 times that of a Web client!

In Kazaa, the bandwidth cost of the entire system scales linearly with the number of participants. This implies that for our environment, adding another 450 Kazaa peers would be equivalent to *doubling* the entire campus Web client population, in terms of bandwidth impact. The enormous amount of bandwidth consumed by an average peer is so high that it seems questionable whether any organization providing bandwidth to a large client population can, in the long run, support a service with these characteristics.

4.3.5 The Potential Role of Caching in Peer-to-Peer Systems

Caching in the Web is well understood: caches have been shown to absorb bandwidth and reduce access latency [26, 29, 33, 46, 51]. In this section, we present an initial study of the effectiveness of caching in the context of the Kazaa peer-to-peer system.

Given the P2P traffic volume that we observed, the potential impact of caching in

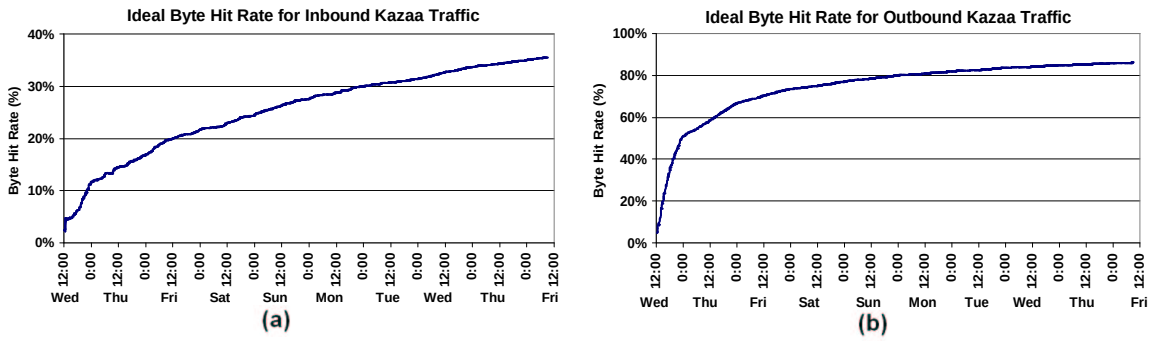

Figure 4.13: Ideal Kazaa cache byte hit rate: cache byte hit rate over time, for (a) inbound traffic (requests from UW clients) and (b) outbound traffic (requests from external clients).

P2P systems may exceed the benefits seen in the Web. In this section, we present an initial exploration of P2P caching. Our goal is not to solve (or even identify) all of the complexities of P2P caching, but rather to gain insight into how important a role caching may play.

To explore this question, we built an ideal (i.e., infinite capacity and no expiration) cache simulator for Kazaa P2P traffic. Since the average P2P object is three orders of magnitude larger than the average Web object, we evaluate the benefits of a P2P cache with respect to its byte hit rate, rather than its object hit rate. Because Kazaa peers can transfer partial fragments as well as entire objects, our cache stores items at the granularity of 32 KB *blocks*. For each Kazaa transfer, we identified and cached all complete 32 KB (aligned) blocks. Future requests for the same object may be partially satisfied from these cached blocks. Because of this, a single Kazaa transfer may involve multiple cache block hits and misses.

Figures 4.13a and 4.13b show the byte hit rate over time for inbound and outbound Kazaa traffic, respectively. The outbound hit rate (Figure 4.13b) increases as the cache warms, stabilizing at approximately 85%. This is a remarkably high hit rate – double that reported for Web traffic. A reverse P2P cache deployed in the University's ISP would result in a peak bandwidth savings of over 120 megabits per second!

The inbound hit rate grows more slowly over time, reaching only 35% by the end of the trace. It is clear that the simulated inbound cache has not fully warmed even for nine days worth of traffic. Accordingly, we will comment only on outbound traffic for the remainder

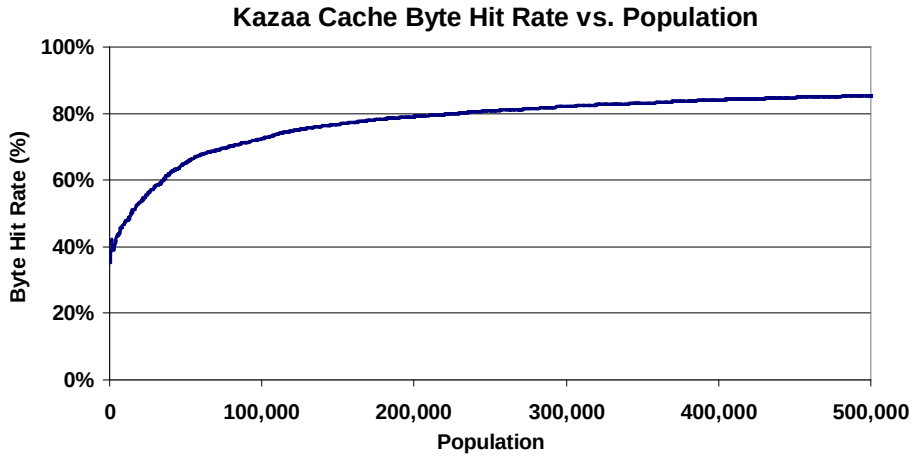

Figure 4.14: Kazaa cache byte hit rate vs. population: byte hit rate as a function of population size, for outbound traffic.

of this section.

We investigated the source of the high outbound hit rate by examining the 300 top bandwidth-consuming objects. These objects had an average size of 614 MB and a total size of 180 GB. Requests for these 300 objects consumed approximately 5.635 TB of traffic throughout the trace, which is 42% of the total bytes consumed by Kazaa outbound traffic. A conservative estimate suggests that a cache serving only these 300 objects would see a byte hit rate of more than 38% when presented with our entire trace. Thus, a small number of large objects are the largest contributors to the outbound P2P cache byte hit rate.

In Figure 4.14, we show the cache byte hit rate as a function of population size for outbound traffic. A population of 1,000 clients sees a hit rate of 40%; hit rate climbs slowly thereafter, until a population of 500,000 clients sees a hit rate of 85%. This indicates that a P2P cache would be effective for a small population, but even more effective for large populations.

Currently, many organizations are either filtering or rate-limiting P2P traffic to control bandwidth consumption. Based on our preliminary investigation in this section, we believe caching would have a large effect on a wide-scale P2P system, potentially reducing wide-area bandwidth demands dramatically.

4.4 Conclusions

This chapter examined Internet content delivery systems from the perspective of traffic flowing in and out of the University of Washington. To do this, we gathered a trace of all incoming and outgoing traffic over a period of nine days, separating out the HTTP traffic components due to standard Web traffic, Akamai-supplied content, Gnutella P2P file transfers, and Kazaa P2P file transfers. Our results confirm that a dramatic shift in Internet traffic and usage has occurred in only a few years. Specifically, for our environment we found that:

- Peer-to-peer traffic now accounts for the majority of HTTP bytes transferred, exceeding traffic due to Web accesses by nearly a factor of three. As a fraction of all TCP traffic, Kazaa alone accounts for 36.9% of bytes transferred, compared to 14.3% for Web documents.

- P2P documents are three orders of magnitude larger than Web objects, leading to a 1000-fold increase in transfer time. As a result, the number of concurrent P2P flows through the University is approximately twice the number of flows for our Web population, despite the extremely low request rate and small population of P2P systems.

- A small number of extremely large objects account for an enormous fraction of observed P2P traffic. For Kazaa transfers out of the University, the top 300 objects, with a total size of 180 GB, were responsible for 5.64 TB of the traffic – almost half of the total outbound Kazaa bandwidth.

- A small number of clients and servers are responsible for the majority of the traffic we saw in the P2P systems. The top 200 of 4,644 UW Kazaa clients accounted for 50% of downloaded Kazaa bytes. More surprisingly, only 600 UW-external peers (out of the 281,026 servers used) provided 26% of the bytes transferred into the University.

- Each P2P client creates a significant bandwidth load in *both* directions, with uploads exceeding downloads for Kazaa users. Our 4,644 Kazaa peers provided 13.573 TB of

data to 611,005 external peers, while requesting 1.78 TB of data from 281,026 peers. Overall, the bandwidth requirement of a single Kazaa peer is ninety-fold that of a single Web client.

Overall, these points indicate that despite the scalability-based design, the bandwidth demands of peer-to-peer systems such as Kazaa will likely prevent them from scaling further, at least within environments similar to the one measured. However, our initial analysis also shows that an organizational P2P proxy cache has the potential to significantly reduce P2P bandwidth requirements.

Chapter 5

INFRASTRUCTURE CHARACTERIZATION OF PEER-TO-PEER SYSTEMS

5.1 Introduction

This dissertation's major goal is to characterize the recently emerged Internet content delivery systems. While in the previous chapter, we examined their workloads, in this chapter, we characterize their infrastructure. To evaluate the architecture of a newly proposed peer-to-peer system [121, 116, 129, 134, 144], the characteristics of the peers that choose to participate in the system must be understood and taken into account. For example, if some peers in a file-sharing system have low-bandwidth, high-latency network connections to the Internet, the system must be careful to avoid delegating large or popular portions of the distributed index to those peers, for fear of overwhelming them and making that portion of the index unavailable to other peers. Similarly, the typical duration that peers choose to remain connected to the infrastructure has implications for the degree of redundancy necessary to keep data or index metadata highly available. In short, the system must take into account the suitability of a given peer for a specific task before explicitly or implicitly delegating that task to the peer.

Surprisingly, however, few of the architectures being developed are evaluated with respect to such considerations. We believe that this is, in part, due to a lack of information about the characteristics of hosts that choose to participate in peer-to-peer systems. In this chapter, we remedy this situation by performing a detailed measurement study of the two most popular peer-to-peer file sharing systems, namely Napster and Gnutella. The hosts that choose to participate in these systems are typically end-users' home or office machines, located at the "edge" of the Internet.

Our measurement study seeks to precisely characterize the population of end-user hosts

that participate in these two systems. This characterization includes the bottleneck bandwidths between these hosts and the Internet at large, IP-level latencies to send packets to these hosts, how often hosts connect and disconnect from the system, how many files hosts share and download, and correlations between these characteristics. Our measurements consist of detailed traces of these two systems gathered over long periods of time — four days for Napster and eight days for Gnutella respectively.

There are two main lessons to be learned from our measurement results. First, there is a significant amount of heterogeneity in both Gnutella and Napster; bandwidth, latency, availability, and the degree of sharing vary between three and five orders of magnitude across the peers in the system. This implies that any similar peer-to-peer system must be very careful about delegating responsibilities across peers. Second, peers tend to deliberately misreport information if there is an incentive to do so. Because effective delegation of responsibility depends on accurate information, this implies that future systems must have built-in incentives for peers to tell the truth, or systems must be able to directly measure or verify reported information.

The rest of the chapter is structured as follows. Section 5.2 discusses our measurement technology. Our measurement results are described in Section 5.3. Section 5.4 contains a brief discussion of our results and several recommendations for future file sharing peer-to-peer system designs. Finally, our conclusions are presented in Section 5.5.

5.2 Methodology

To collect our measurements of Napster and Gnutella, we periodically *crawled* each system in order to gather snapshots of the systems' populations. Our Napster trace captured four days of activity, from Sunday May 6th, 2001 through Wednesday May 9th, 2001 (during this time, Napster was at the peak of its popularity). We recorded a total of 509,538 Napster peers on 546,401 unique IP addresses. Our Gnutella trace spanned eight days (Sunday May 6th, 2001 through Monday May 14th, 2001) and captured 1,239,487 Gnutella peers on 1,180,205 unique IP-addresses.

5.2.1 Directly Measured Peer Characteristics

The kind of data we gathered for Napster users differs from the kind of data we gathered for Gnutella, primarily because of the differences in their protocol design and their implementation. For example, in Napster it is feasible to query the central servers regarding the number of uploads or downloads currently in progress from a given user, while it is impossible to get this data for Gnutella users. Similarly, it is possible to discover users who share zero files in Gnutella through the ping messages, whereas without privileged access to the Napster servers (which we did not have), it is impossible to detect the exact share of such users (we offer an estimate though).

5.2.2 Active Measurements

We also performed direct measurements of additional peers' properties. Our goal was to capture data that would enable us to reason about the fundamental characteristics of the peers (both as individuals and as a population) participating in any peer-to-peer file-sharing system. The data collected includes the distributions of bottleneck bandwidths and latencies between peers and our measurement infrastructure, the number of shared files per peer, the peers' distribution across DNS domains, and the peers' "lifetimes" (i.e., how frequently peers connect to the systems and how long they remain connected).

Unfortunately, in some cases, we could not reuse current network measurement tools due to their unscalability and slow speeds. Instead, we developed ways to incorporate existing measurement techniques into new tools that were more appropriate to the scale of our project. In this section, we describe our techniques and tools used to probe peers in order to measure their latencies, availabilities, and bandwidths. In order to distinguish the direction of probing traffic sent to a remote host, we will use "upstream" to refer to traffic from the remote host to the local host, and "downstream" to traffic from the local host to the remote host.

Latency Measurements

Given the list of peer IP-addresses obtained by the crawlers, we measured the round-trip latency between the peers and our measurement machines. For this, we used a simple tool that measures the RTT of a 40-byte TCP packet exchanged between a peer and our measurement host. Although we realize that the latency to any particular peer is dependent on the location of the host from which it is measured, we feel the distribution of latencies over the entire population of peers from a given host might be similar (but not identical) from different hosts, and hence, is of general interest [67].

Lifetime Measurements

To gather measurements of the availability (or "lifetime") characteristics of peers, we needed a tool that would periodically probe peers to detect when they were participating in the system. Every Napster and Gnutella peer connects to the system using a unique IP-address/port-number pair; to download a file, peers connect to each other using these pairs. There are therefore three possible states for any Napster and Gnutella peer:

1. **offline:** the peer is either not connected to the Internet or is not responding to TCP SYN packets because it is behind a firewall or NAT proxy.

2. **inactive:** the peer is connected to the Internet and is responding to TCP SYN packets, but it is disconnected from the peer-to-peer system and hence responds with TCP RST's.

3. **active:** the peer is actively participating in the peer-to-peer system.

We developed a simple tool (which we call *LF*) using Savage's "Sting" platform [124]. To detect the state of a host, *LF* sends a TCP SYN-packet to the peer and then waits for up to twenty seconds to receive any packets back. If no packet arrives, we mark the peer as offline. If we receive a TCP RST packet, we mark the peer as inactive. If we receive a TCP SYN/ACK, we label the host as active and send back a RST packet to terminate the connection. We chose to manipulate TCP packets directly rather than use OS socket calls

to achieve greater scalability; this enabled us to monitor the lifetimes of tens of thousands of hosts per workstation. Because we identify a host by its IP address, one limitation in the lifetime characterization of peers is our inability of distinguishing hosts sharing dynamic IP addresses (e.g., DHCP).

Bottleneck Bandwidth Measurements

Another characteristic of interest was the speed of peers' Internet connections. The central Napster servers can provide the Internet connection type of any peer as reported by the peer itself. However, as we will show later, a substantial percentage of the Napster peers (as high as 25%) choose not to report their connection types. Furthermore, there is a clear incentive for a peer to discourage other peers from downloading files by falsely reporting a slow connection. The same incentive to lie exists in Gnutella; in addition to this, in Gnutella, the Internet connection type is reported only as part of a successful response to a query, so peers that share no data or whose content does not match any queries never report it.

In consequence, we decided to actively probe the peers' bandwidths to detect the types of their Internet connections. For this, we developed and used our own tool (called *SProbe*). We presented the design and the evaluation of SProbe in Section 3.3.

A Summary of the Active Measurements

For the lifetime measurements, we monitored 17,125 Gnutella peers over a period of 60 hours and 7,000 Napster peers over a period of 25 hours. These peers were randomly selected from the set of all captured hosts. For each Gnutella peer, we determined its status (offline, inactive or active) once every seven minutes, and for each Napster peer, once every two minutes.

For Gnutella, we attempted to measure bottleneck bandwidths and latencies to a random set of 595,974 unique peers (i.e., unique IP-address/port-number pairs). We were successful in gathering downstream bottleneck bandwidth measurements to 223,552 of these peers, the remainder of which were either offline or had significant cross-traffic. We measured

upstream bottleneck bandwidths from 16,252 of the peers. Finally, we were able to measure latency to 339,502 peers. For Napster, we attempted to measure downstream bottleneck bandwidths to 4,079 unique peers. We successfully measured 2,049 peers.

In several cases, our active measurements were regarded as intrusive by several monitored systems. Unfortunately, e-mail complaints received by the computing staff at the University of Washington forced us to prematurely terminate our crawls, hence the lower number of monitored Napster hosts.

5.2.3 Limitations of the Methodology

An ideal characterization of peers should include a characterization of the participating hosts, even when they share or re-use IP addresses. Unfortunately, our crawlers only discover pairs of IP addresses and port numbers, and therefore we make each unique pair represent a single participant. As a result, our findings are susceptible to IP aliasing and IP reuse problems, such as NAT and DHCP. Most related work on peer-to-peer measurements suffer from this methodological limitation; we know of a single previous study [21] that exploited the properties of a different protocol to measure peers' availabilities, finding similar results to ours. Resolving the IP aliasing and IP reuse issues when characterizing hosts from IP-level network measurements is still an open research problem. Section 6.2.3 will revisit this issue further.

5.3 Detailed Characteristics of Peers

Our measurement results are organized according to a number of basic questions addressing the peers' capabilities and behavior. We pose four basic questions that attempt to understand how uniform peers are in practice and whether they behave altruistically:

1. How many peers fit the high-bandwidth, low-latency profile of a server?

2. How many peers fit the high availability profile of a server?

3. How many peers fit the no-files-to-share, always-downloading profile of a client?

4. How much peers are willing to cooperate in peer-to-peer file-sharing systems?

The rest of this section answers these questions in detail.

5.3.1 How Many Peers Fit the High-Bandwidth, Low-Latency Profile of a Server?

One particularly relevant characteristic of peer-to-peer file sharing systems is the percentage of peers in the system having *server*-like characteristics. More specifically, we are interested in understanding what percentage of the participating peers exhibit server-like characteristics with respect to their bandwidths and latencies. Peers worthy of being servers must have high-bandwidth Internet connections, they should remain highly available, and the latency of access to the peers should generally be low. If there is a high degree of heterogeneity amongst the peers, a well-designed system should pay careful attention to delegating routing and content-serving responsibilities, favoring server-like peers.

Downstream and Upstream Measured Bottleneck Link Bandwidths

To fit the profile of a high-bandwidth server, a peer must have a high upstream bottleneck link bandwidth, since this value determines the rate at which a server can serve content. On the left, Figure 5.1 presents cumulative distribution functions (CDFs) of upstream and downstream bottleneck bandwidths for Gnutella peers. From this graph, we see that 22% of peers have upstream bottleneck bandwidths of at most 100Kbps; these low upstream bandwidths make these peers inadequate to play the roles of servers. Not only are these peers unsuitable to provide content and data, they are particularly susceptible to being swamped by a relatively small number of connections. On the other hand, 8% of peers have upstream bottleneck bandwidths of at least 100Mbps; these peers' bandwidths make them adequate to play the roles of servers.

The left graph in Figure 5.1 reveals asymmetry in the upstream and downstream bottleneck bandwidths of Gnutella peers. On average, a peer tends to have higher downstream than upstream bottleneck bandwidth; this is not surprising, because a large fraction of peers depend on asymmetric links such as ADSL, cable modems or regular modems using the V.90 protocol [1]. Although this asymmetry is beneficial to peers that download content,

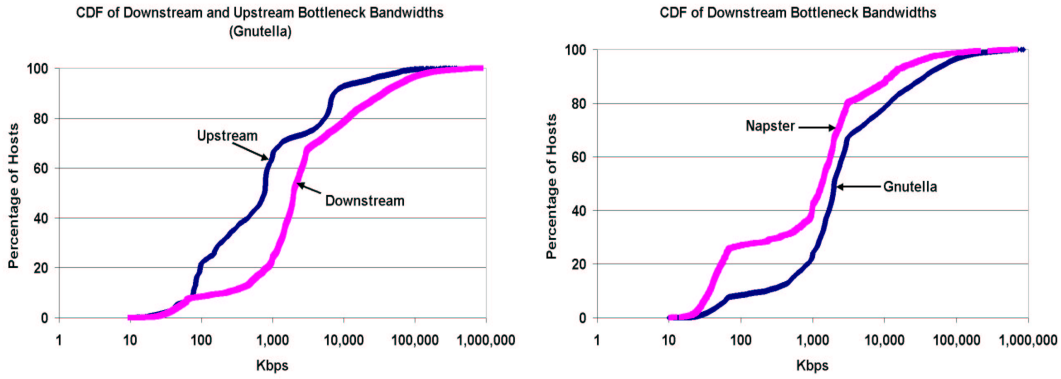

Figure 5.1: Left: CDFs of upstream and downstream bottleneck bandwidths for Gnutella peers; Right: CDFs of downstream bottleneck bandwidths for Napster and Gnutella peers.

it is both undesirable and detrimental to peers that serve content: in theory, the download capacity of the system exceeds its upload capacity. We observed a similar asymmetry in Napster.

The right graph in Figure 5.1 presents CDFs of downstream bottleneck bandwidths for Napster and Gnutella peers. As this graph illustrates, the percentage of Napster users connected with modems (of 64Kbps or less) is about 25%, while the percentage of Gnutella users with similar connectivity is as low as 8%.

At the same time, 50% of the users in Napster and 60% of the users in Gnutella use broadband connections (Cable, DSL, T1 or T3). Furthermore, only about 20% of the users in Napster and 30% of the users in Gnutella have very high bandwidth connections (at least 3Mbps). Overall, Gnutella users on average tend to have higher downstream bottleneck bandwidths than Napster users. Based on our experience, we attribute this difference to two factors: (1) the current flooding-based Gnutella protocol is too high of a burden on low bandwidth connections, discouraging them from participating, and (2) although unverifiable, there is a widespread belief that Gnutella is more popular to technically-savvy users, who tend to have faster Internet connections.

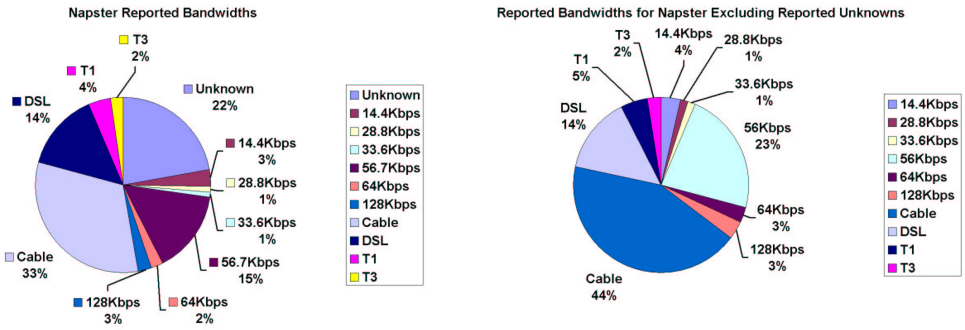

Figure 5.2: Left: Reported bandwidths for Napster peers; Right: Reported bandwidths for Napster peers, excluding peers that reported "unknown".

Reported Bandwidths for Napster Peers

In contrast to Figure 5.1 that reports *measured* peer bandwidths, Figure 5.2 illustrates the breakdown of Napster peers with respect to their voluntarily *reported* bandwidths; the bandwidth that is reported is selected by the user during the installation of the Napster client software. (Peers that report "Unknown" bandwidth have been excluded in the right graph.)

As Figure 5.2 shows, a significant percent of the Napster users (22%) report "Unknown." These users are either unaware of their connection bandwidths, or they have no incentive to accurately report their true bandwidth. Indeed, knowing a peer's connection speed is more valuable to others rather than to the peer itself; a peer that reports high bandwidth is more likely to receive download requests from other peers, consuming network resources. Thus, users have an incentive to misreport their Internet connection speeds. A well-designed system therefore must either directly measure the bandwidths rather than relying on a user's input, or create the right incentives for the users to report accurate information to the system. We will revisit this issue in Section 5.3.4.

Finally, both Figures 5.1 and 5.2 confirm that the most popular forms of Internet access for Napster and Gnutella peers are cable modems and DSLs (bottleneck bandwidths between 1Mbps and 3.5Mbps).

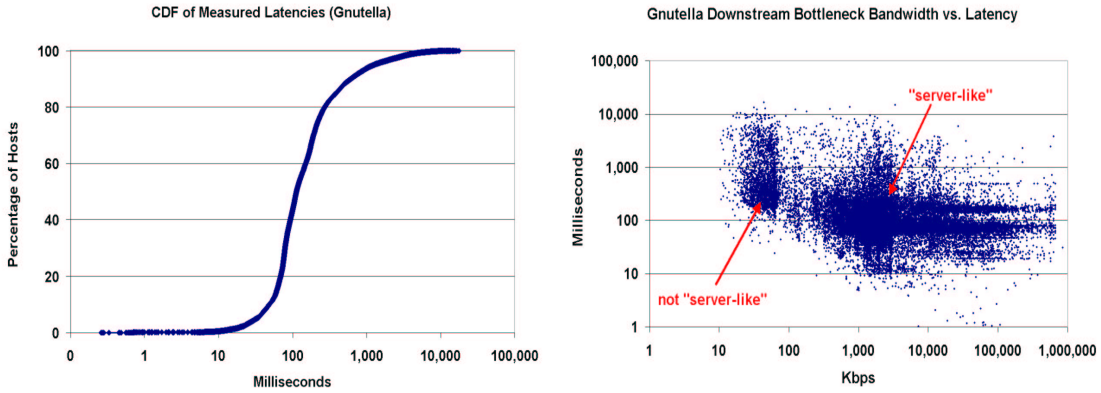

Figure 5.3: Left: Measured latencies to Gnutella peers; Right: Correlation between Gnutella peers' downstream bottleneck bandwidth and latency.

Measured Latencies for Gnutella Peers

Figure 5.3 (left) shows a CDF of the measured latencies from our measurement nodes to those Gnutella peers that form several consecutive snapshots of the Gnutella overlay . Approximately 20% of the peers have latencies of at least 280ms, whereas another 20% have latencies of at most 70ms: the closest 20% of the peers are four times *closer* than the furthest 20%. From this, we can deduce that in a peer-to-peer system where peer connections are forged in an unstructured, ad-hoc way, a substantial fraction of the connections will suffer from high latency.

On the right, Figure 5.3 shows the correlation between downstream bottleneck bandwidth and the latency to individual Gnutella peers (on a log-log scale). This graph illustrates the presence of two clusters; 10% of all peers form a smaller one situated at (20-60Kbps, 100-1,000ms) and 70% of all peers form a larger one at over (1,000Kbps, 60-300ms). These clusters correspond to the set of modems and broadband connections, respectively. The negatively sloped lower-bound evident in the low-bandwidth region of the graph corresponds to the non-negligible transmission delay of our measurement packets through the low-bandwidth links.

An interesting artifact evident in this graph is the presence of two pronounced horizontal bands. These bands correspond to peers situated on the North American East Coast and

in Europe, respectively. Although the latencies presented in this graph are relative to our location (Seattle, WA, USA), these results can be extended to conclude that there are three large classes of latencies that a peer interacts with: (1) latencies to peers on the same part of the continent, (2) latencies to peers on the opposite part of a continent and (3) latencies to trans-oceanic peers. As Figure 5.3 shows, the bandwidths of the peers fluctuate significantly within each of these three latency classes.

5.3.2 How Many Peers Fit the High-Availability Profile of a Server?

Server worthiness is characterized not only by high-bandwidth and low-latency network connectivity, but also by the availability of the server. If peers tend to be unavailable frequently, this will have significant implications about the degree of replication necessary to ensure that content is consistently accessible on this system [22].

On the left, Figure 5.4 shows the distribution of *uptimes* of peers for both Gnutella and Napster. Uptime is measured as the percentage of time that the peer is available and responding to traffic. The "Internet host uptime" curves represent the uptime as measured at the IP-level, i.e., peers that are in the inactive **or** active states, as defined in Section 5.2.2. The "Gnutella/Napster host uptime" curves represent the uptime of peers in the active state, and therefore responding to application-level requests. For all curves, we have eliminated peers that had 0% uptime (peers that were never up throughout our lifetime experiment).

The IP-level uptime characteristics of peers are quite similar for both systems; this implies that the set of peers participating in either Napster or Gnutella are homogeneous with respect to their IP-level uptime. Only 20% of the peers in each system had an IP-level uptime of 93% or more.

In contrast, the application-level uptime characteristics of peers differ noticeably between Gnutella and Napster. On average, Napster peers tend to participate in the system more often than Gnutella peers. One might hastily conclude that since more users participate in Napster, more content is available and therefore peers have, on average, longer uptimes. However, this data can also be used to draw an opposite conclusion: more content means that users can find the files of interest faster, which results in shorter uptimes. We believe

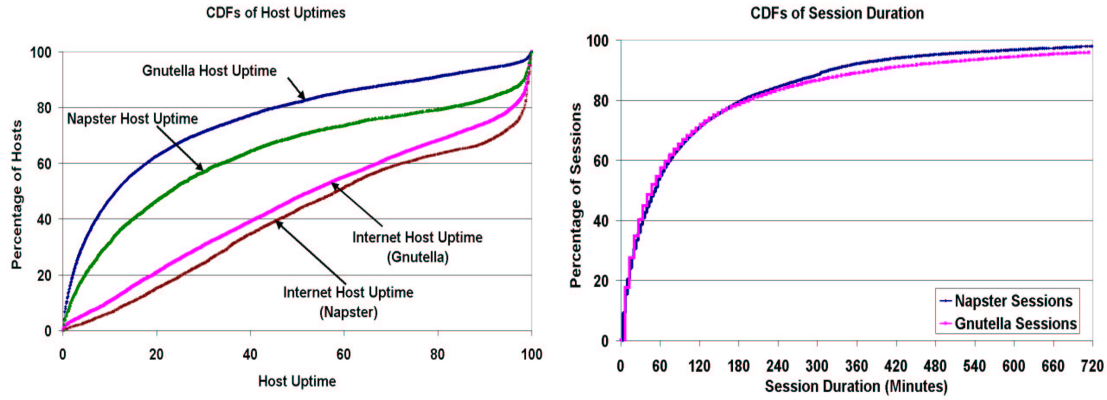

Figure 5.4: Left: IP-level uptime of peers ("Internet Host Uptime"), and application-level uptime of peers ("Gnutella/Napster Host Uptime") in both Napster and Gnutella, as measured by the percentage of time the peers are reachable; Right: The distribution of Napster/Gnutella session durations.

that this difference is primarily a factor of the design of the client software; Napster's software has several features (such as a built-in chat client and an MP3 player) that cause users to run it for longer periods of time.

Another significant difference can be observed in the tail of the application-level distributions: the best 20% of Napster peers have an uptime of 83% or more, while the best 20% of Gnutella peers have an uptime of 45% or more. Our (unproven) hypothesis is that Napster is, in general, a higher quality and more useful service, and that this has a large influence on the uptime of its peers relative to Gnutella.

On the right, Figure 5.4 presents the CDF of Napster and Gnutella session durations that are *less than twelve hours*. The graph is limited to twelve hours because of the nature of our analysis method; we used the create-based method [120], in which we divided the captured traces into two halves. The reported durations are only for sessions that started in the first half, and finished in either the first or second half. This method provides accurate information about the distribution of session durations for session that are shorter than half of our trace, but it cannot provide any information at all about sessions that are longer than half our trace. As Figure 5.4 illustrates, 50% of the peers never remain online for more than one hour. Since we observed a roughly constant number of peers participating in Napster

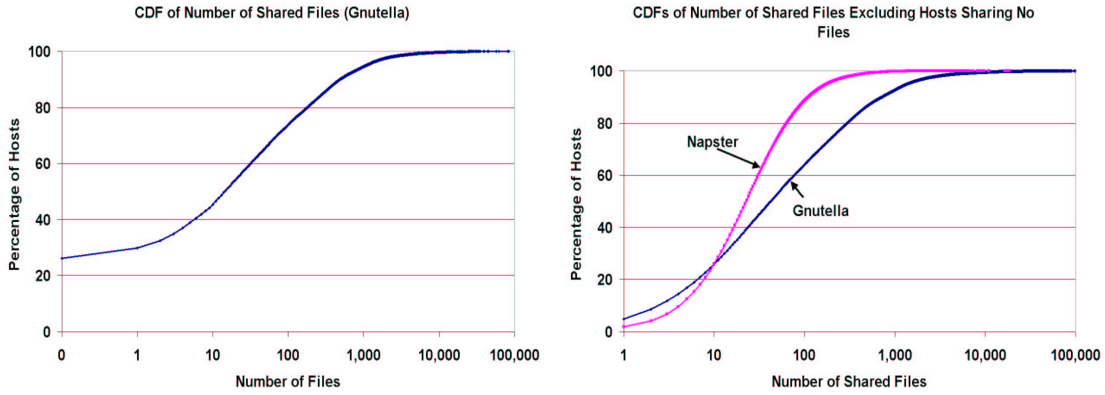

Figure 5.5: Left: The number of shared files for Gnutella peers; Right: The number of shared files for Napster and Gnutella peers (peers with no files to share are excluded).

and Gnutella, we conclude that over the course of one hour, half of the participants leave these systems and are replaced by another half.

There is an obvious similarity between Napster and Gnutella; for both, most sessions are quite short—the median session duration is approximately 60 minutes. This is not surprising, as it corresponds to the time it typically takes for a user to download a small number of music files from the service.

5.3.3 How Many Peers Fit the No-Files-to-Share, Always-Downloading Profile of a Client?

In addition to understanding the percentage of server-like Napster and Gnutella peers, it is equally important to determine the number of client-like peers. One aspect of client-like behavior is that no data is shared in the system. Previous studies refer to these peers as *free-riders* [3] in the system. Another variable of interest is the number of downloads and uploads a participating peer is performing at any given time. A peer with a high number of downloads fits the profile of a client, whereas a peer with a high number of uploads fits the profile of a server.

Number of Shared Files in Napster and Gnutella

In Figure 5.5, the left graph shows the distribution of shared files across Gnutella peers, and the right graph shows this distribution for both Napster and Gnutella, but with peers

sharing no files eliminated from the graph. (As previously mentioned, we could not capture any information about peers with no files from Napster.)

From the left graph, we see that as high as 25% of the Gnutella clients do not share any files. This fact illustrates that in spite of claims that *every peer is both a server and a client*, Gnutella has an inherently large percentage of *free-riders* [3].

Figure 5.5 also reveals that there is a lot of variation in the number of files shared by each peer. On the left, 75% of Gnutella clients share 100 files or less, whereas only 7% of the clients share more than 1,000 files. A simple calculation reveals that these 7% of users together offer more files than all of the other users combined. The right graph shows that Napster peers are slightly more consistent and offer less variation in the number of shared files than Gnutella peers. Nonetheless, about 40-60% of the peers share only 5-20% of the shared files, which indicates that there is a large amount of variation in the amount of content shared in Napster as well.

Number of Downloads and Uploads in Napster

In Figure 5.6, the left graph shows the distribution of concurrent downloads by Napster peers classified by the peer's reported bandwidth, and the right graph shows a similar curve for the number of concurrent uploads. Because these graphs were obtained by capturing snapshots of the download and upload activity using our crawler, these distributions are biased towards capturing low-bandwidth peers, since downloads take longer through low-bandwidth connections.

Nonetheless, this graph shows interesting correlations between peers' reported bandwidths and their concurrent downloads and uploads. First, there are 20% more zero-download high-speed peers than zero-download low-speed peers. We see two possible explanations: either higher-bandwidth peers tend to download less often, or they spend less time downloading because they have higher connection speeds. Second, the correlation between bandwidths and the downloads is reversed relative to bandwidths and uploads (the percentage of zero-upload peers is higher for modems than for cable modems).

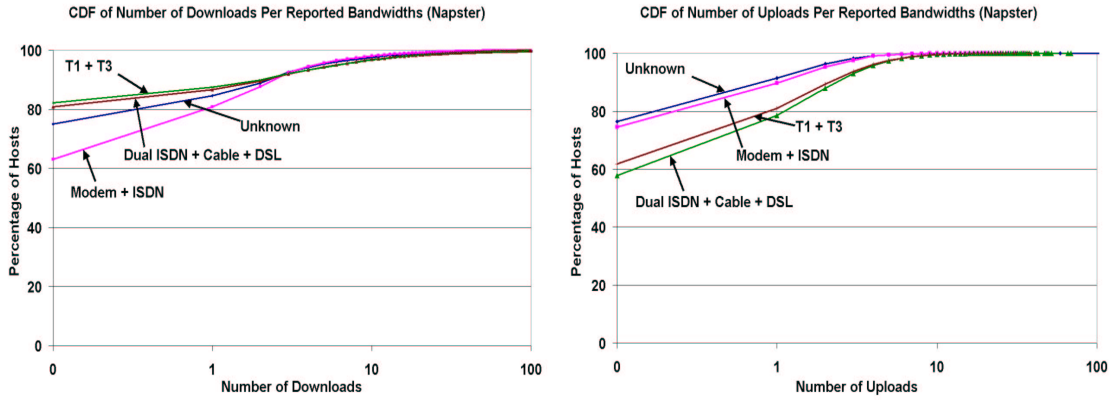

Figure 5.6: Left: The number of downloads by Napster users, grouped by their reported bandwidths; Right: The number of uploads by Napster users, grouped by their reported bandwidths.

Correlation between the Number of Downloads, Uploads, and Shared Files

On the left, Figure 5.7 shows the percentage of downloads, the percentage of the peer population, the percentage of uploads and the percentage of shared files, grouped according to the reported bandwidth from Napster peers. The number of shared files seems to be uniformly distributed across the population: the percentage of peers in each bandwidth class is roughly the same as the percentage of files shared by that bandwidth class.

However, the relative number of downloads and uploads varies significantly across the bandwidth classes. For example, although 56Kbps modems constitute only 15% of the Napster peers, they account for 24% of the downloads. Similarly, cable modems constitute 32% of the peers, but they account for 46% of the uploads. The skew in the number of uploads is attributed to users selecting high-bandwidth peers from which to download content. The skew in the number of downloads, however, seems to be more representative of the natural tendency of low-bandwidth peers to be free-riders.

On the right, Figure 5.7 shows the distribution of the number of shared files by Napster peers classified by the peer's reported bandwidth. A peer's bandwidth has little effect on the number of shared files. In Napster, half of the modem participants share 18 files or less, whereas half of the users with higher Internet connection speeds share 28 files or less. Unfortunately, since our methodology cannot capture peers sharing no files, it is possible

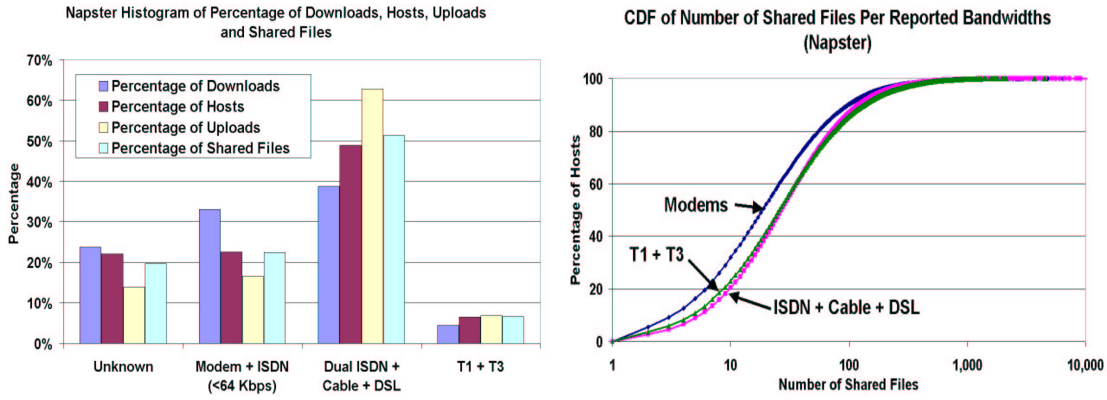

Figure 5.7: Left: Percentage of downloads, peers, uploads and shared files, grouped by reported bandwidths (in Napster); Right: The number of shared files by Napster users, grouped by their reported bandwidths.

that a different degree of correlation between a peer's bandwidth and its number of shared files might exist.

Figure 5.8 shows the distribution of concurrent downloads (on the left) and uploads (on the right) by Napster users classified by their number of shared files. On average, peers with fewer shared files perform fewer downloads and uploads. Having little data to share directly impacts the ability of a peer to contribute a large number of uploads to the system. However, as Figure 5.8 illustrates, peers with less shared data seem to be less interested, on average, in downloading from the system. Since, on the right, Figure 5.7 does not indicate any substantial lack of bandwidth available to these peers, we conclude that, although they are able to download as much data as everyone else, these participants prefer to rather download fewer files. Finally, the contrast in the slopes of the downloads and uploads distribution curves for the peers sharing similar numbers of files demonstrates that peers contributing more data have a higher number of uploads, on average.

5.3.4 How Much Are Peers Willing to Cooperate in a P2P File-Sharing System?

The peer-to-peer model fundamentally depends on the concept of cooperation. How willing peers are to cooperate is of vital importance to the viability of these systems. Devising

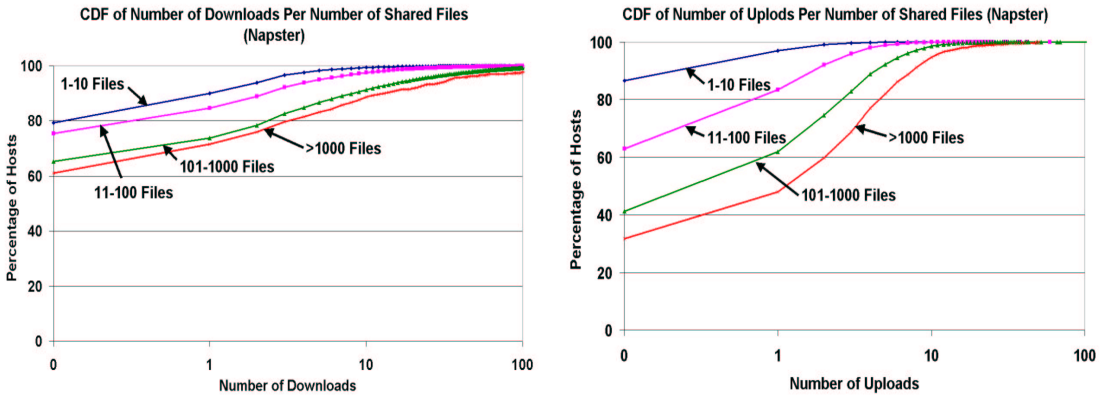

Figure 5.8: Left: The number of downloads by Napster users, grouped by their number of shared files; Right: The number of uploads by Napster users, grouped by their number of shared files.

an experiment to quantify a peer's willingness to cooperate is of course very difficult; as a first-order approximation, we measured the extent to which peers deliberately misreport their bandwidths.

The user interfaces for file querying are quite similar in Napster and Gnutella. When peers holding a requested file are discovered, their information, such as IP address, DNS name, network latency, and reported bandwidth (possibly including the value "Unknown"), are returned and presented to the user requesting the file. This user then selects one of these target peers and, as a result, it initiates a direct download of the requested file. The user downloading the file is likely to select a peer that has high bandwidth and low latency. In consequence, participating peers have an incentive to deliberately misreport lower bandwidths to the system, in order to discourage others from initiating downloads from them.

On the left, Figure 5.9 shows the distribution of measured bandwidths for Napster peers, classified by their reported bandwidth. Note that as much as 30% of the users that report their bandwidth as 64 Kbps or less actually have a significantly greater bandwidth. In Napster (and any similar system), a peer has an incentive to report a smaller bandwidth than the real value, in order to discourage others from initiating downloads and consuming the peer's available bandwidth. Similarly, we expect most users with high bandwidths

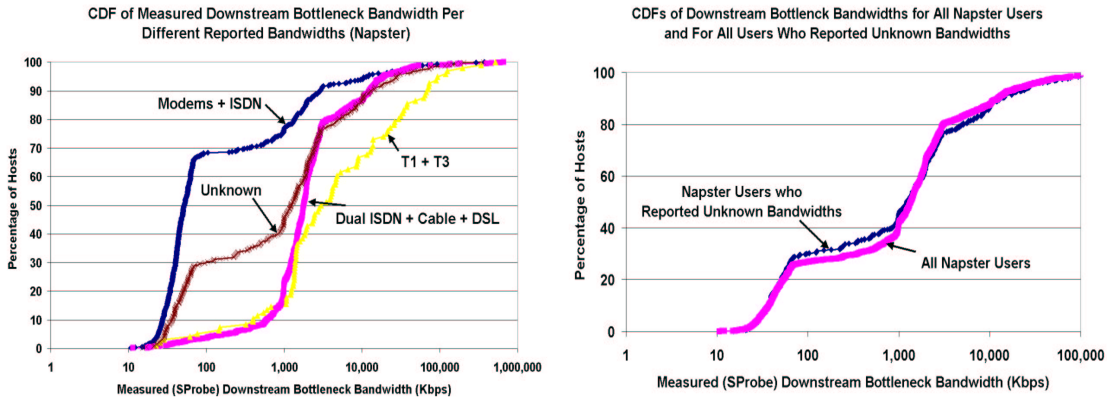

Figure 5.9: Left: Measured downstream bottleneck bandwidths for peers, grouped by their reported bandwidths; Right: CDFs of measured downstream bottleneck bandwidths for those peers reporting unknown bandwidths along with all Napster users.

to rarely misreport their actual bandwidths. Indeed, Figure 5.9 confirms that only than 10% of the users reporting high bandwidth (T1 and T3) in reality have significantly lower bandwidth. Because more high bandwidth peers "misreport" their bandwidth than low bandwidth peers, it is unlikely that these are the result of ignorance or misconfiguration.

In addition to showing that many peers are uncooperative in Napster, this graph serves to validate the accuracy of our bottleneck bandwidth estimation technique. There is an extremely strong correlation between measured bandwidth and reported bandwidth, across all reported classes.

The right graph in Figure 5.9 shows the distribution of measured downstream bottleneck bandwidth of Napster peers reporting unknown bandwidths. Overlain on top of this distribution, we have shown the distribution of measured bandwidths of all Napster peers, regardless of their reported bandwidth. The similarity between the two curves implies that peers reporting unknown bandwidths are uniformly distributed across the population.

5.4 Recommendations to Peer-To-Peer System Designers

There has been a flurry of proposed distributed algorithms for routing and location in a P2P system. Most of these protocols and proposals make the implicit assumption that the

delegation of responsibility across nodes in the overlay should be uniform, and hence that all nodes will tend to participate and contribute equally in information exchange and routing. In contrast, our measurements indicate that the set of hosts participating in the Napster and Gnutella systems is heterogeneous with respect to many characteristics: Internet connection speeds, latencies, lifetimes, shared data. In fact, the magnitudes of these characteristics vary between three and five orders of magnitude across the peers! Therefore, P2P systems should delegate different degrees of responsibility to different hosts, based on the hosts' physical characteristics and the degree of trust or reliability.

Another frequent implicit assumption in these systems is that peers tend to be willing to cooperate. By definition, to participate in a P2P system, a peer must obey the protocol associated with the system. In addition, most users tend to download pre-created software clients to participate in these systems (as opposed to authoring their own). These software packages typically ask users to specify configuration parameters (such as Internet connection speed) that will be reported to other peers. As we have shown, many of these parameters are in practice either left unspecified or deliberately misreported. Instead of relying on reported characteristics, we believe that a robust system should attempt to directly measure the characteristics of peers in the system.

Another myth in P2P file-sharing systems is that all peers behave equally, both contributing resources and consuming them. Our measurements indicate that this is not true: client-like and server-like behavior can clearly be identified in the population. As we have shown, approximately 26% of Gnutella users shared no data; these users are clearly participating to download data and *not* to share. Similarly, in Napster we observed that on average 60-80% of the users share 80-100% of the files, implying that 20-40% of users share little or no files.

The experiments and the data presented in this chapter indicate that many of the characteristics that Napster and Gnutella P2P systems in practice match the characteristics of the classic server-client model. Thus, we believe that future robust P2P protocols should account for the hosts' heterogeneity, relying on self-inspection and adaptation to exploit the differences in the hosts' characteristics, behavior, and incentives.

5.5 Conclusions

In this chapter, we presented a measurement study performed over the population of peers that choose to participate in the Gnutella and Napster peer-to-peer file sharing systems. Our measurements captured the bottleneck bandwidth, latency, availability, and file sharing patterns of these peers. Our results invalidate a number of key assumptions made by current peer-to-peer system designs. Specifically, we found that:

- There is a significant amount of heterogeneity in both Gnutella and Napster. We found that bandwidths, latency, availability, and the degree of sharing vary between three and five orders of magnitude across the peers in the system.

- There is clear evidence of client-like and server-like behavior in a significant fraction of the systems' populations. The bottom 25% of Gnutella peers do not share any files. These peers cannot act as servers. On the other hand, the top 7% of Napster peers together offer more files than all the other users combined.

- Peers tend to deliberately misreport information if there is an incentive to do so. One third of Napster peers that report their bandwidths as 64Kbps or less actually have a significantly greater bandwidth. In Napster, a peer has an incentive to report a smaller bandwidth than the real value, in order to discourage others from initiating downloads and consuming the peer's bandwidth.

Overall, these points indicate the presence of a gap between the peer-to-peer premises and the reality of peers' capabilities and behavior. Even though these systems were designed with a symmetry of roles and responsibilities, in practice, peers offer and derive different amounts of services from the system. New peer-to-peer designs need to be deliberate and careful about delegating responsibilities across peers. Similarly, future systems need to build and enforce incentive mechanisms for peers to contribute and participate.

Chapter 6

RELATED WORK

Measurement tools are lenses through which an Internet system can be viewed and monitored. These tools' drawbacks are capable of introducing distortions of the system's view seen through these lenses. Once the tool is in place and its "distortions" are appropriately handled, researchers can ask and answer questions about the system's workload and infrastructure. In this chapter, we survey the related work on characterizing Internet content delivery systems. In Section 6.1, we examine previous tools (i.e., the "lenses") for measuring large-scale Internet systems and their limitations (i.e., the "distortions"). In Section 6.2, we survey related studies of Internet content delivery systems, focusing on their workload and infrastructure questions and answers.

6.1 Tools and Techniques for Measuring Large-Scale Internet Systems

A rigorous characterization of an Internet content delivery system includes a characterization of its workload and a characterization of its infrastructure. Characterizing the workload of a large-scale Internet system is different from characterizing its infrastructure; the tools and the techniques are different in each case. In the remainder of this section, we survey previous research on designing, developing, and implementing tools and techniques for: (1) measuring a large-scale Internet system's workload; and (2) measuring a large-scale Internet system's infrastructure.

6.1.1 Measuring Workloads

There are three general classes of techniques for measuring the workloads of large-scale Internet systems: (1) network tracing; (2) active probing; and (3) software instrumentation.

Network Tracing

Network tracing is a common method for collecting Internet systems' workloads [52, 41, 125, 109, 100, 141, 92, 63, 53, 98, 132, 113, 54, 127, 135]. Network tracing has several advantages as a technique for measuring large-scale Internet systems. First, because of the passive nature of network tracing, deploying a network tracing system in a production environment carries relatively little risk. Second, the network's performance is not impacted by the deployment of a network tracing system. Third, a network tracing system can collect information about the full population being monitored. Finally, network tracing systems are not dependent on users' willingness to install specialized software on their systems. The results in this dissertation are based on logs collected using network tracing. This further demonstrates the feasibility of network tracing as a measurement technique for capturing the behavior of large-scale Internet systems.

The primary disadvantage of network tracing is that measurements are collected at a fixed number of points in the entire system. As a result, the question haunting most of the measurement studies based on network tracing is whether these studies are representative for the entire Internet-scale system. In addition to this intrinsic drawback of network tracing, several practical problems make this technique hard to use. First, the tracing system needs to reconstruct all higher-level network protocols of interest from low-level packet traces. Ewing et al. [52] list FTP's use of separate communication ports for control and data as one of several complications in gathering their data. Wooster et al. [141] report HTTP reconstruction from TCP packet traces as a major effort. In some cases, the reconstruction of higher level semantics is next to impossible. For example, Plonka reports that identifying Napster traffic in packet traces is difficult [113]. Mena and Heidermann report that they cannot distinguish between users and IP addresses, because of the heavy use of proxies in their traces [98]. Second, the network tracing implementation needs to be robust against any traffic patterns, including erroneous ones. A network traffic system needs to handle duplicate packets and packet reordering, together with malformed packets [109]. Third, the implementation must be highly scalable and efficient to ensure that no packets are being dropped due to high network loads [52, 100]. A network tracing system has no means

of requesting the retransmission of lost data. Finally, packet traces are typically large in size. Processing them often involves the implementation of fast, efficient algorithms for computing various statistics [100], or the use of a well-tuned database.

An early system for Web tracing is Httpdump [141]. Httpdump is a layer of software on top of tcpdump, identifying HTTP headers. Httpdump reconstructs neither HTTP traffic nor TCP traffic, but rather creates a log of requests and responses. Httpdump suffers from serious scalability problems; its packet loss rate is high when monitoring the traffic generated by 21 Web clients only.

BLT [54] is a passive network monitoring system developed specifically for studying the behavior of HTTP traffic. The primary goal of BLT is to support continuous online network monitoring. The BLT system has been used in different locations, running for weeks at a time with no more than 0.3% packet loss when monitoring network links of up to 100Mbps. At a high level, BLT's functionality closely monitored the network monitoring systems used in this dissertation. However, there are some key differences. Our system needs to scale to network speeds an order of magnitude higher than BLT. Another difference is that our system takes a more aggressive approach to users' privacy. This prevents us from logging raw packets directly to the disk. Finally, the implementation strategies of the two systems are different.

IPSE is a user-level packet filter that recreates a TCP network conversation on-the-fly as it collects TCP segments. The original goal of IPSE was to scan network conversations in order to expose security vulnerabilities (e.g., passwords in plain text) [62]. Gribble and Brewer [63] extended IPSE to parse and collect HTTP requests over a 10Mbps Ethernet network connecting a pool of home dialup modems. Their traces are 18 days long and they report no packet loss. Unlike our traces, the traces collected with IPSE are anonymized offline; the HTTP requests captured are directly written to the disk. Another difference between IPSE and our tracing system is the anonymization scheme; IPSE's anonymization scheme does not preserve the locality of IP addresses (e.g., it does not hash each IP address octet separately).

A large piece of the network tracing infrastructure used in this dissertation is based on a previous network tracing system developed by Wolman et al. [139, 138, 34, 137]. This

network tracing system has been used to collect Web logs [139, 138] and logs of multimedia streaming traffic [34] at the University of Washington. There are two key differences between Wolman et al.'s system and ours. First, our system identifies and reconstructs peer-to-peer traffic (e.g., Gnutella and Kazaa) and content delivery networks traffic (e.g., Akamai), in addition to Web traffic. Second, our system scales to an order of magnitude higher along several different dimensions: network speeds (we trace at speeds of 1Gbps rather than 100Mbps), amount of data (we collect terabytes of data rather than gigabytes), and robustness (we continuously trace for several consecutive months rather than several consecutive days).

In a recent study, Smith et al. [127] use tcpdump to record the TCP headers of packets traversing a 1Gbps network link. This is the link that connects the campus of the University of North Carolina to the rest of the Internet. Smith et al. perform several small adjustments to tcpdump to improve its scalability. They report a maximum packet loss rate of 0.02%. The methodology used by Smith et al. has several properties that make it unsuitable for our goals. First, our privacy concerns prevent us from directly logging packet headers. Second, Smith et al. collect traces that are very short in length. Their longest trace is one hour in length and only collects half of the total network traffic (i.e., only the inbound traffic). Third, the packet loss rate reported is high (1 packet out of 5,000) given the short duration of their traces. Our packet loss rate is at least two order of magnitude lower. Finally, the reported packet loss rate ignores packets lost in the network card driver. Instead, the reported loss rate is the one measured by the bpf filter installed in the kernel. Packets lost by the network card driver cannot be detected by the kernel.

Active Probing

Another technique used to study the performance of wide-area Internet systems is active probing. Active probing is the generation of network traffic whose purpose is to measure a system's behavior. Several previous studies use active probes to characterize the World Wide Web [142, 140, 2, 82].

While active probing is a viable technique for measuring large-scale Internet systems, a

relatively small number of studies use this technique in their measurements. The reason is that active probing in an inadequate technique for characterizing systems' workloads. It is hard to determine which objects are being transferred or the object popularity distribution through actively probing the network. As a result, few Web studies use active probing.

The main benefit of active probing is that it allows for full control over the experiment. For instance, the selection of the population to be monitored could be done based on certain attributes that the researcher wants to measure. Another advantage of active probing is the ease of implementation. Unlike network tracing, the amount of probing traffic generated can be tuned to match the amount of processing available to conduct the experiment.

There are several disadvantages of active probing. The traffic being measured is synthetic. In some cases, it is hard to replicate the shape of the real traffic needed to be studied [82]. Another disadvantage is that for certain types of measurements, active probing might consume a lot of resources [2].

More recently, it has become increasingly hard to measure Internet systems using active probing. Unfortunately, active proving is regarded as an intrusive network operation by several monitoring systems. One of the studies described in this dissertation uses active probing. During our experiments, several e-mail complaints were received by the computing staff at the University of Washington. These actions forced us to prematurely suspend our measurements.

Software Instrumentation

Software instrumentation is a common technique for collecting Internet systems' workloads. Many studies use software instrumentation to collect and analyze World Wide Web workloads [112, 60, 125, 39, 32, 20, 85, 95, 100, 92, 11, 6, 84, 130, 26, 27, 9, 43, 101, 10, 77, 76] and content delivery networks workloads [73, 83]. The principal advantage of this technique is the ease of instrumentation. Software instrumentation has been used to instrument client software, proxy software, and server software. For clients, instrumentation is typically done by modifying and extending the client software, such as a Web browser. For proxies and servers, instrumentation is already present in the software in form of proxy and server log-

ging. Instrumenting proxies and servers is only a simple extension to the existing logging mechanism.

However, multiple disadvantages prevent software instrumentation from being a viable technique for measuring Internet content delivery systems, especially peer-to-peer systems. Instrumenting peer software presents all the drawbacks associated with instrumenting client software. Most peer-to-peer client software is closed-source. There are numerous privacy concerns associated with instrumenting peer software. It is hard to deploy instrumented software across a large number of peers participating in these systems. All these drawbacks prevented us (and others) from measuring the workloads of peer-to-peer systems using software instrumentation.

Instrumenting Client Software Several previous studies instrumented browsers to collect logs of Web activity [39, 32, 20, 130, 84]. There are two main advantages of client software instrumentation. First, instrumenting client software allows for direct measurement of user-perceived performance of the system. Second, the client software has access to information that can be the most relevant for certain types of studies (e.g., usability studies [32, 130]). This information might not be reconstructible through any other measurement technique, such as network tracing or active probing.

Instrumenting client software has many disadvantages. Early studies modified the Mosaic browser to intercept all the user's input and Web traffic [39, 32, 130]. While this approach worked on early, open-source browsers like Mosaic, subsequent browsers (e.g., Netscape and Microsoft Internet Explorer) are closed-source and not easily modifiable. A different approach to browser instrumentation is based on browser plugins. A browser plugin is a small program that is invoked upon a browser event. Examples of browser events are: browser startup, send HTTP request, retrieve HTTP object, and browser shutdown. Browser plugins are supported by several modern and popular browsers, including Netscape and Microsoft Internet Explorer.

There are numerous privacy concerns associated with client software instrumentation. Researchers must ensure that their methodology adheres to the users' privacy concerns [32]. Alexa (now a subsidiary of Amazon.com) [5] is a recent example of client software instru-

mentation. Alexa has instrumented many Web browsers through a downloadable plugin (i.e., a browser toolbar) that reports surfing activity to a central logging site. The data collected and the collection method are neither described in detail nor used for research purposes. As a result, many Web users consider Alexa a form of spyware [122].

A final disadvantage of client software instrumentation relates to deployment. It is hard to deploy instrumented software across a large set of clients. Most of the studies using client software instrumentation are small in scope; they measure at most hundreds of users [130].

Instrumenting Proxy Software Several previous studies analyzed logs collected by proxy software [60, 51, 9, 43, 101, 77, 76]. Most Web proxies already include support for logging in the source code. Furthermore, the source code for commonly used proxies such as Squid is freely available. Modifying it and extending it to collect Web data is relatively straightforward.

Several problems exist with proxy software instrumentation. First, proxies might serve stale data, corrupting the correctness of the log being gathered [77]. Second, proxies fail to see the requests being served from the browsers' caches [77]. Third, the researchers conducting the experiments might be restricted from accessing the proxy, making it hard to modify and extend the proxy software. This can lead to problems; for example the time resolution used by the proxy software might be too coarse-grained to be useful [76]. Another example is that most proxy software only collect a small amount of information about their workloads [43]. Finally, the proxy software instrumentation must ensure that any use of the collected traces does not compromise users' privacy [51]. Privacy concerns prohibit exposing information that identifies users with the objects being traced.

Instrumenting Server Software Many previous studies analyzed logs collected by server software [112, 125, 100, 85, 95, 11, 92, 6, 27, 10, 107]. Most Web servers already include support for logging in the source code, and it is just a matter of editing some configuration files to enable the logging. However, as with proxy software, most server software does not provide all the information of interest. For example, Web server software typically records only the number of bytes transferred to service a request, rather than the whole

object size [11]. As a result, many studies extended the logging mechanism to address these issues [100, 11, 92, 27]. Many of the shortcomings of conventional server log formats (and proxy log formats) are documented in Davison [43] and Cáceres et al. [27]. There are also known cases when Web server software instrumentation introduced unbearable overhead and caused performance problems [38]. Finally, another serious problem with server logs is their large size. At least one measurement study had to distill its collected server logs to deal with the magnitude of the size of the data [10].

General Techniques for Software Instrumentation Although software instrumentation is a common technique for measuring Internet systems, little work has been done in developing general tools and techniques or creating a set of "rules-of-thumb" and principles on how to instrument software for large-scale Internet systems' workload collection.

An initial piece of work is the Simultaneous Proxy Evaluation (SPE) architecture proposed by Davison [42]. The SPE system mirrors Web clients' requests across a set of proxies running simultaneously, but with different proxy configurations. The goal of SPE is to investigate the performance of different proxy configurations from the perspective of Web clients. Although a full design of SPE exists [44], its current software implementation cannot be used in practice [45].

Rajamony and Elnozahy present a framework for measuring the user-perceived response time of clients accessing a particular Web service [115]. Their system uses client-side JavaScript 1.1 [57]. Upon starting, a user is presented with a special Web page containing JavaScript code that constantly runs in the background collecting user behavior information. There are several limitations with this approach [115]. First, this scheme is limited to measuring Internet systems that use JavaScript. Second, this scheme only works as long as a user's browser supports JavaScript. In particular, this scheme does not work with older browser versions or browsers that disabled JavaScript. Third, the information is collected only as long as the user browses instrumented Web pages. For example, pages loaded by directly entering a URL into a browser's location bar cannot be monitored.

Finally, Koletsou and Voelker present the design and the implementation of Medusa, a Web proxy designed to collect information about the user-perceived performance of the

World Wide Web [80]. Medusa incorporates the mirroring design described by SPE [42]. In addition, Medusa can be used to simulate a workload by actively replaying the workload of collected client requests. Finally, Medusa can validate the correctness of an experimental proxy's behavior. For example, it can verify whether the proxy returned a document older than the document served by the origin server. One drawback of Medusa is that it only supports HTTP 1.0.

6.1.2 Measuring Infrastructure

A rigorous characterization of the behavior of a content delivery system is not complete without an understanding of the system and network properties of its infrastructure. These properties include network bandwidths, latencies, machine availabilities, and packet losses of network paths. In the case of overlay networks, an infrastructure characterization must include mapping the topological structure of the overlay. Understanding the layout of the system (i.e., its topological map) is important as it can affect the system's properties.

This dissertation makes two contributions related to the development of tools and techniques for measuring large-scale Internet systems' infrastructure. The first contribution is the design of a crawler for Gnutella, an unstructured peer-to-peer network. The second contribution is the development of a bottleneck bandwidth measurement tool. We therefore limit our discussion to related work relevant to our contributions.

Crawling Peer-to-Peer Systems

Crawling peer-to-peer systems to discover their topology is an important ingredient of a full characterization of these systems' infrastructure. There is very little published work on describing how to overcome the challenges of crawling large-scale peer-to-peer systems. Some of these challenges are due to specific protocol or software implementations, and they are not fundamental to these systems' architectures. For example, some protocols are proprietary or encrypted; crawling these systems is hard, if not impossible. Other protocols implement optimizations that make it hard to discover an accurate topology.

Ripeanu et al. describes a technique to crawl Gnutella, an unstructured peer-to-peer

system [118, 117]. Their study focuses on understanding graph properties of the Gnutella overlay, such as the distribution of node degrees, or the average number of alternate paths between two arbitrary peers. There are several key differences between their technique and the design of our crawler. First, Ripeanu relies on generic implementations of Gnutella clients, instrumented to log topological information. Theses clients are then strategically placed throughout the entire Gnutella overlay. These clients map the topology of the network in their immediate vicinity only. The collected fragments are then individually analyzed to extract the graph properties of the overlay fragments. This technique cannot capture the complete topology of the Gnutella overlay. Second, the crawling sessions are much longer than ours: a couple of hours versus a couple of minutes. A short crawling session ensures that the topology discovered is a close snapshot of the true state of the network. Given a large peer churn rate, crawling for more than several minutes will collapse several different snapshots of the system into one single crawl. Third, unlike us, their technique discards discovered peers to which their crawler cannot connect. In Gnutella, each peer individually selects a threshold on the maximum number of connections allowed to other peers. Once a peer reaches this threshold, the peer refuses to be contacted by other peers. This means that their technique discards all peers that have reached their connection thresholds. Finally, some of their collected data suffers from the "invisible peers problem." In March 2001, Bearshare, arguably the most popular version of Gnutella software, implemented an optimization aimed at reducing the number of messages broadcasted throughout the network. Once a peer reaches the maximum number of connections to other peers, it refuses to answer to any packets received other than queries. These peers are therefore invisible to Ripeanu's crawler that uses Gnutella PING packets to discover the network.

Chu et al. [35] also implemented a crawler for Gnutella. However, their study focused on collecting information about the file workloads and the node availabilities, rather than on discovering topological information. As a result, the goal of their crawler is to discover as many peers as possible ignoring connectivity information. For example, the crawler uses queries to discover peers rather than Gnutella PING messages. The crawler also discards all discovered peers that do not use a specific Gnutella client program. (Bearshare or SwapNut). These two clients were the only software programs that allowed their users'

directories to be remotely retrieved. Several reasons make this methodology unable to collect a complete snapshot of the Gnutella overlay. First, this methodology only returns peers with content, ignoring free-riders. Second, the crawler focuses on collecting information about as many peers as possible, collapsing different snapshots into one single crawl. Third, users not running the two specific Gnutella software programs are ignored. Finally, users not answering to the crawler's set of queries are ignored.

While crawling unstructured peer-to-peer systems, such as Gnutella and Kazaa, is difficult in practice, crawling structured peer-to-peer systems, such as DHTs, is easier and more accurate. Bhagwan et al. implemented a crawler for Overnet [21]. Unlike Gnutella, Overnet is a distributed hash table, a structured peer-to-peer system. Overnet is based on Kademlia [96]. As a result, this crawler takes advantage of the DHT's lookup abstraction to map the overlay. The crawler relies on creating random IDs and then searching the network for the presence of the peer with that randomly generated ID. This technique also determines the status of a peer – whether it is available and participating in the network or not. Mapping the entire overlay takes approximately four hours of crawling. Like Ripeanu, this strategy suffers from the problem of collapsing multiple snapshots into one crawl. There are two main advantages of this approach. First, this technique is not susceptible to DHCP and NAT problems [21]. Dynamic address assignment protocols, such as DHCP, can easily cause the same host to be counted multiple times and thereby underestimate host availability. Similarly, the growth in the use of NAT boxes can affect the correctness of the TCP/IP addressing protocols. All crawlers for unstructured peer-to-peer systems are susceptible to DHCP and NAT problems. Second, it reduces the intrusiveness of the network measurement.

Tools and Techniques for Measuring Bottleneck Bandwidth on the Internet

Estimating bottleneck bandwidth is not a new area of research [71, 78, 23, 31, 110, 88, 48]. Broadly speaking, all existing techniques fall into two categories: the "one-packet" technique and the "packet pair" technique. Several tools that use these techniques have been developed and studied in the past [86, 72, 93, 49, 30, 47]. In this section, we present a qualitative

analysis of key properties of these tools. First, we introduce some terminology.

The endpoints of a network path are referred to as *endhosts*. A *cooperative endhost* is an endhost on which measurement software has been deployed, and an *uncooperative endhost* is one on which measurement software cannot be deployed. However, an uncooperative endhost can still unwittingly participate in a measurement; for example, content served by a Web server can be used as input traffic to a measurement tool. A measurement is done in a *cooperative environment* when the measured network path runs between two cooperative endhosts. In contrast, when one endhost is uncooperative, the measurement is performed in a *uncooperative environment*.

The One-Packet Technique The one-packet technique, first described in [14], relies on the observation that a packet's traversal time across a single link can be approximated by the sum of three variables: queuing delay, transmission delay (signal modulation and serialization), and signal propagation delay across the transmission medium. In the absence of cross traffic, queuing delay can be assumed to be negligible; by taking a large number of measurements, it is likely that at least one will be obtained without queuing delay.

A second assumption is that transmission delay varies linearly with packet size and can be approximated by the ratio of packet size to link bandwidth. A final assumption of the model is that latency remains constant for different packet sizes. Therefore, sending a large number of packets of different sizes ensures that the minimum value of their transmission times will approximate a line whose slope is the inverse of link bandwidth (Figure 6.1). This technique produces an estimate at each hop along the measured network path; the bottleneck bandwidth of the entire path is the minimum value of these link bandwidths. For more information, see [88, 14, 50].

Strengths and Weaknesses of the One-Packet Technique and Tools Several tools are based on the one-packet model, including *Pathchar* [72], *pchar* [93], and *clink* [49]. An attractive property of the one-packet model is its ability to measure the bottleneck bandwidth in an uncooperative environment. Also, this technique produces a picture of the entire network path measured by estimating the link bandwidths at each hop along the

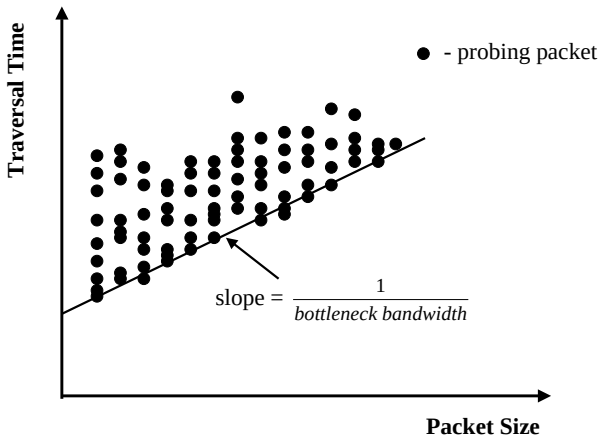

Figure 6.1: One-Packet Model: ideally, the minimum value of the traversal times for each packet size approximates a line whose slope is the inverse of the bottleneck bandwidth.

path. Finally, because they send a large amount of probing packets and record the minimum traversal times, the one-packet tools are less susceptible to cross traffic.

Unfortunately, the one-packet model has several important limitations. First, the current tools implementing this technique rely on a functional ICMP implementation at each router along the measured network path. Second, this technique measures bandwidth in a single direction, from the local to the remote endhost. Third, because the estimates of each link rely on the estimates of the previous links, the errors accumulate and amplify with each measured link. (Previous studies have presented evidence that these tools are inaccurate and slow [88, 48].) Finally, the large number of probing packets generated by the tools adds considerable stress to the network path. A quick calculation reveals that for a single Ethernet hop with a latency of 1ms, the average bandwidth consumed is 6.02Mbps [87]. There are several negative implications due to this network flood: the tools are unscalable, slow and inflexible to bandwidth changes.

The Packet Pair Technique The packet pair technique was introduced by [71] and [78]. Two consecutive packets are queued one after the other at the bottleneck queue of a network path, hence the name *packet pair*. After traversing the bottleneck link, the time dispersion

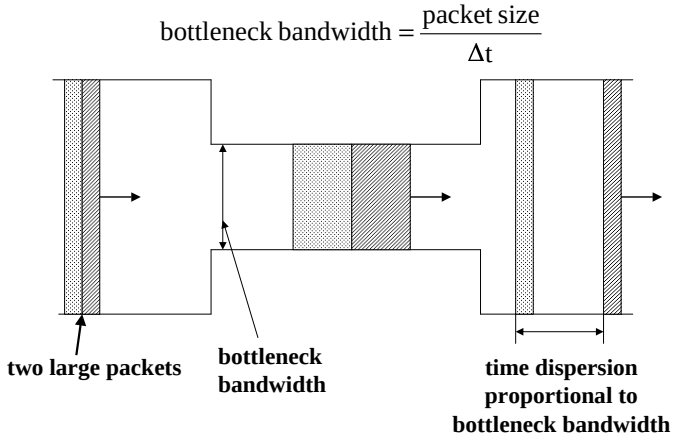

Figure 6.2: Packet Pair Model: after the bottleneck link, the time dispersion between the packets is proportional to the bottleneck bandwidth.

between the two packets is linearly related to the bottleneck link bandwidth, assuming same-sized packets. For this technique to work, the packets must be large enough so that queuing occurs at the bottleneck link. After the bottleneck link, the temporal spacing between the packets will remain constant, as the packets have the same size and no other link has a bandwidth lower than the bottleneck link. Figure 6.2 illustrates the technique. For more details on this technique, see [110, 88].

Four assumptions must be valid for this technique to work in practice [88]. First, the packets' size must be large enough so that they queue at the bottleneck link but no later link. Second, the two probe packets must be sent consecutively, with no space between them. Third, the bottleneck router must use FIFO queuing. Finally, transmission delay must be proportional to packet size.

Strengths and Weaknesses of the Packet Pair Technique and Tools Unlike the one-packet model, the packet pair technique only produces an estimate of the bottleneck link bandwidth, as opposed to the bandwidth of each link along the network path. Since the bottleneck link estimate does not rely on previous links' bandwidth estimates, the packet pair technique is typically more accurate.

Like the one-packet technique, the accuracy of packet pair measurements can degrade due to cross traffic interference. A rogue packet queued at the bottleneck link between the packet pair invalidates the measurement. Current packet pair tools deal with cross traffic by sending a large amount of probe packet pairs and gathering a large number of measurements. Assuming that cross traffic interference does not dominate the measurements, statistical algorithms are used to extract the average, free-of-cross-traffic case. Different statistical algorithms have been proposed to extract accurate estimates from sets of gathered data points [31, 89, 48].

Several implementations of the packet pair technique exist: *bprobe/cprobe* [30], *Nettimer* [86], *pathrate* [47]. *Nettimer* is a passive tool that relies on monitoring traffic between the endhosts, whereas *bprobe/cprobe* and *pathrate* actively send probe packets. Estimates can be based on different protocols: *bprobe/cprobe* uses ICMP, *Nettimer* uses TCP and *pathrate* uses UDP (*pathrate* also exchanges control information between the endhosts over a TCP connection, however the probes are UDP packets). *bprobe/cprobe* is designed for uncooperative environments, whereas *pathrate* assumes a cooperative environment. *Nettimer* has different estimating techniques for both cooperative and uncooperative environments.

Since ICMP packets are filtered, dropped or even answered by the routers along the network path [124], tools that use the ICMP protocol to produce bandwidth estimates (*bprobe/cprobe*) can be highly inaccurate.

Although faster than their one-packet counterparts, certain implementations are still too slow to be used in practice (*bprobe/cprobe*, *pathrate*), e.g. on a 100Mbps, 2-hop Ethernet, *pathrate* spends 73 seconds to produce an estimate with a 5% relative accuracy.

Out of all the current packet pair tools, *Nettimer* is the only one studied in a controlled environment using a network simulator [87]. The authors acknowledge the limitations of their simulation results, due to the inability of the network simulator to reproduce characteristics of the Internet traffic [111]. Nevertheless, the simulations show that for long TCP sessions (e.g. 30 seconds) with consistent amounts of data being exchanged, *Nettimer* is accurate.

Unfortunately, the same properties can also have a negative impact on the tool's usability and practicality. In certain cases, bandwidth needs to be estimated on network paths

where no data is being exchanged, or the shape of the existing traffic violates packet pair assumptions (e.g. large packet pairs). As an example, the traffic of a *telnet* application generates very few, if any, packet pairs. Moreover, the packets are usually very small in size – 40-50 bytes. In this case, *Nettimer* has no choice but to produce inaccurate and inconsistent estimates.

In static environments, changes in bottleneck bandwidth occur very infrequently. Nevertheless, *Nettimer* display a continuous range of different estimates, based on the properties of the traffic exchanged and the degree of cross traffic in the estimates. It is unclear which of the displayed estimates is the most accurate and can be used as a single estimate.

6.2 Characterizing Internet Content Delivery Systems

Understanding and evaluating the performance of an Internet content delivery system is difficult in practice. Important metrics, such as user-perceived performance and server throughput, depend on a multitude of interactions among numerous protocols and software components. Furthermore, these metrics are tied among themselves. For example, a higher rate of unsuccessful requests could cause a higher rate of duplicate requests in the system. Evaluating system performance in a systematic and controlled manner is hard, because it requires a decoupling of the workload from the underlying system.

In this section, we investigate previous research on characterizing Internet content delivery systems. Our focus is on briefly examining the invariants found across many studies and characterizations. The question we are attempting to answer is: *What are the common properties of Internet content delivery systems and what are their implications?*

6.2.1 Characterizing the World Wide Web

In this section, we present previous research on characterizing Web workloads. We focus on briefly examining the invariants found across a large number of Web studies. We examine Web workloads from the perspective of Web object properties, the cacheability of Web workloads, and Web server properties. We do not discuss studies of Web clients, as most of these have focused on characterizing the behavior of humans when browsing the Web, such

as the average number of clicks per Web page or the interarrival times of Web clicks.

Characterizing Web Objects

Web workloads include objects with a variety of different types: text, images, archives, executable code, audio, and video data. However, the content types of the overwhelming majority of objects served are text and images [11, 63, 46, 10, 139, 137]. The remaining content types account for a relatively small portion of the resources. There are two implications of the predominance of text and image data in Web workloads. First, text is amenable to compression. The prevalence of text in Web workloads prompted early research on compressing Web traffic. Second, text and image data are highly cacheable types of content. A large fraction of Web traffic is therefore likely to be cacheable.

The average Web object size is relatively small, although a small proportion of objects are large. Text and image objects tend to be smaller than other types of content, such as audio and video data. The average size of an HTML file is around 4 to 8 KB, whereas the average size of an image is around 14KB, though the exact numbers differ from study to study [11, 92, 63, 10].

Although most objects are small in size, some objects are very large. Many workloads exhibit a common invariant – while most of the objects are small in size, they account for a relatively small portion of the bandwidth consumed. Instead, a small number of very large objects account for a large portion of the bandwidth consumed [11, 63, 10].

The emergence of new applications can have a sudden and profound influence on the distribution of content types. Acharya and Smith [2] found that the availability of multimedia content in Web workloads is increasing. A higher portion of Web workloads is dedicated to transporting multimedia data. This implies that the size of Web objects could change over time. A different trend that can affect Web workloads is the penetration of high-speed Internet connections (broadband). Users with high-bandwidth Internet connections are likely to download larger resources [9].

A large number of empirical studies [60, 39, 11, 7, 139, 138, 123] of Web workloads show that the distribution of Web object popularities is Zipf-like. At the same time, it can be shown theoretically that workloads with object popularities following a Zipf-like distribution exhibit excellent temporal locality [25]. This implies that Web workloads have good cache hit rates.

Early empirical studies report low rates of uncacheable objects [63, 94]. Later studies [26, 55] dispute these findings, pointing out that HTTP/1.0 resources accompanied by cookies should be considered uncacheable. Taking cookies into account leads to high rates of uncacheable Web objects. In their study of Web traffic at University of Washington and at Microsoft, Wolman et al. [139] use the Squid [128] proxy cache implementation to determine whether objects are cacheable. This cacheability filter is less strict than the one using cookies to determine whether objects are cacheable. This filter uses ten reasons to mark an object uncacheable: CGI requests, queries, pragma headers, cache-control headers, set-cookie responses, authorization headers, vary headers, uncacheable methods, push content types, and response status codes. Wolman et al. [139] find that 40% of requests are to uncacheable objects in the University of Washington trace and 49% in the Microsoft trace.

Empirical studies by Cao et al. [28] and Rizzo et al. [119] show that workloads with more users exhibit higher degrees of temporal locality. This indicates that temporal locality also arises by merging streams of accesses from many independent users, who share a certain amount of common interests. In fact, a large body of work studies the effect of client population sizes to cache hit rates. In their 1997 study, Gribble et al. [63] empirically found that the asymptotic cache hit rate grows logarithmically with the client population size. Similar findings are reported by Duska et al. [51] and Wolman et al. [138]. A model proposed by Breslau et al. [25] and extended by Wolman et al. [138] shows that cache hit rates growing logarithmically with respect to client population size are also a direct consequence of the Zipf-like popularity distribution.

Characterizing Web Servers

Little research is focused on characterizing Web servers and most of it is focused on measuring server availability. In an early study, Viles and French [133] monitor 542 Web servers over several weeks. They measure a 95% availability rate across all servers monitored. Their methodology however is unable to distinguish between server failures and network unreachability. In a later study, Arlitt and Williamson [11] find that a very small percentage of requests result in Web errors (less than 4%). This indicates that most Web requests made are successful. Although few research studies have measured Web server availability, there are several Internet sites (e.g., Netcraft [103], Keynote [79]) that continuously monitor the availability of popular Web servers.

6.2.2 Characterizing Content Delivery Networks

A content delivery network consists of a collection of non-origin servers that attempt to offload work from origin servers by delivering content on their behalf. The servers belonging to a CDN are typically placed at different locations around the network, with some or all of the origin server's content cached or replicated amongst the CDN servers. For each request, the CDN attempts to locate a CDN server close to the client to serve the request, where the notion of "close" could include geographical, topological or latency considerations.

Unlike systems based on the client-server or the peer-to-peer architecture, there is a limited number of content delivery networks on the Internet. These networks are typically run and managed by private companies. There is a relatively little known about how well these systems perform in practice, or how available they are.

The true challenge of CDNs is not to pick the best server, but to pick a reasonably good server. Several studies [73, 83, 80, 15, 17, 16] show that CDNs mostly do a good job at choosing an appropriate CDN server, although there are occasional bad choices. Most of the studies of CDN performance treat these networks as a black-box and they only measure end-to-end performance. As a result, it is unclear why these CDNs do occasionally make bad choices. There are several possibilities: (1) the CDN has incorrect or stale data about a user's network conditions; (2) the CDN has incorrect or stale data about the origin

server; (3) the CDN uses an incorrect algorithm to pick the closest CDN server; (4) the load balancing mechanism does not work properly; or (5) the CDN uses occasional sampling to measure the performance of different servers; these occasional samples could look like "bad choices".

6.2.3 Characterizing Peer-to-Peer Systems

Because peer-to-peer systems are a recent phenomenon, little work on characterizing their workload and their infrastructure exists. A large fraction of previous work focused on measuring properties specific to a particular peer-to-peer protocol. These findings are unlikely to be generalizable. For example, measuring what fraction of Gnutella traffic is made of Gnutella PING packets is a property specific to the Gnutella peer-to-peer system only. We limit our discussion in this section to previous work whose findings are generalizable across peer-to-peer systems' workloads.

Sen and Wang [126] analyze traces of flow-level data of a large tier-1 ISP's backbone. Their focus is on a network-level characterization of peer-to-peer traffic, such as the distribution of hosts across autonomous systems (ASes) and network prefixes, the flow interarrival times, and other low-level network characteristics. Although their goals are different than the ones in this dissertation, two of their results are similar to ours: (1) a small number of IP addresses are responsible for a large fraction of the P2P traffic (0.1% of all IPs, network prefixes, and ASes were responsible for 33%, 27% and 26% of all P2P traffic); and (2) hosts have poor availabilities (60% of IPs participating in P2P systems have sessions whose durations are 10 minutes or less).

Bhagwan et al. [21] analyze 2,400 hosts participating in Overnet, a peer-to-peer system based on Kademlia [96]. The primary contribution of this study is measuring the availability of hosts using a methodology that is not susceptible to DHCP aliasing effects. With DHCP, a single host can use different IP addresses for different sessions of participation in the overlay. Most studies' methodologies (including the ones presented in this dissertation) cannot distinguish between two different hosts with two different IP addresses and one host using two different IP addresses. This methodological flaw can artificially decrease the

measured P2P hosts' availabilities. Bhagwan et al. quantify this error by measuring P2P hosts' availabilities using the two different methodologies: the one susceptible to DHCP and the one that is not. Their results show that DHCP aliasing introduces substantial errors. However, even accounting for DHCP aliasing effects, their study finds that P2P hosts have poor availabilities.

Ripeanu et al. [118, 117] crawl the Gnutella network several times in late 2000, March 2001, and May 2001. Each subsequent crawl discovers more Gnutella hosts: 2,063 peers in 2000, 14,949 peers in March 2001, and 48,195 peers in May 2001. Their study is focused on topological properties of the Gnutella overlay, such as the node degree distribution. In their early crawls, they find that the Gnutella nodes' degree distribution approximates a power-law distribution. However, their later crawls (after March 2001) find a multi-modal distribution, one in which the power-law distribution is a not a good fit for nodes with small degrees. As we pointed out in Section 6.1.2, in March 2001 a series of optimizations were introduced in the protocol that made peers appear invisible to crawlers. We believe that this is a likely cause of the shift of the power-law distribution over time.

Chu et al. [35] perform an analysis of the file workloads of Napster and Gnutella. They find that the distribution of file popularities does not fit a Zipf distribution. The measured file popularity has a "flatter" head and a "shorter" tail than a Zipf curve. Several other results are similar to the ones found in our studies. The average size of a file is 4.2MB in these workloads. The peers' availabilities is poor (31% of the peers' sessions are less than 10 minutes long), although some peers are online for long periods of time (20% of peers' sessions are at least two hours). The only surprising finding is that most of the files and bytes in Gnutella are due to audio: 62% of the file and 79% of the bytes, whereas video accounts for 2% of the files and 19% of the bytes. This finding is contrary to our finding of video bytes dominating the file workloads of peer-to-peer systems. We believe that the discrepancy is due to their measurements preceding ours by one year. In 2002, audio was likely dominating video as the most popular file type in peer-to-peer systems. In fact, one of the most popular peer-to-peer system in 2001 and 2002, Napster, only allowed the exchange of audio files.

Two recent studies [13, 114] characterize BitTorrent, a recent peer-to-peer system used

for distribution of large files. The findings of these studies are remarkably similar to ours: (1) the distribution of file popularities is not Zipf; (2) most downloads last for hours and days; (3) most downloads do not complete in one single session; (4) the speed of most downloads is very slow, on the order of tens of kilobytes per second; and (5) most users' connections are asymmetric.

6.3 Summary

We survey previous research in the areas of (1) designing tools and techniques for measuring large-scale Internet systems; and (2) measurement-based characterizations of Internet content delivery systems. This dissertation further contributes to understanding workload characteristics for the World Wide Web, a content delivery network (Akamai), and two peer-to-peer systems (Gnutella and Kazaa). Furthermore, we provide a measurement-based infrastructure characterization of Gnutella. Finally, we present the design and implementation of a measurement infrastructure for modern Internet content delivery system, including a network tracing system, a bottleneck bandwidth measurement tool, and a crawler for Gnutella.

Three classes of techniques for measuring Internet systems' workloads exist: network tracing, active probing, and software instrumentation. Network tracing represents a viable way to conduct measurement study of Internet systems. However, we point out the numerous challenges associated with implementing a highly scalable network tracing infrastructure. Our network tracing infrastructure addresses many of these challenges. Active probing is not an adequate technique for Internet systems' workload characterization, but rather for infrastructure characterization. Finally, software instrumentation presents many drawbacks making it inadequate for large-scale measurement studies.

Two bottleneck bandwidth measurement techniques exist: the one-packet model and the packet-pair model. The packet-pair technique is more adequate for scaling to large systems measurements. Several key challenges limit the effectiveness of current tools when used in the context of measuring large scale systems. This dissertation presented a tool for measuring bottleneck bandwidth that addresses many of these challenges.

Crawling structured peer-to-peer networks, such as DHTs, is easier and more accurate than crawling unstructured peer-to-peer networks. We find that most of the challenges of crawling P2P are due to specific protocol and software implementations, and they are not fundamental to the peer-to-peer architecture.

We survey related research on characterizing Internet content delivery systems. Similar to our results presented in this dissertation, previous studies found that most Web objects are small in size and highly cacheable. Our traces were collected later than all these other studies, and they are an order of magnitude larger than most of them. We find similar results when characterizing Web workloads.

Most related work on characterizing peer-to-peer systems show findings similar to the ones presented in this dissertation. Like us, previous work finds that: (1) peers have poor availabilities; (2) a small fraction of peers account for a large fraction of the traffic; (3) most files are large in size (on the order of megabytes); (4) the file popularity distribution is not Zipf. However, our traces characterize a larger user population than most of the previous work. Our focus is on characterizing the content delivery on the Internet by comparing the workloads of the Web, content delivery networks, and peer-to-peer systems, rather than characterizing peer-to-peer systems at the network level [126], or characterizing their file workloads [35].

Chapter 7

FUTURE WORK

The Internet is in constant change and evolution. At the microscopic level, the Internet is continuously transforming: new hosts join the Internet, old hosts leave, software and protocols are upgraded, etc. At the macroscopic level, Internet changes occur much more infrequently. However, these macroscopic changes are likely to have a drastic and profound impact on the nature of Internet traffic. The emergence of the Web two decades ago is one example of a macroscopic Internet change. Today's emergence of new content delivery architectures, such as peer-to-peer systems and content delivery networks, together with new audio and video workloads is another example of a macroscopic change. Like today, the Internet is likely to go through other macroscopic changes in the future. The goal of this chapter is to examine current trends and changes that can potentially impact and transform the future Internet as a content delivery vehicle. We classify these changes in two categories: (1) changes to the Internet infrastructure; and (2) changes to workloads and applications.

7.1 Future Changes to the Internet Infrastructure

A current technological trend is the increased penetration of broadband and wireless networking in residential homes. As a result, network conditions (e.g., bandwidths, latencies, and availabilities) of the Internet's last mile are improving. The current last mile's poor network conditions have forced centralized architectures, like the Web, to deliver content from the core and the edge of the network only. Improvements in the last-hop network links suggest that Web architectures might emerge at the outer end of the Internet, as network conditions in this region improve. Several open problems arise in this new scenario. First, will the Web continue to use DNS as the name lookup mechanism, in the face of an increased load in name updates? Will the DNS access control mechanism accommodate a large population of end users? Second, how should we design Web servers that do not interfere with

the users' other foreground activities? We need adequate resource management policies and security policies that isolate the Web server from the rest of the applications.

Improvements in the last-hop network links might also push content delivery networks architectures toward the outer end of the Internet. Although several research projects proposals already exist [108, 58], two serious issues remain unsolved. First, it is unclear how these content delivery networks will maintain centralized control over their content once they are deployed over endhost machines. Second, several serious security attacks and abuses have been already reported on current research prototypes [108].

Improvements in the last-hop network links will also result in more content transport functionality migrating to the outer end of the Internet. We have already witnessed the migration of multicast protocols from the IP layer to the application layer [68, 12]. We anticipate a recast of the streaming media transport protocols in the context of a decentralized peer-to-peer infrastructure, especially with the proliferation of Voice-over-IP technology.

While the network conditions of the current Internet's last mile are likely to improve, a new class of computing platforms is emerging. Tomorrow's clients will request Internet content from mobile PDAs, cell phones, and TV set top-boxes. We believe that today's last mile is slowly being incorporated in the edge of the Internet. Tomorrow's last mile will migrate to these new devices with new constraints. Some devices are mobiles, others are power-constrained, and others are specialized in their functionalities. This transformation adds two new dimensions to the challenges of Internet content delivery. First, content delivery architectures need to start optimizing their content transport for power limited clients. Second, we believe that the clients' mobility should be exploited for transporting content and that newer content delivery architectures need to operate in disconnected mode over mobile endhosts.

7.2 Future Application Workloads

RSS-based Web services are a new class of applications where we believe future workload analysis research is needed. RSS stands for Really Simple Syndication or Rich Site Summary. RSS is an XML-based format for easily distributing and aggregating Web content, such as

news headlines. Users determine their favorite website and a properly configured RSS aggregator combines selected lists of hyperlinks and headlines, along with other information about the websites, then displays the content on the user's desktops at regular intervals. While the current Web browsing mechanism is pull-based, RSS changes Web browsing into a push-based mechanism. Because RSS pushes content to the user, RSS will amplify a user's load on the network. The use of RSS is also likely to change the current Web object popularity distribution which in turn can affect the current Web caching hit rates.

An extension of RSS is the rich media RSS standard (RM RSS). RM RSS allows publishers to make rich media content available to the users in the same way news articles are made available online today. Instead of streaming audio or video, users will create personal "channels" using RM RSS viewers. These act as TiVo-like devices connected to the Web and subscribed to different types of multimedia content.

One recent trend in Internet usage is the deployment Voice-over-IP (VoIP) services. Although VoIP has been in existence for some years, packet-based telephony is currently experiencing a surge in the number of users and companies offering service. Voice protocols have further developed to offer a richer set of features, scalability, and standardization than what was available a few years ago. Most of the VoIP protocols are not based on the TCP protocol, but instead on real-time transport protocols (e.g., RTP or RTCP), multimedia protocols (e.g., H.323), or connectionless protocols (e.g., UDP). VoIP content has a set of drastically different transport requirements than current popular content services like the Web and P2P systems. VoIP traffic is much more sensitive to jitter and and packet loss. As a result, quality of service issues for Internet traffic are likely to re-emerge as important open research problems. Furthermore, a large proliferation of VoIP services in the future Internet will result in a larger increase of non-TCP Internet traffic. It is unclear how the Internet traffic will change in the face of an increasing fraction of traffic whose congestion control mechanisms are different than TCP's.

Another technological trend that will impact the Internet as a content delivery vehicle is the current surge in the number of home digital cameras and camcorders. These devices produce large-sized content (e.g., pictures and movies) that is relatively unpopular. There is already evidence that Internet has become the primary delivery method for these large

objects [91]. Although this content is large in size, current techniques for optimizing the transport of large objects, such as multicast protocols or proxy caching, are unsuitable, due to the low degree of popularity of these objects. Instead we argue that new transport techniques for large-sized, unpopular objects need to be developed that will not interfere with the transport of interactive content or with other, latency sensitive Internet traffic.

Chapter 8

CONCLUSIONS

In this dissertation, we present an in-depth analysis of the nature of content delivery in today's Internet. Our analysis is based on two measurement studies. The common theme across these studies is their focus on measuring and characterizing the properties of workloads and architectures of several popular content delivery systems. In addition to characterizing today's Internet content delivery systems, this dissertation describes a scalable infrastructure for measuring today's large Internet scale systems. In this chapter, we summarize the findings of each of the two studies.

8.1 Workload Characterization

Our first study is a workload characterization of four content delivery systems: the World Wide Web, the Akamai content delivery network, and the Kazaa and Gnutella peer-to-peer systems. The primary goal of our study is to capture the dramatic shift in Internet traffic and usage that has occurred in only several years. To perform this kind of analysis, one needs simultaneously collected traces of each of these four content delivery systems. Because certain workload characteristics change over long periods of time (e.g., the object popularity distribution of audio content), our traces span several consecutive days.

We show that the Internet has undergone substantial change in a few years and we characterize the extent to which it changed. Three years before our study (in 1999), peer-to-peer systems did not exist. In the course of three years only, this new class of systems, based on a new architecture, has emerged. These systems today account for a significant fraction of the Internet content delivery bytes. In 2002, at the University of Washington, Kazaa accounted for 36.9% of all bytes, compared to 14.3% for Web documents.

We find that the mixture of object types in Internet content delivery workloads has also changed. P2P objects (primarily video and audio) account for a substantially larger

fraction of the traffic than ever before. These objects are three orders of magnitude larger than Web objects, leading to 1000-fold increases in transfer times. While P2P systems have low request rates and relatively small populations, the number of concurrent P2P flows is twice the number of flows for the Web.

We find that a small number of P2P objects are responsible for the majority of the bytes consumed by P2P systems. This suggests (and we confirm through experiments) that caching techniques are effective for these workloads.

An original goal of P2P systems was the uniformity of peers' resource contributions. We find that a small number of P2P peers are responsible for the majority of the traffic in these systems. We also find that not all peers have uniform roles: only a small number of peers serve most of the content. These findings show that there is a significant gap between the design of these systems and the practical conditions under which they operate.

Finally, we show that the bandwidth requirements of a single peer in current popular P2P systems like Kazaa are very large. In our traces, we find that a single Kazaa peer consumes ninety times more bandwidth than a single Web client. This suggests that despite the scalability-based designs, the bandwidth demands of peer-to-peer systems like Kazaa will likely prevent them from scaling further, at least within environments similar to the one measured.

8.2 Infrastructure Characterization

Our second study is an infrastructure characterization of Gnutella and Napster. The primary goal of our study is attempting to validate the set of assumptions made when designing new peer-to-peer architectures. For example, a key assumption made in a large number of peer-to-peer architectures is the uniformity of peers' roles and responsibilities. Another assumption is the altruistic nature of the peers: users voluntarily cooperate and share their resources in a non-greedy fashion. To answer these questions, we measure and analyze the network-level characteristics of over one million Gnutella and Napster hosts. We use a two-step process. In the first step, we gather detailed snapshots of these systems in order to collect a large fraction of their entire host population. Second, we probe the discovered

hosts in order to measure their network-level characteristics: bandwidths, latencies, and the amount of time participating in the system.

One of the premises of the peer-to-peer architecture is that peers voluntarily join the system in order to cooperate, exchange resources, and derive benefits from the system. These systems have single and uniform roles; there is no client-server demarcation unlike in the Web or Akamai. As our study shows, we find clear evidence of client-like or server-like behavior in a significant fraction of peer-to-peer systems' populations. We also find a significant amount of heterogeneity in both Gnutella and Napster; bandwidth, latency, availability, and the degree of sharing vary between three and five orders of magnitude across the peers in the system. Even though these systems were designed with a symmetry of roles and responsibilities, in practice, each peer offers and derives a different amount of services from the system. There is a gap between the peer-to-peer premises and the reality of peers' capabilities and behavior.

We also find that peers tend to deliberately misreport information if there is an incentive to do so. This finding has several implications. First, peer-to-peer systems need to be deliberate about building incentive mechanisms in their systems for peers to participate. Second, the peer-to-peer system must be able to enforce these incentives. Subsequent work has investigated schemes for incorporating incentives in P2P systems, such as introducing currency [136], building reputation-based systems [74], or incorporating fair exchange protocols [59, 8].

8.3 Measurement Infrastructure

We present the design and implementation of our measurement infrastructure for characterizing Internet content delivery systems. We start by describing the design and implementation of a tracing system used to collect workloads of four content delivery system: the Web, Akamai, Gnutella, and Kazaa. Our system is an extension of the system used by Wolman et al. [137, 139, 138, 34] to collect Web workloads. There are two key differences between Wolman et al.'s system and ours. First, our system identifies and reconstructs peer-to-peer traffic (e.g., Gnutella and Kazaa) and content delivery network traffic (e.g.,

Akamai), in addition to Web traffic. Second, our system scales to an order of magnitude higher along several different dimensions: network speeds (we trace at speeds of 1Gbps rather than 100Mbps), amount of data (we collect terabytes of data rather than gigabytes), and robustness (we continuously trace for several consecutive months rather than several consecutive days). Our tracing system was in use at the University of Washington Internet border for approximately two years. We monitored Web and Akamai workloads from April 2002 to December 2002. We monitored Gnutella and Kazaa workloads from April 2002 to July 2003.

We present the architecture of a crawler for Gnutella, a peer-to-peer system. The goal of our Gnutella crawler is to gather nearly instantaneous snapshots of a significant subset of the Gnutella population, as well as metadata about peers in the captured subset as reported by the Gnutella system itself. We crawled Gnutella for eight consecutive days (Sunday May 6th, 2001 through Monday May 14th, 2001) and captured 1,239,487 Gnutella peers on 1,180,205 unique IP-addresses.

Finally, we present the design, implementation, and evaluation of SProbe, a bottleneck bandwidth estimation tool based on the packet-pair technique. Unlike previous bottleneck bandwidth tools, SProbe can measure bottleneck bandwidth in an uncooperative environment, one in which measurement software is only deployed on the local host. Our evaluations show that SProbe is accurate, fast, scalable and works in uncooperative environments. It is precisely these properties that made SProbe viable, useful in large-scale network measurements and in different realistic scenarios.

BIBLIOGRAPHY

[1] 3Com. 3Com V.90 technology, 1998. `http://www.mcoecn.org/WhitePapers/3COM-V90-Technology.pdf`.

[2] Soam Acharya and Brian Smith. An experiment to characterize videos stored on the Web. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking*, San Jose, CA, January 1998.

[3] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[4] Akamai. Akamai, August 2004. `http://www.akamai.com`.

[5] Alexa. Alexa, August 2004. `http://www.alexa.com`.

[6] Jussara M. Almeida, Virgilio Almeida, and David J. Yates. Measuring the behavior of a World-Wide Web server. In *Proceedings of the 7th Conference on High Performance Networking*, White Plains, NY, April 1997.

[7] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, December 1996.

[8] Kostas G. Anagnostakis and Michael B. Greenwald. Exchange-based mechanisms for peer-to-peer file sharing. In *The 24th IEEE International Conference on Distributed Computing (ICDCS 2004)*, Tokyo, Japan, March 2004.

[9] Martin Arlitt, Rich Friedrich, and Tai Jain. Workload characterization of a Web proxy in a cable modem environment. *ACM SIGMETRICS Performance Evaluation Review*, 27(2):25–35, September 1999.

[10] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 World Cup Web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[11] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workloads characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5), October 1997.

[12] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM SIGCOMM Technical Conference*, Pittsburgh, PA, August 2002.

[13] Anthony Bellissimo, Prashant Shenoy, and Brian Neil Levine. Exploring the use of BitTorrent as the basis for a large trace repository. Technical Report 04-41, University of Massachusetts at Amherst, 2004.

[14] Steven M. Bellovin. A best-case network performance model, February 1992. `http://www.research.att.com/~smb/papers/netmeas.pdf`.

[15] Leanne Bent and Geoffrey M. Voelker. Whole page performance. In *Proceedings of 7th International Web Caching Workshop (WCW)*, Boulder, CO, August 2002.

[16] Leeann Bent, Michael Rabinovich, Geoffrey M. Voelker, and Zhen Xiao. Characterization of a large web site population with implications for content delivery. In *Proceedings of the International World Wide Web Conference (WWW)*, New York, NY, May 2004.

[17] Leeann Bent, Michael Rabinovich, Geoffrey M. Voelker, and Zhen Xiao. Towards informed Web content delivery. In *Proceedings of the 9th International Workshop on Web Content Caching and Distribution (WCW)*, Beijing, China, October 2004.

[18] Tim Berners-Lee. Information management: A proposal, May 1990. `http://www.w3.org/History/1989/proposal.html`.

[19] Tim Berners-Lee, Larry Masinter, and Mark McCahill. RFC 1738 - uniform resource locators (url), December 1994. `http://www.faqs.org/rfcs/rfc1738.html`.

[20] Azer Bestavros, Bob Carter, Mark Crovella, Carlos Cunha, Abdelsalam Heddaya, and Suliman Mirdad. Application-level document caching in the Internet. In *Proceedings of SDNE'95: The second International Workshop on Services in Distributed and Network Environments*, Whistler, BC, Canada, June 1995.

[21] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Boston, MA, March 2002.

[22] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[23] Jean-Chrysostome Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of the ACM SIGCOMM 1993 Technical Conference*, San Francisco, CA, August 1993.

[24] Karlheinz Brandenburg. MP3 and AAC explained. In *Proceedings of the 17th International Conference on High Quality Audio Coding*, Florence, Italy, September 1999.

[25] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the 18th Conference of the IEEE Communications Society (INFOCOM)*, New York, NY, March 1999.

[26] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: The devil is in the details. In *Proceedings of the ACM SIGMETRICS Workshop on Internet Server Performance*, Madison, WI, June 1998.

[27] Ramón Cáceres, Balachander Krishnamurthy, and Jennifer Rexford. HTTP 1.0 logs considered harmful. In *Proceedings of the W3C Web Characterization Group Workshop*, Cambridge, MA, November 1998.

[28] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[29] Pei Cao, Jin Zhang, and Kevin Beach. Active cache: Caching dynamic contents on the Web. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, The Lake District, UK, September 1998.

[30] Robert Carter. Cprobe and bprobe tools. `http://cs-people.bu.edu/carter/tools/Tools.html`, 1996.

[31] Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report BU-CS-96-006, Computer Science Department, Boston University, March 1996.

[32] Lara D. Catledge and James E. Pitkow. Characterizing browsing strategies in the World-Wide Web. In *Proceedings of the Third International World-Wide Web conference on Technology, Tools and Applications*, Darmstadt, Germany, April 1995.

[33] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, CA, January 1996.

[34] Maureen Chesire, Alec Wolman, Geoffrey M. Voelker, and Henry M. Levy. Measurement and analysis of a streaming-media workload. In *Proceedings of the Third*

USENIX Conference on Internet Systems and Technologies (USITS), San Francisco, CA, March 2001.

[35] Jacky Chu, Kevin Labonte, and Brian Neil Levine. Availability and locality measurements of peer-to-peer file systems. In *Proceedings of ITCom: Scalability and Traffic Control in IP Networks II Conferences*, Boston, MA, July 2002.

[36] David D. Clark. The design philosophy of the DARPA Internet protocols. In *Proceedings of the ACM SIGCOMM Technical Conference*, Stanford, CA, September 1988.

[37] Clip2. The Gnutella protocol specification v0.4, August 2004. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[38] Adrian Cockcroft. Watching your Web servers. Unix Insider, March 1996. `http://www.sun.com/sun-on-net/itworld/UIR960301perf.html`.

[39] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of World Wide Web client-based traces. Technical Report BUCS-TR-1995-010, Boston University University Technical Report, April 1995.

[40] Michel Dagenais, Richard Moore, Robert Wisniewski, Karim Yaghmour, and Thomas Zanussi. Efficient and accurate tracing of events in linux clusters. In *Proceedings of 2003 High Performance Computing Systems and Applications*, Sherbrooke, PQ, Canada, May 2003.

[41] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A case for caching file objects inside internetworks. In *Proceedings of the ACM SIGCOMM 1993 Technical Conference*, San Francisco, CA, August 1993.

[42] Brian D. Davison. Simultaneous proxy evaluation. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, San Diego, CA, March/April 1999.

[43] Brian D. Davison. Web traffic logs: An imperfect resource for evaluation. In *Proceedings of the Ninth Annual Conference of the Internet Society (INET'99)*, San Jose, CA, June 1999.

[44] Brian D. Davison and Chandrasekar Krishnan. Rope: The Rutgers online proxy evaluator. Technical Report DCS-TR-445, Rutgers University, Department of Computer Science, August 2001.

[45] Brian D. Davison and Chandrasekar Krishnan. Rope: The rutgers online proxy evaluator, August 2004. `http://www.cse.lehigh.edu/~brian/pubs/2001/dcs-tr-445/`.

[46] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and JEffrey C. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[47] Constantinos Dovrolis. Pathrate. `http://www.cis.udel.edu/~dovrolis/bwmeter.html`, 2001.

[48] Constantinos Dovrolis, Parmesh Ramanathan, and David Moore. What do packet dispersion techniques measure? In *Proceedings of the IEEE INFOCOM 2001*, Anchorage, AK, USA, April 2001.

[49] Allen B. Downey. Clink, 1999. `http://rocky.wellesley.edu/downey/clink/`.

[50] Allen B. Downey. Using pathchar to estimate internet link characteristics. In *Proceedings of the ACM SIGCOMM 1999*, Cambridge, MA, September 1999.

[51] Bradley M. Duska, David Marwood, and Michael J. Feeley. The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[52] David J. Ewing, Richard S. Hall, and Michael F. Schwartz. A measurement study of Internet file transfer traffic. Technical Report CU-CS-571-92, University of Colorado Technical Report, January 1992.

[53] Anja Feldmann. Continuous online extraction of HTTP traces from packet traces. In *Proceedings of the W3C Web Characterization Group Workshop*, Cambridge, MA, November 1998.

[54] Anja Feldmann. BLT: Bi-layer tracing of HTTP and TCP/IP. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Holland, May 2000.

[55] Anja Feldmann, Ramón Cáceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proceedings of the 18th Conference of the IEEE Communications Society (INFOCOM)*, New York, NY, March 1999.

[56] Roy Fielding. RFC 1808 - relative uniform resource locators, June 1995. `http://www.faqs.org/rfcs/rfc1808.html`.

[57] David Flanagan. *JavaScript, The Definitive Guide*. O'Reilly & Associates Inc., 1998.

[58] Michael Freedman, Eric Freudenthal, and David Mazieres. Democratizing content publication with Coral. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[59] Paul Gauthier, Brian Bershad, and Steven D. Gribble. Dealing with cheaters in anonymous peer-to-peer networks. Technical Report 04-01-03, University of Washington, Computer Science and Engineering, January 2004.

[60] Steven Glassman. A caching relay for the World Wide Web. In *Proceedings of the 1st World Wide Web Conference*, Geneva, Switzerland, May 1994.

[61] Gnutella. Gnutella, August 2004. `http://www.gnutella.com`.

[62] Steven D. Gribble. Personal communication, December 2004.

[63] Steven D. Gribble and Eric A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[64] C.D. Group. Gnutella: To the bandwidth barrier and beyond., May 2001. Originally found at `http://dss.clip2.com/gnutella.html`.

[65] Trusted Computing Group. The trusted computing group home page, August 2004. `https://www.trustedcomputinggroup.org/home`.

[66] Edward Growchowski. Emerging trends in data storage on magnetic hard disk drivers. *datatech*, pages 11–16, September 1998.

[67] Krishna P. Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[68] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002.

[69] Information Sciences Institute. RFC 793 - transmission control protocol, September 1981. `http://www.faqs.org/rfcs/rfc793.html`.

[70] Internet2. Internet2 netflow: Weekly reports: Week of 20020422, August 2004. `http://netflow.internet2.edu/weekly/20020422`.

[71] Van Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM 1988 Conference*, Stanford, CA, August 1988.

[72] Van Jacobson. Pathchar, August 1997. `http://www.caida.org/tools/utilities/others/pathchar/`.

[73] Kirk L. Johnson, John F. Carr, Mark S. Day, and M. Frans Kaashoek. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[74] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International World Wide Web Conference*, Budapest, Hungary, May 2003.

[75] Kazaa. Kazaa, August 2004. `http://www.kazaa.com`.

[76] Terence Kelly. Optimization in web caching: Cache management, capacity planning, and content naming. Technical report, Ph.D. Dissertation – University of Michigan, 2002.

[77] Terence Kelly and Jeffrey Mogul. Aliasing on the World Wide Web: Prevalence and performance implications. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, HI, May 2002.

[78] Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of the ACM SIGCOMM 1991 Technical Conference*, Zurich, Switzerland, September 1991.

[79] Keynote. Keynote, August 2004. `http://www.keynote.com`.

[80] Mimika Koletsou and Geoff Voelker. The Medusa proxy: A tool for exploring user-perceived Web performance. In *Proceedings of the Sixth Annual Web Caching Workshop (WCW01)*, Boston, MA, June 2001.

[81] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice*. Addison-Wesley, 2001.

[82] Balachander Krishnamurthy and Craig E. Wills. Analyzing factors that influence end-to-end Web performance. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Holland, May 2000.

[83] Balachander Krishnamurthy, Craig E. Wills, and Yin Zhang. On the use and performance of content distribution networks. In *Proceedings of the 1st SIGCOMM Internet Measurement Workshop*, San Francisco, CA, November 2001.

[84] Thomas M. Kroeger, Darrel D.E. Long, and Jeffrey C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[85] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. User access patterns to NCSA's World-Wide Web server. Technical Report UIUCDCS-R-95-1934, University of Illinois, Dept. of Computer Science, February 1995.

[86] Kevin Lai. Nettimer. `http://mosquitonet.stanford.edu/~laik/projects/nettimer/`, 2000.

[87] Kevin Lai and Mary Baker. Measuring bandwidth. In *Proceedings of the IEEE INFOCOM 1999*, New York, NY, March 1999.

[88] Kevin Lai and Mary Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of the ACM SIGCOMM 2000 Technical Conference*, Stockholm, Sweden, August 2000.

[89] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[90] Jian Liang, Rakesh Kumar, and Keith W. Ross. Understanding Kazaa. In *Proceedings of the Fifth New York Metro Area Networking Workshop (NYMAN)*, Ney York, NY, September 2005.

[91] Peter Lyman and Hal R. Varian. How much information 2003?, 2003.

[92] Bruce A. Mah. An empirical model of HTTP network traffic. In *Proceedings of the 16th Conference of the IEEE Communications Society (INFOCOM)*, Kobe, Japan, April 1997.

[93] Bruce A. Mah. pchar, 1999. `http://www.caida.org/tools/utilities/others/pathchar/`.

[94] Stephen Manley and Margo Seltzer. Web facts and fantasy. In *Proceedings of the First USENIX Symposium on Internet Technology and Systems (USITS)*, Monterey, CA, December 1997.

[95] Evangelos P. Markatos. Main memory caching of Web documents. In *Proceedings of the 5th International World Wide Web Conference*, Paris, France, May 1996.

[96] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Boston, MA, March 2002.

[97] Steve McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Technical Conference*, San Diego, CA, January 1993.

[98] Art Mena and John Heidermann. An empirical study of real audio traffic. In *Proceedings of the 19th Conference of the IEEE Communications Society (INFOCOM)*, Tel Aviv, Israel, March 2000.

[99] David Meyer. RouteViews project, August 2004. `http://www.routeviews.org`.

[100] Jeffrey C. Mogul. Network behavior of a busy Web server and its clients. Technical Report WRL-TR-95.5, Western Research Laboratory Research Report, October 1995.

[101] Jeffrey C. Mogul. Errors in timestamp-based HTTP header values. Technical report, Research Replort 99/3, Compaq Computer Corporation Western Research Laboratory, November 1999.

[102] Napster. Napster, August 2004. `http://www.napster.com`.

[103] Netcraft. Netcraft, August 2004. `http://www.netcraft.com`.

[104] Free Music Now. history of mp3.com, August 2004. `http://www.free-music-now.com/history_of_mp3-dot-com.shtml`.

[105] Markus F. X. J. Oberhumer. LZO real-time data compression library, August 2004. `http://www.oberhumer.com/opensource/lzo/`.

[106] Jitendra Padhye and Sally Floyd. Identifying the TCP Behavior of Web Servers. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, August 2001.

[107] Venkata N. Padmanabhan and Lili Qiu. The content and access dynamics of a busy Web site: Findings and implications. In *Proceedings of the ACM SIGCOMM 2000 Technical Conference*, Stockholm, Sweden, August 2000.

[108] Vivek Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The dark side of the Web: An open proxy's view. In *Proceedings of 2nd Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, MA, November 2003.

[109] Vern Paxson. Growth trends in wide-area TCP connections. *IEEE Network*, 8(4):8–17, July/August 1994.

[110] Vern Paxson. Measurements and dynamics of end-to-end Internet dynamics. Technical report, Ph.D. Dissertation – University of California, Berkeley, 1997.

[111] Vern Paxson and Sally Floyd. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, December 1997.

[112] James E. Pitkow and Margaret M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of the Second International World Wide Web Conference*, Chicago, IL, October 1994.

[113] Dave Plonka. UW-Madison Napster traffic measurement, August 2000. `http://net.doit.wisc.edu/data/Napster/`.

[114] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. A measurement study of the BitTorrent peer-to-peer file-sharing system. Technical Report PDS-2004-003, Technical Report – University of Colorado, 2004.

[115] Ramakrishnan Rajamony and Mootaz Elnozahy. Measuring client-perceived response times on the WWW. In *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco, CA, March 2001.

[116] Sylvia Ratnasamy, Paul Francis, Mark Hendley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, August 2001.

[117] Matei Ripeanu and Ian Foster. Mapping Gnutella network. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Boston, MA, March 2002.

[118] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal special issue on peer-to-peer networking*, 6(1), January/February 2002.

[119] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, September/October 2000.

[120] Drew Roselli, Jacob Lorch, and Thomas Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, USA, June 2000.

[121] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001.

[122] Stefan Saroiu, Steven D. Gribble, and Henry M. Levy. Measurement and analysis of spyware in a university environment. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[123] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of Internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[124] Stefan Savage. Sting: a TCP-based network measurement tool. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, October 1999.

[125] Jeff Sedayao. Mosaic will kill my network! – studying network traffic patterns of Mosaic use. In *Proceedings of the Second World Wide Web Conference*, Chicago, IL, October 1994.

[126] Subharata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks.

In *Proceedings of the 2nd SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[127] F. Donelson Smith, Félix Hernández Campos, Kevin Jeffay, and David Ott. What TCP/IP protocol headers can tell us about the Web. In *Proceedings of ACM SIGMETRICS/Performance*, Cambridge, MA, June 2001.

[128] Squid. Squid Web proxy cache, December 2003. `http://www.squid-cache.org`.

[129] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[130] Linda Tauscher and Saul Greenberg. How people revisit Web pages: empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47(1):97–137, 1997.

[131] Henk Uijterwaal and Olaf Kolkman. Internet delay measurements using test traffic, May 1997. `http://www.ripe.net/ripe/docs/ripe-158.html`.

[132] Jacobus van der Merwe, Ramón Cáceres, Yang hua Chu, and Cormac Sreenan. mmdump: A tool for monitoring Internet multimedia traffic. *ACM Computer Communication Review*, 30(4), 2000.

[133] Charles L. Viles and James C. French. Availability and latency of World Wide Web information servers. *Computing Systems*, 8(1):61–91, 1995.

[134] Marc Waldman, Avi D. Rubin, and Lorrie F. Cranor. Publius: A robust, tamper-evident, censorship-resistant, Web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, August 2000.

[135] David Watson, G. Robert Malan, and Farnam Jahanian. An extensible probe architecture for network protocol performance measurement. *Software: Practice & Experience*, 34(1):47–67, January 2004.

[136] Bryce Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Boston, MA, March 2002.

[137] Alec Wolman. Sharing and caching characteristics of Internet content. Technical report, Ph.D. Dissertation – University of Washington, 2002.

[138] Alec Wolman, Geoff Voekler, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry Levy. On the scale and performance of cooperative Web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, Kiawah Island, SC, December 1999.

[139] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Molly Brown, Tashana Landray, Denise Pinnel, Anna Karlin, and Henry Levy. Organization-based analysis of Web-object sharing and caching. In *Proceedings of the Second USENIX Conference on Internet Systems and Technologies (USITS)*, Boulder, CO, October 1999.

[140] Allison Woodruff, Paul M. Aoki, Eric Brewer, Paul Gauthier, and Lawrence A. Rowe. An investigation of documents from the World Wide Web. *Computer Networks and ISDN Systems*, 28(7–11):963–980, May 1996.

[141] Roland Wooster, Stephen Williams, and Patrick Brooks. HTTPDUMP network HTTP packet snooper, April 1996. `http://ei.cs.vt.edu/~succeed/96httpdump/`.

[142] Kurt J. Worrell. Invalidation in large scale network object caches. Technical report, Master's Thesis – University of Colorado, 1994.

[143] Rouzbeh Yassini. *Planet Broadband*. Cisco Press, 2003.

[144] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

VITA

Stefan Saroiu was born on April 6th, 1975 in Bucharest, Romania. In 1999, he received his B.Math. in Computer Science and Combinatorics & Optimization from the University of Waterloo. He attended University of Washington, where he received his M.S. degree in 2001, and his Ph.D. degree in 2005 in Computer Science.