Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload

Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

{gummadi,rdunn,tzoompy,gribble,levy,zahorjan}@cs.washington.edu

ABSTRACT

Peer-to-peer (P2P) file sharing accounts for an astonishing volume of current Internet traffic. This paper probes deeply into modern P2P file sharing systems and the forces that drive them. By doing so, we seek to increase our understanding of P2P file sharing workloads and their implications for future multimedia workloads. Our research uses a three-tiered approach. First, we analyze a 200-day trace of over 20 terabytes of Kazaa P2P traffic collected at the University of Washington. Second, we develop a model of multimedia workloads that lets us isolate, vary, and explore the impact of key system parameters. Our model, which we parameterize with statistics from our trace, lets us confirm various hypotheses about file-sharing behavior observed in the trace. Third, we explore the potential impact of locality-awareness in Kazaa.

Our results reveal dramatic differences between P2P file sharing and Web traffic. For example, we show how the immutability of Kazaa's multimedia objects leads clients to fetch objects at most once; in contrast, a World-Wide Web client may fetch a popular page (e.g., CNN or Google) thousands of times. Moreover, we demonstrate that: (1) this "fetch-at-most-once" behavior causes the Kazaa popularity distribution to deviate substantially from Zipf curves we see for the Web, and (2) this deviation has significant implications for the performance of multimedia file-sharing systems. Unlike the Web, whose workload is driven by document change, we demonstrate that clients' fetch-at-most-once behavior, the creation of new objects, and the addition of new clients to the system are the primary forces that drive multimedia workloads such as Kazaa. We also show that there is substantial untapped locality in the Kazaa workload. Finally, we quantify the potential bandwidth savings that locality-aware P2P file-sharing architectures would achieve.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Measurement, performance, design

Keywords

Peer-to-peer, multimedia workloads, Zipf's law, modeling, measurement

1. INTRODUCTION

A decade after its birth, the Internet continues its rapid and often surprising evolution. Recent studies have shown a dramatic shift of Internet traffic away from HTML text pages and images and towards multimedia file sharing. For example, a March 2000 study at the University of Wisconsin found that the bandwidth consumed by Napster had edged ahead of HTTP bandwidth [28]. Only two years later, a University of Washington study showed that peer-to-peer file sharing dominates the campus network, consuming 43% of all bandwidth compared to only 14% for WWW traffic [29]. When comparing these bandwidth consumption statistics to those in a 1999 study of a similar campus workload [37], the portion of network bytes ascribed to audio and video increased by 300% and 400%, respectively, over a 3-year period. Without question, multimedia file sharing has become a dominant factor in today's Internet. In all likelihood, it will dominate the Internet of the future, barring a chilling effect from legal assaults.

Two traits of today's file-sharing systems distinguish them from Web-based content distribution. First, current file-sharing systems use a "peer-to-peer" design: peers voluntarily provide resources as well as consume them. Because of this, the system must dynamically adapt to maintain service continuity as individual peers come and go. Second, file-sharing is being used predominantly to distribute multimedia files, and as a result, file-sharing workloads differ substantially from Web workloads [29]. Multimedia files are large (several megabytes or gigabytes, compared to several kilobytes for typical Web pages), they are immutable, and there are currently far fewer distinct multimedia files than

Web pages. Video-on-demand (VoD) systems also distribute multimedia files; we contrast our Kazaa measurements and analysis with related work on VoD systems in Section 3.

This paper presents an in-depth analysis of a modern P2P multimedia file-sharing workload, considering the "peer-to-peer" and the "multimedia" aspects of the workload independently. Our goals are:

1. to understand the fundamental properties of multimedia file-sharing systems, independent of the design of the delivery system,
2. to explore the forces driving P2P file-sharing workloads so we can anticipate the potential impacts of future change, and
3. to demonstrate that significant opportunity exists to optimize performance in current file-sharing systems by exploiting untapped locality in the workload.

To meet these goals, we employ several approaches. First, we analyze a 200-day trace of Kazaa [19] traffic that we collected at the University of Washington between May and December of 2002. We traced over 20 terabytes of traffic during that period, from which we distill several key lessons about the Kazaa workload. Second, we derive a model of this multimedia traffic based on our analysis. The model helps to explain the root causes of many of the trends shown by Kazaa and to predict how the trends may change as the workload evolves. Third, we use trace-driven simulation to quantify the significant potential that exists to improve the performance of multimedia file-sharing in our environment.

Our analysis reveals that the Kazaa workload is driven by considerably different forces than the Web. Kazaa objects are immutable, and as a result, the vast majority of its objects are fetched *at most once* per client; in contrast, Web pages (e.g., Google or CNN) can be fetched thousands of times per client. Our measurements show that the popularity distribution of Kazaa objects deviates *substantially* from the Zipf curves we commonly see for the Web, and our model confirms that the "fetch-at-most-once" behavior of Kazaa clients is the cause. Our model demonstrates another consequence of object immutability: unlike the Web, whose workload is driven by document change, the primary forces in Kazaa are the creation of new objects and the addition of new users. Without these forces, fetch-at-most-once behavior would drive the system to stagnation.

The structure of this paper follows the multi-tiered approach cited above. Section 2 describes our trace methodology and presents our trace-based analysis of Kazaa. In Section 3, we analyze the popularity distribution of Kazaa requests and the forces that shape it. Section 4 uses our observations and analysis to develop an analytical model; we then use the model to explore the processes that drive Kazaa's behavior in greater depth. Section 5 considers the performance potential of bandwidth-saving techniques suggested by our modeling and analysis. We describe research related to our study in Section 6, while Section 7 summarizes our results and presents conclusions.

2. THE MEASURED PROPERTIES OF P2P FILE-SHARING WORKLOADS

This section uses our trace data to identify key properties of the Kazaa multimedia file-sharing system. Recent studies have described high-level characteristics of P2P workloads [6, 29, 30, 32]. Our goals in this section are to: (1) dig beneath these high-level studies to uncover the processes that drive the workloads, and (2) demonstrate ways in which these processes fundamentally differ from those of the Web.

trace length	203 days, 5 hours, 6 minutes
# of requests	1,640,912
# of transactions	98,997,622
# of unsuccessful transactions	65,505,165 (66.2%)
average transaction size	252KB (all transactions) 752KB (successful transactions only)
# of users	24,578
# of unique objects	633,106 (totaling 8.85TB)
bytes transferred	22.72TB
content demanded	43.87TB

Table 1: *Kazaa trace summary statistics, 5/28/02 to 12/17/02.* A transaction refers to a single Kazaa HTTP transfer; a request refers to the set of transactions a client issues while downloading an object, potentially in pieces from many servers. Clients are identified by Kazaa username. We only present statistics for downloads made by university-internal clients for data on university-external peers.

2.1 Trace Methodology

The data in this section are based on a 200-day trace of Kazaa peer-to-peer file-sharing traffic collected at the University of Washington between May 28th and December 17th, 2002. The University of Washington (UW) is a large campus with over 60,000 faculty, students, and staff. Table 1 describes the trace, which saw over 20 terabytes of incoming data resulting from 1.6 million requests. Our trace was long enough for us to observe seasonal traffic variations, including the end of spring quarter in June, the summer months, and the start and end of the fall quarter. We also observed the impact of bandwidth rate-limiting instituted by the university's networking organization midway through the trace in an attempt to control the cost of Kazaa traffic.[1]

We collected our trace using hardware and software installed at the network border between the university and the Internet. Our hardware consists of a 2.0 GHz Pentium-III workstation that monitored traffic flowing between the university and the Internet. Our workstation had sufficient CPU and network capacity to ensure that no packets were dropped, even during peak load. An adjacent workstation acted as a one-terabyte file store for archiving trace data. Our software used a kernel packet filter [24] to deliver TCP packets to a user-level process, which identified HTTP requests within the TCP flows. Throughout our trace, the packet filter reported packet drop rates of less than 0.0001%. We made all sensitive information anonymous – including IP addresses, URLs, usernames, and object names – before compressing and storing the trace. Overall, our tracing and analysis software consists of over 30,000 lines of code.

Our hardware monitored all incoming and outgoing traffic. However, the data presented in this paper (including Table 1) are for one direction only: requests made by university-internal peers to download data stored on university-

[1]The imposed rate limits bounded *upload* traffic out of the university's dormitory population and had little effect on *download* traffic (to the dorms or to the university as a whole), which is the focus of our research.

external peers. This unidirectional trace captures all requests issued by a stable, complete user population over a period of time. Kazaa control traffic, which consists primarily of queries and their responses, is encrypted and was not captured as part of our trace.

Throughout this paper, the term "user" refers to a person and "client" refers to the application instance running on behalf of a user. We assume there is largely a one-to-one correspondence between users and specific application instances in our environment (although this may not always be true); therefore, we draw conclusions about users based on observations of clients in our trace. Note, however, that client-side caches may absorb some requests from users, meaning that the *client* request rate, which we observe in our trace, may be lower than the true *user* request rate, which we cannot directly observe.

Kazaa clients supply Kazaa-specific "usernames" as an HTTP header in each transaction. We use these usernames (rather than IP addresses) to distinguish between different users in our trace. Unfortunately, an unofficial version of Kazaa, called "KazaaLite,"[2] became popular during our tracing period and is compiled with a predefined username embedded in the application itself. We "special-case" requests from KazaaLite, resorting to distinguishing between KazaaLite users by their IP addresses. Although DHCP is used in portions of our campus, and identifying users by IP address is known to have issues when DHCP is present [6], only 5.7% of transactions in our trace were from KazaaLite clients. Furthermore, KazaaLite clients did not appear within the first 59 days of our 203 day trace.

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 for us to identify precisely which object is being transferred in a given transaction. When a client attempts to download an object, that object may be downloaded in pieces (often called "chunks") from several sources over a long period of time. We define a "transaction" to be a single HTTP transfer between a client and a server, and a "request" to be the set of transactions a client participates in to download an entire object. A failed transaction occurs when a client successfully contacts a remote peer, but that remote peer does not return any data, instead returning an HTTP 500 error code.

A single request may span many minutes, hours, or even days, since the Kazaa client software will continue to attempt to download the object long after the user has asked for it. 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). We call this a "partial request."

The Kazaa application has an auto-update feature, meaning that a running instance of Kazaa will periodically check for updated versions of itself. If found, it downloads the new executable over the Kazaa network. We chose to filter out these auto-update transactions from our logs, as they are not representative of multimedia requests from users. Such filtering removed 0.4% of total transactions (0.3% of bytes) from our trace.

[2]Many more unofficial versions of Kazaa that use generic usernames have appeared since our trace period finished; precisely distinguishing between peer-to-peer users will become very difficult, given that neither IP addresses nor application-specific usernames are unique.

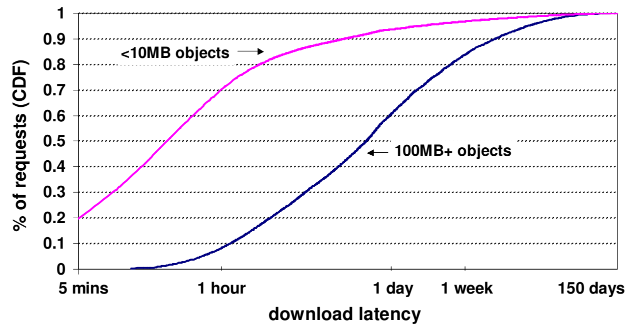

Figure 1: ***Users are patient.*** **A CDF of object transfer times in Kazaa for small (<10 MB) and large (100 MB+) objects. The X-axis is on a log-scale.**

2.2 User Characteristics

Our first slice through the trace data focuses on the properties of Kazaa users. Previous studies [29, 2] have shown that peer-to-peer users in general are "greedy" (i.e., most users consume data but provide little in return) and have poor availability [30]. We confirm some of these characteristics, but we also explore others, such as user activity.

2.2.1 Kazaa Users Are Patient

As any Web-based enterprise knows, users are very sensitive to page-fetch latency. Beyond a certain small threshold (measured in seconds), they will abandon a site and move to another, possibly competing, site. For this reason, many online businesses engage services such as Keynote [20] to tell them quickly if their servers are not sufficiently responsive. In the world of the Web, users expect instant gratification and are unforgiving if they do not receive it.

In this context, the behavior of Kazaa users is surprising. Figure 1 shows the distribution of transfer times in Kazaa; *transfer time* is defined as the difference between the start time of the first transaction and the end time of the last transaction of a given user request. We filtered out partial requests (i.e., we only counted transfers for which the user eventually obtained the entire object). To deal with edge effects, we ignored requests for which at least one transaction occurred during the first month of the trace; note that this will tend to result in an underestimate of user patience. We present our results in terms of requests for "small" objects (less than 10MB, typically audio files) and requests for "large" objects (more than 100MB, typically video files). As we will show in Section 2.3.1, this is a natural and representative way to decompose the overall workload.

The results show incredible patience on the part of Kazaa users. Even for small objects, 30% of the requests take over an hour, and 10% take nearly a day. For large requests, less than 10% complete in an hour, 50% take more than a day, and nearly 20% of users are willing to wait a week for their downloads to complete! From this graph, it is clear that the dynamics of multimedia downloads differ totally from Web usage. The Web is an *interactive* system where users want an immediate response. Kazaa is a *batch-mode* delivery system, where downloads are made in the background and the

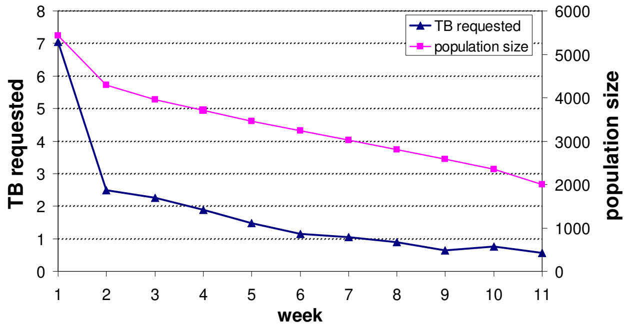

Figure 2: ***Bytes requested by the population and attrition as a function of age.*** **"Older" clients request a smaller fraction of bytes than newer clients. There are fewer old clients than young clients, but attrition occurs at a more gradual rate than the slowdown in bytes requested.**

content is examined later. Users do not wait for their content to arrive. They go about their business and eventually return to review the content after it has been received.

2.2.2 Users Slow Down As They Age

An interesting question we explored is how user interest in Kazaa varies over time. Do users become hungrier for content as they gain experience with Kazaa? Are their request rates relatively constant? Do they lose interest over time? The answer to such questions significantly affects the growth of the system.

To understand user behavior over time, we first calculated the average number of bytes consumed by clients as a function of age. The methodology for this measurement is complex: our trace has finite length, so we must avoid end effects that would overcount short-lived or undercount long-lived users. We compensate, first, by counting transferred bytes only from clients whose "births" we could observe. Because there are no detectable birth events in our trace, we used the heuristic of treating the first observed download from a client as a birth event if *at least* a full month had elapsed in our trace before seeing that first download. To compensate at the end of the trace, we counted bytes only from clients born prior to the last 11 weeks of the trace. Because of this "end threshold," we could draw definitive conclusions about clients' behavior only during the first 11 weeks of their lifetimes.

Figure 2 shows the total number of bytes requested by the population as a function of its age. From this graph, we can see that older clients consume fewer bytes than newer clients. There are two reasons for this effect: (1) attrition reduces the number of older clients, since clients may "die" (i.e., leave the system forever) over time, and (2) some clients may continue to issue requests but do so at a slower rate as they age. We explore each of these in turn.

Attrition. To understand attrition in the system, we analyzed the number of clients that remain alive as a function of age (also shown in Figure 2). Population size declines at a more gradual rate than bytes requested, at least over the first 11 weeks of clients' lifetimes. Attrition therefore only partially explains why older clients demand less in aggregate from the system. To fully explain this phenomenon, clients must also slow down their request rates as they age.

Slowing down over time. Older clients may have slower

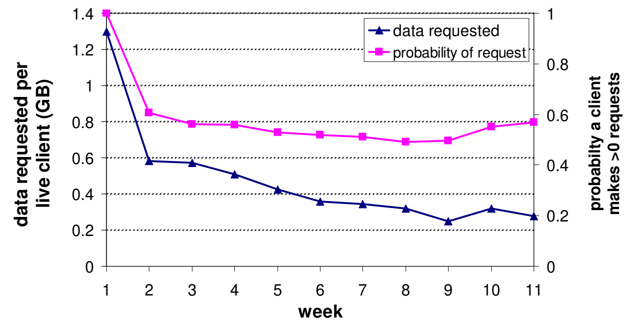

Figure 3: ***Older clients have slower request rates.*** **On average, clients use the system equally often as they age (having approximately a 50% chance of using the system any given week), but they request less data per session as they age. Note that the point corresponding to new clients (week 1) is artificially high, since by definition every new client requests an object immediately.**

request rates for two reasons: (1) they may use the system less often, or (2) they may ask for less when they use the system. Figure 3 shows that clients are equally likely to use the system regardless of age: on average, clients have about a 50% chance of making a request on any given week. Older clients slow down because they ask for less each time they use the system, not because they use the system less often.

In summary, new clients generate most of the load in Kazaa, and older clients consume fewer bytes as they age. In part, this is because of attrition: clients leave the system permanently as they grow older. Also, older clients tend to interact with the system at a constant rate, but ask for less during each interaction.

2.2.3 Client Activity

Quantifying the availability of clients in a peer-to-peer system is notoriously difficult [6]: no one metric can accurately capture availability in this environment, since any individual client might exist only for a fraction of the traced time period. Given our passive tracing methodology, we faced an additional methodological problem: we can detect that users are participating in the system only when their clients transfer data (either by downloading or uploading files). If a client is on-line but not active, we could not observe them. Because of this, we report statistics about *client activity* only, which is a lower bound on availability.[3]

We use two specific metrics to quantify the amount of client transfer activity: (1) *activity fraction*, which measures the fraction of time a client is transferring content over the client's lifetime or over the duration of the entire trace, and (2) *average session length*, in which a session is defined as an unbroken period of time during which a client has one or more active transactions. Average session length measures the typical duration of the periods during which a client is receiving or transmitting data. Our measurements indicate that the distributions of average session length and activity fraction over the measured population are heavy-tailed.

[3]P2P software is often designed to make it difficult to close the program once it starts, "fooling" users into making their clients more available than intended. Accordingly, we suggest that client activity is a more universally comparable and stable indicator of "availability" than other metrics.

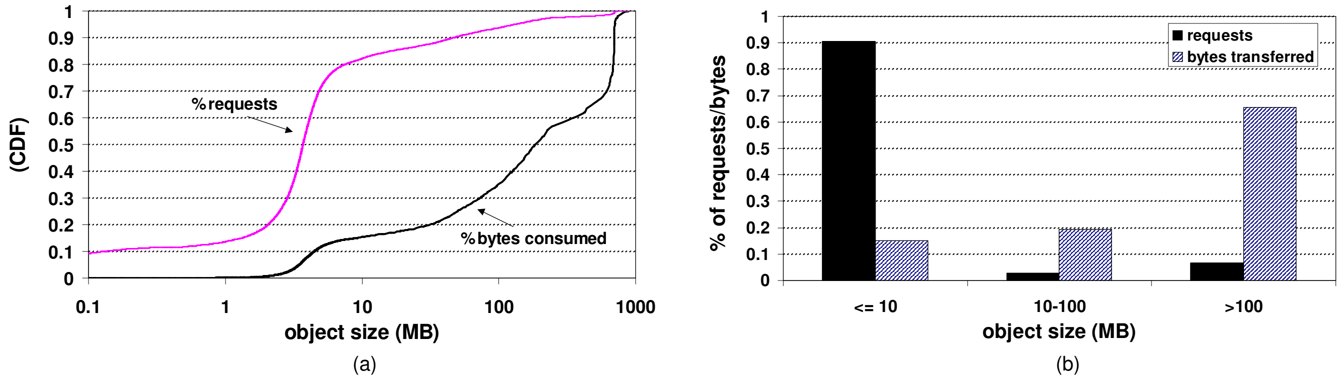

Figure 4: ***Bandwidth consumed vs. object size.*** **(a) CDFs of the total bandwidth consumed and the number of requests generated, as a function of object size, and (b) bandwidth consumed and requests generated as a function of object size, grouped into three regions.**

		all clients	clients with >1 transfer
activity fraction over lifetime	median	66.08%	5.54%
	90^{th} percentile	100%	94.41%
activity fraction over trace	median	0.01%	0.20%
	90^{th} percentile	2.29%	4.15%
total activity over trace	median	0.69 hours	10.09 hours
	90^{th} percentile	111.93 hours	202.65 hours
average session length	median	2.40 mins	2.40 mins
	90^{th} percentile	28.25 mins	28.33 mins

Table 2: ***Client session lengths and activity fractions.*** **Summary statistics over two different subsets of our client population: all clients, and only those clients that have transferred more than one file. We show activity fraction summary statistics relative to both the clients' lifetimes and to the entire trace.**

Table 2 summarizes these metrics. Average session lengths are typically small: the median average session length is only 2.4 minutes, while the 90th percentile is 28.25 minutes. Most clients have high activity fractions relative to their lifetimes. However, this is a potentially misleading statistic, since many clients have very short lifetimes, leaving the system permanently after requesting only one file. The activity fraction for clients with greater than one request is a more revealing statistic. The median client in this category was active only 5.54% of its lifetime, or 0.20% of the entire trace. Highly active clients in this category are active for most of their lifetime, but even these clients are typically active for only a short fraction of the trace: the 90th percentile client has an activity fraction of only 4.15% over the trace.

At first glance, our user patience statistics may appear at odds with our average session length statistics. The median small object takes about 17 minutes to download fully, but the median average session length is only 2.41 minutes. This apparent paradox is reconciled by several facts: (1) many transactions fail (Table 1), producing many very short sessions, (2) periods of no activity may occur during a request if the client cannot find an available server with the object, and (3) some successful transactions are short, either because servers close connections prematurely or the "chunk" size requested is small.

2.3 Object Characteristics

We now turn our attention to the characteristics of requested objects. Kazaa objects consist of a mixture of media types, including audio, video, images, and executables [29]. Our trace captured 1,640,912 requests to 633,106 different objects, resulting in 98,997,622 transactions. In the remainder of this section, we discuss the most prominent properties of the requested objects.

2.3.1 *Kazaa Is Not One Workload*

While we tend to think of a system like Kazaa as generating a single workload, Kazaa is better described as a blend of workloads that each have different properties. Figure 4(a) shows the percentage of total bandwidth consumed in our trace as a function of object size. The graph shows three prominent regions: "small" objects that are less than 10 MB in size, medium-sized objects from 10 to a few hundred MB in size, and large objects that are close to 1 GB in size. Relative to the Web, our "small" objects are three orders of magnitude larger than an average Web object, while our large objects are nearly five orders of magnitude larger. This was shown in [29] as well.

Figure 4(a) also graphs the percentage of requests observed in the trace as a function of object size, demonstrating that most requests are for the "small" objects. Taking a closer look, Figure 4(b) breaks down the number of requests and bytes transferred into three regions: 10MB or less, 10MB to 100MB, and over 100MB. We chose the value 10MB as this corresponds to an obvious plateau in Figure 4(a) and it serves as an upper bound on the size of most digitized audio clips. We chose the value 100MB as it is a reasonable lower bound on the size of most digitized full-length movies transferred. While the majority of requests (91%) are for objects smaller than 10MB (primarily audio clips), the majority of *bytes* transferred (65%) are due to the largest objects (video files). Thus, if our concern is bandwidth consumption, we need to focus on the small number of requests to those large objects. However, if our concern is to improve the overall user experience, then we must focus on the majority of requests for small files, despite their relatively small bandwidth demand.

The stark differences between the large and small object workloads lead us to analyze their behaviors independently for much of the evaluation that follows.

	small objects (primarily audio)		large objects (primarily video)	
	top 10	top 100	top 10	top 100
overlap in the most popular objects between first and last 30 days of trace	0 of 10	5 of 100	1 of 10	44 of 100
# of newly popular objects that are recently born	6 of 10	73 of 95	2 of 9	47 of 56

Table 3: ***Object popularity dynamics in Kazaa.*** **There is significant turnover in the set of most popular objects between the first 30 days and the last 30 days of the trace. The newly popular objects (those in the set of most popular objects over the last 30 days but not in the set over the first 30 days) tend to be recently born.**

2.3.2 Kazaa Object Dynamics

A simple but crucial difference between multimedia and Web workloads is that *multimedia objects are immutable*, while Web pages are not. Though obvious, this fact and its implications have not been discussed in the research literature. A video clip of "Bambi Meets Godzilla" will be the same clip tomorrow or the next day: it never changes. On the other hand, the Web page CNN.com may change every hour, or upon every access if the page is personalized for clients. Web workloads are thus strongly driven by dynamic content creation. It has been shown that the rate of document change is a key factor in Internet behavior and has enormous implications for caching, performance, and content delivery in general [11, 37]. We now show how immutability affects object dynamics.

Kazaa clients fetch objects at most once. Because objects are immutable and take non-trivial time to download, we believe that users typically download a Kazaa object only once. Our traces confirm that 94% of the time, a Kazaa client requests a given object at most once; 99% of the time, a Kazaa client requests a given object at most twice. In comparison, based on a Web trace we gathered during the first nine days of our Kazaa trace period, a Web client requests a given object at most once only 57% of the time.

The popularity of Kazaa objects is often short-lived. Object immutability also has an impact on object popularity dynamics. The set of most popular pages remains relatively stable for the Web and these pages account for a significant fraction of overall accesses [26]. In contrast, many of the most popular audio/video objects are routinely replaced by newly released objects, often in only a few weeks.

To illustrate this change in popular Kazaa objects, we compared the first 30 days of the trace to the last 30 days of the trace. For each of these 30-day (month-long) segments, we identified the top-10 most popular and the top-100 most popular objects (Table 3). For small objects, there was no overlap between the top-10 most popular objects: the most popular small objects had changed completely in the space of only six months. For large objects, there was only one object in common in the top-10 across these segments. The top-100 objects show only 5 small objects in common across the segments, and 44 objects in common across the large objects. Popularity is fleeting in Kazaa, and popular audio files tend to lose popularity faster than popular video files.

The most popular Kazaa objects tend to be recently born objects. Given that there is significant turnover in popularity within Kazaa, we wanted to understand whether objects that become popular are old objects that have grown in popularity over time or recently born objects that enjoy sudden popularity. Using the same month-long segments as before, we calculated the fraction of objects that were newly popular (i.e., in the top-10 or top-100 in the last month of the trace but not the first month of the trace), but did not receive any requests at all during the first month of the trace (i.e., they were likely "born" after the first month of the trace). Table 3 shows the results: newly popular objects tend to be recently born in Kazaa, although this is more true for audio objects than video objects.

Most requests are for old objects. The previous experiments confirmed that the most popular objects tend to decay in popularity, and that the newly popular objects that replace them tend to be newly born. A related, but different, question is whether most requests go to old or new objects. We categorize an object as "old" if at least a month has passed since we observed the first request for that object. We categorize it as "new" if it has been less than a month since it was first requested. Note that we can be sure that an object is old, but we can never be sure that an object is new, since we may have missed requests for the object before our trace began. To deal with edge effects, we do not include the first month of requests in our statistics, but we do use them to help distinguish between old and new objects in subsequent months.

Using this methodology, 72% of requests for large objects go to old objects, while 28% go to new objects. For small objects, 52% of requests go to old objects, and 48% go to new objects. This shows that a substantial fraction of requests are for the old objects. Large objects requested tend to be older than small objects, reinforcing our assertion that Kazaa is really a mixture of workloads: the pace of life is slower for large objects than for small objects.

From the above discussion, it is clear that the forces driving the Kazaa workload differ in many ways from those driving the Web. Web users may download the same pages many times; Kazaa users tend to download objects at most once. The arrival of new objects plays an important role in P2P file-sharing systems, while changes to existing pages are a more important dynamic in the Web. We discuss implications of these differences in Sections 3 and 4.

2.3.3 Kazaa Is Not Zipf

Much has been written about the Zipf-like qualities of the WWW [7]. In fact, researchers commonly quote the Zipf parameter of the popularity distributions seen in their traces [14, 27], in part to demonstrate that their results are "correct." This Zipf property of Web access patterns is thought to be a basic fact of nature: a small number of objects are extremely popular, but there is a long tail of unpopular requests. Zipf's law states that the popularity of the ith-most popular object is proportional to $i^{-\alpha}$, where α is the "Zipf coefficient" or "Zipf parameter." Zipf distributions look linear when plotted on a log-log scale.

Figure 5 shows the Kazaa object popularity distribution on a log-log scale for large (>100MB) objects, along with the best-fit Zipf curve; a qualitatively similar curve exists for small (<10MB) objects. This figure also shows the popularity distribution of Web objects, drawn from our Web trace. Unlike the WWW, the Kazaa object request distribu-

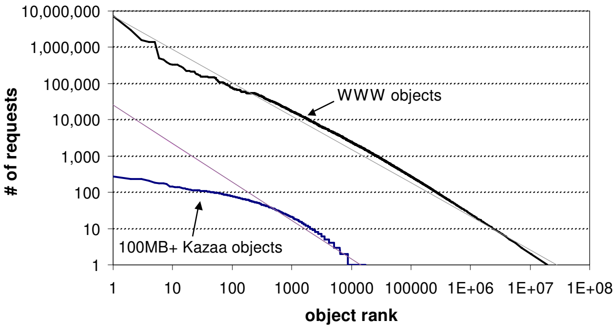

Figure 5: *Kazaa is not Zipf.* The popularity distribution for large objects is much flatter than Zipf would predict, with the most popular object being requested 100x less than expected. Similarly shaped distributions exist for small objects and the aggregate Kazaa workload. For comparison, we show the popularity distribution of Web requests during a subset of the Kazaa trace period: the Web is well described by Zipf.

tion as observed in our trace does *not* seem to follow a Zipf curve. The strongest difference from Zipf appears among the *most* popular objects, where the curves are noticeably flatter than the Zipf lines. The most popular multimedia objects are significantly less popular than Zipf would predict. For example, our trace shows that the most popular large object has only 272 requests, and only 47 objects have 100 requests or more. Popularity distributions of Web traffic sometimes have slightly flattened "heads," but the objects in the flattened region account for only a small percentage of total requests and bytes. The flattened portion of the Kazaa popularity curve consists of the top 1000 objects, and these objects account for more than 50% of bandwidth consumed.

2.4 Summary

Peer-to-peer file-sharing workloads are driven by considerably different processes than the Web. Kazaa users are much more patient than Web users, waiting hours, days, and, in some cases, weeks for their objects to arrive. Even so, the median session length is only 2.4 minutes, because: (1) many transactions fail, (2) many transactions are short, and (3) clients occasionally have periods of no activity while satisfying a request. As Kazaa clients age, they demand less from the system. Part of this is due to attrition (clients permanently leaving the system), but part is also due to older clients requesting fewer bytes per session.

The Kazaa file-sharing workload consists of a mixture of large, immutable audio and video objects, but the measured popularity distribution of these objects is not Zipf. The popularity of the most requested objects is limited, since clients typically fetch objects at most once, unlike the Web. Nonetheless, there is substantial non-uniformity in Kazaa object popularity, which suggests that caching is a potentially effective bandwidth savings tool. In Kazaa, object arrivals play an important role, while in the Web, updates to existing pages are a more important dynamic.

In the next section, we explore what we believe are the reasons why Kazaa's workload does not appear to follow Zipf's law. We also compare Kazaa's non-Zipf behavior to non-Zipf behavior in other systems, including video-on-demand servers and the Web. Following this, we present a model of multimedia workloads in Section 4, and we use this model to explore implications of non-Zipf behavior.

3. ZIPF'S LAW AND MULTIMEDIA WORKLOADS

Previous studies of multimedia workloads examined object popularity and found conflicting results, noting both Zipf and non-Zipf behavior [4, 7, 8, 12, 33]. This section examines previous work in the context of our own, with the goal of explaining the similarities, differences, and causes of the behavior observed both by us and others. We begin by presenting a hypothesis that explains the non-Zipf behavior in Kazaa. Next, we discuss previous studies that have observed or modeled non-Zipf workloads and contrast our hypothesis with previous explanations. Finally, we attempt to show the generality of our claim by revealing non-Zipf behavior in previously studied workloads. Section 4 then supports our hypothesis through the use of a generative workload model whose output closely matches our observations.

3.1 Why Kazaa Is Not Zipf

The previous section highlighted a crucial difference between the Web and the Kazaa P2P file-sharing system: Kazaa objects are immutable, while Web objects change, sometimes frequently. As we observed in Section 2.3.2, the immutability of Kazaa objects causes two important effects, one affecting user behavior and the other object dynamics. First, Web clients often fetch the same Web page many times ("fetch-repeatedly"). In contrast, Kazaa clients rarely request the same object twice ("fetch-at-most-once"). Second, the Web's primary object dynamic is *updates* to existing pages. In contrast, the primary object dynamic in the Kazaa workload is the *arrival* of entirely new objects.

These object and user characteristics can be used as two axes for classifying systems: **immutable objects vs. mutable objects**, and **fetch-repeatedly clients vs. fetch-at-most-once clients**. In systems with large client-side caches, these characteristics are intimately linked: fetch-repeatedly behavior exists when there are object updates, such as in the Web, and fetch-at-most-once user behavior exists when the only object dynamic is new arrivals, such as in the Kazaa system. We believe that these differences in object and user dynamics explain why we observed Zipf request distributions in our measured Web workload, but not in Kazaa.

As an initial comparison of fetch-at-most-once and fetch-repeatedly behavior, we simulated the request distribution generated by two hypothetical 1,000-user populations: one in which users fetch objects repeatedly, and one in which they fetch objects at most once. In both cases, we simulated the same initial Zipf distribution (with Zipf parameter $\alpha = 1.0$ over 40,000 objects). However, in the fetch-at-most-once case, we prevented users from making subsequent requests for the same object. (Section 4 more fully explains the model behind this simulation.)

Figure 6 shows the results when users have made an average of 1000 requests each. While fetch-repeatedly behavior recreates the underlying Zipf distribution, fetch-at-most-once behavior shows markedly different results: the most popular objects are requested much less, while objects down the tail show elevated numbers of requests. This behav-

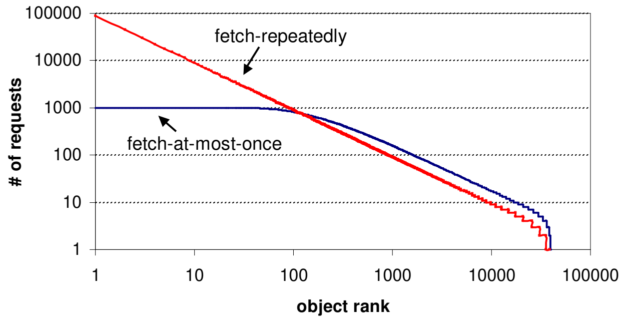

Figure 6: ***Simulated object popularity for fetch-repeatedly and fetch-at-most-once behavior.*** **When users fetch an individual object at most once, the resulting request distribution has a significantly flattened head, approximating the Kazaa popularity distribution.**

ior strikingly resembles the measurement results shown in Figure 5, confirming our intuition that fetch-at-most-once behavior is a key contributor to non-Zipf popularity.

3.2 Non-Zipf Workloads in Other Systems

Non-Zipf popularity distributions have been observed in many previous studies [1, 3, 8, 33]. We now compare our results to two categories of previous studies: those that analyzed Web workloads and those that analyzed video (or other multimedia) workloads.

3.2.1 Kazaa vs. Web Workloads

While most Web studies conclude that Web popularity distributions follow Zipf's law, some studies have observed non-Zipf behavior, specifically in workloads that emerge as miss traffic from a proxy cache [4, 7, 12]. Web proxy caches, by design, absorb requests to popular documents; consequently, the most popular documents in a workload will appear only once (as cold misses) in cache miss traces. Moderately popular documents will occasionally be absorbed by the proxy cache, but they will also appear in the miss traffic as capacity misses. The least popular documents will always miss in the proxy cache, assuming the cache has finite size and cannot hold the entire object population. As a result, a Zipf workload, when fed through a shared proxy cache, will result in a non-Zipf popularity distribution with a flattened head [12], visually similar to the non-Zipf distributions we observed in Kazaa. Breslau et al. [7] derive equations that relate the Zipf parameter of a workload fed to a proxy cache and the hit rate that the proxy cache will experience.

Although proxy cache miss distributions may visually resemble our measured Kazaa popularity distributions, we believe the two workloads diverge from Zipf for different reasons. In Kazaa, users' requests do not flow through a shared proxy cache: the workload we observed is the aggregate of a large number of individual fetch-at-most-once streams, with an absence of shared proxy caches. The only caches that currently exist in Kazaa are client-side caches. Note that there is no practical difference between true fetch-at-most-once user behavior, and fetch-repeatedly user behavior filtered through infinite client-side caches resulting in fetch-at-most-once client behavior.

The most popular Web object will show up only one time in a proxy cache miss trace. The most popular object will show up many times (potentially as many times as there are users) in a fetch-at-most-once workload such as Kazaa, even with individual client caches. Both result in non-Zipf distributions, but for different reasons.

3.2.2 Kazaa vs. Other Multimedia Workloads

Tang et al. [33] show that streaming media server workloads exhibit flattened popularity distributions similar to those we observed. They attribute this flattened distribution to the long duration over which they gathered their trace. Their hypothesis is that with a longer trace, more files with similar popularity can be observed, in effect creating "groups" of files with similar properties. Instead of Zipf's law being appropriate for describing the popularity of individual files, they show that Zipf's law better describes the popularity of these groups, and they provide a mathematical transformation inspired by these groups to convert observed non-Zipf distributions into a Zipf-like distribution.

We believe that fetch-at-most-once behavior is the likely cause of the non-Zipf popularity in their workload. A short trace can cause a non-Zipf workload to appear Zipf-like, as not enough requests have been gathered within the trace for objects with similar popularity to be observed [8]. However, the fact that an adequate sample size will reveal a non-Zipf workload does not explain why this true workload is non-Zipf in the first place. The length of a trace by itself does not provide a satisfactory explanation of the forces driving popularity in a system. Fetch-at-most-once behavior partially provides this explanation. Additionally, in a system that experiences object births (for example, new video objects becoming available), a short trace may miss important birth events. A longer trace may reveal objects spread out over time that have equivalent short-term popularity, similar to the "group" effect Tang et al. proposed.

In another study, Almeida et al. [3] observe a flattened Zipf-like popularity distribution in an educational media server. They propose that this distribution is well described by the concatenation of two true Zipf distributions. Although the data appears to be well modeled by two Zipf distributions, their study does not provide an explanation of what causes this effect.

Another often studied multimedia workload is that of video-on-demand (VoD) servers. VoD researchers frequently use Zipf distributions to model the popularity of video documents in their systems [10, 18, 31]. Surprisingly, many of these researchers appear to base their Zipf assumption on results published from a single data set: one week of rental data from a video store [35].

Our results suggest that their workload (i.e., requests for immutable video objects) should reveal similar non-Zipf behavior stemming from fetch-at-most-once client behavior. To understand the apparent discrepancy between their assumptions and our results, we manually extracted the video rental data set from Figure 2 of [10]. In Figure 7(a), we show this data as it appeared in that paper, plotted on a linear scale with a Zipf curve fit. Figure 7(b) shows the same data plotted on a log-log scale with the same Zipf curve fit. While the linear scale plot seems to suggest the data may be well described by Zipf's law, the log-log plot reveals that even this data set appears to show the flattened head that is characteristic of fetch-at-most-once systems. We also gathered recent box-office movie ticket sales data [34]. Figures 7(c) and (d) show that this data, too, is consistent with

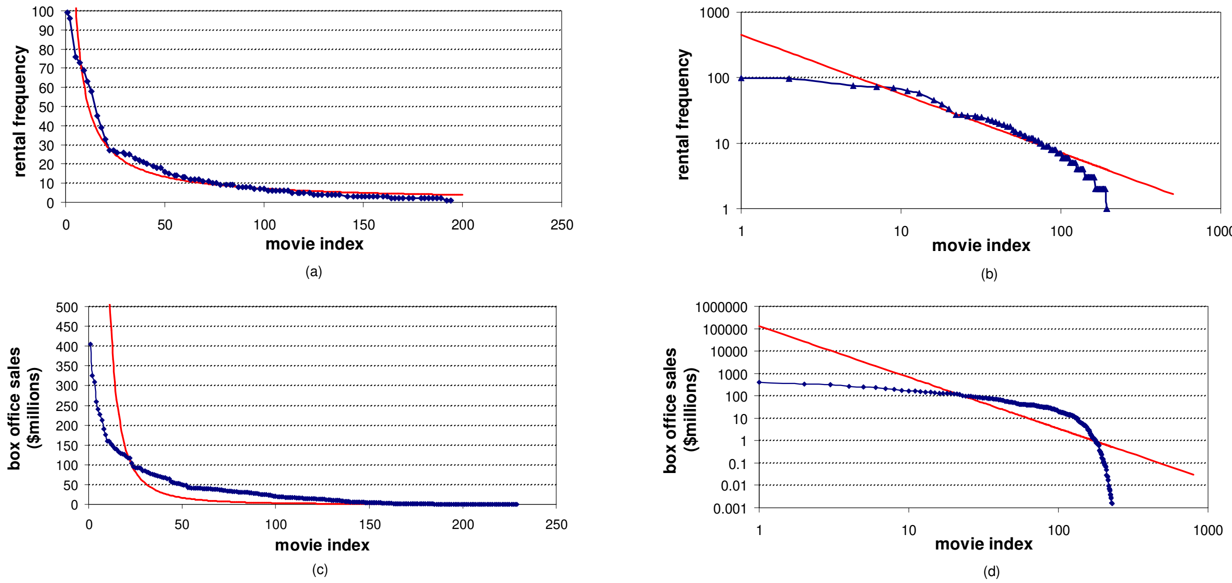

Figure 7: ***Video rental and box office sales popularity.*** **(a) The popularity distribution from a 1992 video rental data set used to justify Zipf's law in many video-on-demand papers, along with a Zipf curve fit with $\alpha = 0.9$, and (b) the same data set and curve fit plotted on a log-log scale. Contrary to the assumption of many papers, video rental data does not appear to follow Zipf's law. (c) The distribution of 2002 U.S. box office ticket sales on a linear scale, along with a Zipf-fit with $\alpha = 2.28$, and (d) on a log-log scale. This data set also appears to be non-Zipf.**

our fetch-at-most-once popularity distribution.

The VoD community has recently begun to recognize the importance of object births in video-on-demand systems [16]. Because new video titles are created (often with high popularity), and because the popularity of existing video titles tends to diminish over time, VoD workload models are starting to incorporate schemes for changing popularity distributions over time. This is consistent with our earlier observations that both client and object birth rates are important factors in multimedia workloads.

In the next section, we present a model of fetch-at-most-once systems. We use this model to demonstrate the implications of the resulting non-Zipf behavior and explore the importance of client and object birth rates on system performance. We believe that this model is relevant to both P2P file-sharing and video-on-demand systems.

4. A MODEL OF P2P FILE-SHARING WORKLOADS

In the previous section, we hypothesized that the non-Zipf behavior of Kazaa, and of multimedia workloads in general, is best explained by the fetch-at-most-once behavior of clients. This section presents a model of P2P file-sharing systems that enables us to explore this hypothesis further. The model, used to generate Figure 6 in the previous section, captures key parameters of the P2P file-sharing workload, such as request rates, number of clients, number of objects, and changes to the set of clients and objects. From these parameters, it produces a stream of client requests for objects that we then analyze. By varying model parameters, we explore the underlying processes that lead to certain observable workload characteristics and how changes to the parameters affect the system and its performance.

The remainder of this section describes our basic model of a P2P file-sharing system. This model shows the impact that fetch-at-most-once behavior has on overall object popularity. We use caching as a lens to observe how the addition of new objects and new clients affects performance in a fetch-at-most-once file-sharing system. Finally, we validate the model by comparing the popularity distributions of a model-generated workload with that extracted from our trace data.

Symbol	Meaning	Base value
C	# of clients	1,000
O	# of objects	40,000
λ_R	per-user request rate	2 objects/day
α	Zipf parameter driving object popularity	1.0
P(x)	probability that a user requests an object of popularity rank x	based on Zipf(1) *(see text)*
A(x)	probability that a newly arrived object is inserted at popularity rank x	Zipf(1)
M	cache size, measured as fraction of all objects	varies
λ_O	object arrival rate	varies
λ_C	client arrival rate	varies

Table 4: ***Model structure and notation.*** **These parameter settings reflect the values seen in our trace for large (>100MB) objects.**

4.1 Model Description

Table 4 summarizes the parameters used in our model. We chose parameter values that reflect our trace measurements for large objects, since this component dominates bandwidth consumption. One parameter value that differs from the measured trace is the number of clients. In order to run a substantially larger number of experiments, we model a system with 1,000 clients rather than the roughly 7,000 large-object clients in our trace. We verified that the predictions of our model were not affected by this difference. To simplify our model, we also assumed that all objects in the system were of equal size.

Our model captures key aspects of our P2P file-sharing workload, in particular, the differences between file-sharing and Web workloads. In a Web workload, clients select objects from a Zipf distribution, $P(x)$, in an independent and identically distributed fashion. In contrast, the object selection process in a file-sharing system depends on three factors: (1) the Zipf distribution, $P(x)$, (2) the way in which *new* objects are inserted into that distribution, $A(x)$, and (3) the clients' fetch-at-most-once behavior.

Our model generates requests as follows. On average, a client requests two objects per day, choosing which object to fetch from a Zipf probability distribution with parameter 1.0 ("Zipf(1)").[4] We hypothesize that the underlying popularity of objects in a fetch-at-most-once file-sharing system is *still* driven by Zipf's law, even though the observed workload becomes non-Zipf because of fetch-at-most-once clients. In our model, subsequent requests from the same client obey distributions obtained by removing already fetched objects from the candidate object set and re-scaling so the total probability is 1.0. Given two previously unrequested objects, the ratio of the probabilities that the client will request these objects next is identical to their ratio in the original Zipf distribution. For fetch-repeatedly systems, each request is made according to the original Zipf distribution.

When modeling fetch-at-most-once systems, $\lambda_O > 0$ is the object arrival rate. When an object is born in a fetch-at-most-once system, its popularity rank is determined by selecting randomly from the Zipf(1) distribution. Pre-existing objects of equal or lesser popularity are "pushed down" one Zipf position, and the resulting distribution is re-normalized so the total probability is again 1.0. In fetch-repeatedly systems, we set the object arrival rate to 0. Objects may be updated, but for simplicity we ignore the second-order effect of completely new objects on request behavior.

While our trace shows all requested objects, it cannot observe the total object population, since many available objects were never accessed. However, total object population is a key parameter of our model, as it influences the amount of overlap that will likely occur in requests from different clients. We therefore estimated a base value for total object population by back-inference: how many large objects are most likely to have existed in total, given that we saw about 18,000 distinct large objects requested in the trace? We find that a total population of about 40,000 large-media objects is consistent with the trace data; therefore, we use this number as the base value. This number is also comparable to statistics that describe commercial movie releases: the Internet Movie Database reports between 50,000 and 60,000 movie releases world-wide over the past 20 years [34].

To quantify file-sharing effectiveness, we use the hit rate that the aggregate workload experiences against a 100% available shared cache with LRU replacement, whose size we vary in each experiment. Selected experiments using optimal replacement showed no qualitative differences from LRU results, and quantitative differences varied by only a few percent. For Web (fetch-repeatedly) scenarios, we make

[4]Attempting to best-fit a Zipf curve to our measured non-Zipf distribution resulted in a Zipf parameter of 0.98.

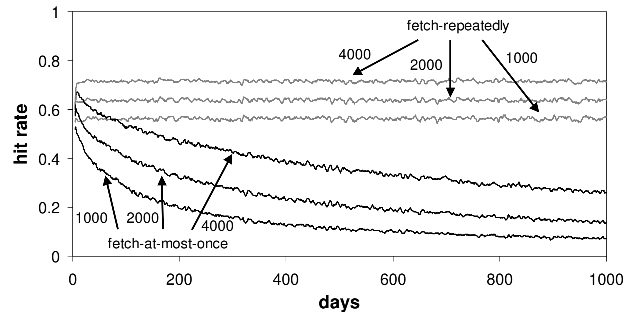

Figure 8: ***File-sharing effectiveness diminishes with client age.*** **After an initial cache warm-up period, the request stream generated by fetch-at-most-once client behavior experiences rapidly decreasing cache performance, while the stream generated by fetch-repeatedly clients remains stable. This effect is shown for various cache sizes (1000, 2000, and 4000 objects).**

the optimistic assumption that all objects are cachable and are never updated.

4.2 File-Sharing Effectiveness Diminishes with Client Age

Imagine an organization experiencing current demand for external bandwidth due to fetches of P2P file-sharing objects. How should this organization expect bandwidth demand to change over time, given a shared proxy cache? We address this question using the "static" case in which no new clients or objects arrive. This static analysis allows us to focus on one factor at a time (fetch-at-most-once behavior, in this case). We relax these assumptions in later subsections to show the impact of other factors, such as new object and client arrivals.

Figure 8 shows hit rate against time for various shared cache sizes, assuming that at time zero no clients have fetched any objects. After a brief cache warm-up period, hit rate decreases as clients age, even for a cache that can hold 10% of all file-sharing objects. This is because fetch-at-most-once clients consume the most popular objects early. Later requests are selected with nearly uniform likelihood from an increasingly large number of objects. That is, the system evolves towards one in which there is no locality, and objects are chosen at random from a very large space.

This behavior suggests that if client request rates remain constant over time, the external bandwidth load they present *increases*, since more of their requests are directed to objects that are only available externally. Conversely, if we hope to have stable bandwidth demand over time, clients must behave in a way that reduces the intensity of their requests in a manner consistent with the shape of the hit-rate decreases shown in Figure 8.

The decrease in hit-rate over time is a strong property of fetch-at-most-once behavior. The underlying popularity distribution need not be heavy tailed for it to occur. We have performed experiments using initial object popularity distributions that have higher locality (i.e., Zipf parameters larger than 1.0). In a fetch-repeatedly context, the larger skew of these distributions towards the most popular objects makes file sharing easier and hit rates rise. For fetch-at-most-once systems, hit rates start out much higher, but as clients age

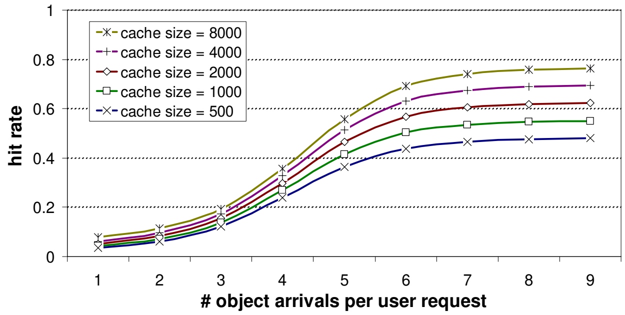

Figure 9: ***Object arrivals improve performance.*** **Cache hit rates improve with new object arrival rates for fetch-at-most-once clients because they replenish the supply of popular objects. This improvement is shown across varying cache sizes (500 to 8000 objects). The X-axis shows the global object arrival rate relative to the average client's request rate. Since the average client makes 2 requests per day, the point $x = 1$ implies that 2 new objects arrive globally per day.**

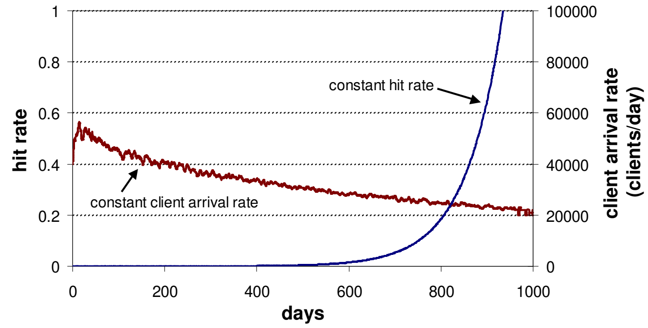

Figure 10: ***Client arrivals cannot stabilize performance.*** **With** ***constant client arrival rate*****, hit-rate decreases with time. To maintain** ***constant hit-rate*****, client arrival rate must increase exponentially with time.**

they fall off even more sharply than in Figure 8. Thus, even if the file-sharing system evolves in a way that increases the popularity of their most requested objects, the hit rate of existing clients on existing objects will still decrease towards zero as clients age.

4.3 New Object Arrivals Improve Performance

The decay of hit rate with client age explains another surprising characteristic of P2P file-sharing systems: while Web performance suffers due to object updates, object arrivals are actually *beneficial* in a file-sharing system. This is because arrivals replenish the supply of popular objects that are the source of file-sharing hits.

Figure 9 shows this effect. We repeated the previous simulation, but this time introduced a non-zero object arrival rate. Over any realistic range of arrival rate[5], hit rates increase, approaching at maximum the hit rate of an equivalent fetch-repeatedly system. For parameters set to the base values of our model (in which clients average two requests per day), an object arrival rate as small as twelve new objects introduced worldwide per day compensates for nearly all the loss due to client aging.

The arrival of new objects in a P2P file-sharing system is therefore an important rejuvenating force that counterbalances fetch-at-most-once behavior. Without new popular objects to choose from, existing clients quickly exhaust the set of popular objects, after which they are forced to choose from the remaining heavy tail of unpopular objects. Without the infusion of new objects, the workload in a fetch-at-most-once system loses its locality over time.

4.4 New Clients Cannot Stabilize Performance

Because new clients have higher hit rates than old clients, it might be possible for new clients joining a P2P file-sharing system to compensate for the performance loss due to the aging of existing clients. The infusion of new clients may therefore be an equivalent rejuvenating force to the infusion of new objects. Unfortunately, in any practical sense, this turns out not to be the case.

Figure 10 shows two different results. First, we examine hit-rate over time when new clients are introduced at a constant rate. Initially, new clients bring up the hit-rate, but eventually the constant arrival rate cannot compensate for the increasing numbers of old clients. Second, Figure 10 shows an estimate of the arrival rate needed to keep hit rate constant as the system ages. An arrival rate above this line improves the average hit rate, while one below allows it to decrease. This "break-even arrival rate" rises steeply over time, too steeply to be realized in practice over more than a short period. That, plus the fact that the overall bandwidth requirement increases in proportion to population size even when hit rate is stable, leads us to conclude that the introduction of new clients cannot compensate for the hit-rate penalty of clients aging in a P2P file-sharing system.

4.5 Model Validation

A primary goal of our model was to capture the peculiar characteristics of fetch-at-most-once systems and the importance of new object and client arrivals. While we are confident about many aspects of our model, some of our assumptions cannot be validated against trace data. For example, we assume that client requests are governed by a Zipf distribution and that object arrivals also obey a variant of Zipf's law in terms of where they are placed in the overall popularity distribution. Even though we cannot directly verify these assumptions, we can verify that the observed behavior in our trace is consistent with our model.

We validated our model by using it to replicate the object popularity distribution measured in our trace. To do this, we parameterized the model from the trace data to the extent possible and then compared the popularity distributions generated by the model (using simulation) with those observed in the trace. We emphasize that we are *not* driving the simulation with the detailed trace and simply getting out what we put in. The simulation is driven synthetically from our model, with rate and size parameters set from the average values measured in the trace. We set distributional parameters that cannot be obtained from any trace to the values shown in Table 4.

It is not possible to set λ_O (the arrival rate of new objects) with any confidence from our trace data, since our trace can-

[5]In the limit, when object arrival rate is high enough, cache hit rate goes back to zero: objects arrive so fast that they are displaced by even newer objects before two client requests can be made.

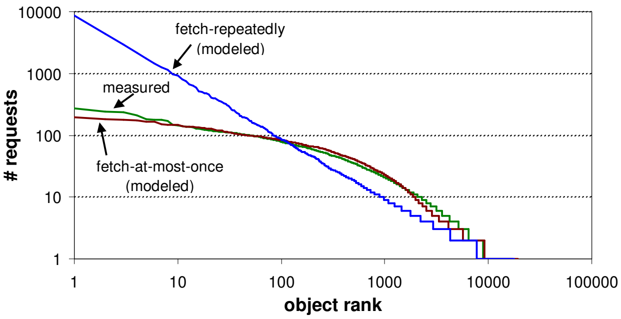

Figure 11: ***Predicted versus measured object popularity.*** **The popularity curves from our model and the actual trace data match remarkably well. This supports our conjecture that the measured popularity distribution is non-Zipf because of** *fetch-at-most-once* **behavior.**

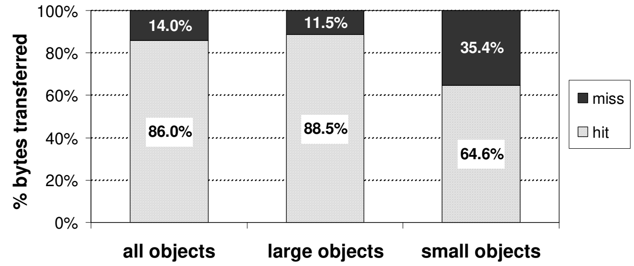

Figure 12: ***Bandwidth savings with an ideal proxy cache.*** **This graph shows the byte hit rate for a simulated ideal cache (i.e., infinite capacity and bandwidth).**

not measure the worldwide introduction of new objects. For that reason, we leave object arrival rate as a free parameter, adjusting it to obtain as tight a correspondence between the model and measured data as possible.

Figure 11 shows the results. With λ_O set to 5,475 new objects per year, the popularity distribution predicted by the model is remarkably close to what we observed. It also clearly deviates from Zipf because of the influence of fetch-at-most-once behavior and object arrivals. The value $\lambda_O = 5,475$ is reasonable; in comparison, the Internet Movie Database tracked approximately 10,606 new objects worldwide during 2002.

4.6 Summary

This section developed a model of P2P file-sharing behavior, with client requests based on an underlying Zipf distribution. Based on this model, our analysis shows that:

1. Fetch-at-most-once client behavior, caused by the immutability of objects in P2P file-sharing systems, leads to a significant deviation from Zipf popularity distributions.
2. As a result, without the introduction of new objects and clients, P2P file-sharing system performance decreases over time (bandwidth demands rise), because client requests "slide down the Zipf curve."
3. The introduction of new objects in P2P file-sharing systems acts as a rejuvenating force that counter-balances the impact of fetch-at-most-once client behavior.
4. Introducing new clients does not have a similar effect, because they cannot counteract the hit rate penalty of client aging, which occurs at the same rate.

The next section examines a scheme for reducing the external bandwidth consumption predicted by our model and shown by our measurements.

5. EXPLORING LOCALITY-AWARE REQUEST ROUTING

Previous studies have shown that a significant fraction of Internet bandwidth is consumed by Kazaa file-sharing traffic [29]. As a result, many organizations now curb P2P file-sharing bandwidth consumption through shaping or filtering tools. This section explores an alternative strategy, namely, the exploitation of *locality* in the file-sharing workload. By locality exploitation, we mean the more effective use of content available within an organization to substantially decrease external bandwidth usage. We begin by using a cache simulation to show the potential for locality exploitation and then explore the benefits of locality-aware P2P file-sharing request routing within an organization such as a university.

5.1 Measuring Locality in the Workload

The most common technology for capturing locality in an Internet workload is a proxy cache placed at an organizational border. A proxy guarantees that every object is downloaded into the organization at most once, on the cold miss. Additional requests for a previously downloaded object are then satisfied from the proxy without consuming external bandwidth. Simulating an ideal cache (i.e., infinite capacity and bandwidth) therefore gives us an upper bound on the bandwidth savings of any locality-aware mechanism, because the cache captures and serves all content transferred into the organization.

Figure 12 graphs the byte hit rate for an ideal, centralized proxy cache, given our trace as its workload. Over all objects, a proxy cache would result in an external bandwidth savings of 86%. In our UW environment, this implies that 86% of the downloaded bytes already existed on other UW-local clients at the time they were downloaded from UW-external clients. It is therefore very clear that substantial untapped locality exists in the Kazaa workload that Kazaa does not exploit. If the university deployed an internal proxy cache for P2P file-sharing content, it would save substantial bandwidth, and therefore money.

In practice, IT departments may not wish to support a cache that stores P2P file-sharing content, given the current legal and political problems this could present. For this reason, we explore an alternative to the deployment of a centralized proxy cache: the use of organization-based locality-aware mechanisms for reducing external downloads. These schemes favor organization-internal peers whenever possible to serve data, effectively creating a distributed cache from local peers. There are many potential implementations of such a locality-aware architecture, including:

1. **Centralized request redirection:** instead of deploying a cache, an organization could deploy a *redirector* at its boundary. The redirector would index the locations of objects on peers within the organization and route internal clients' requests to other inter-

nal peers whenever possible. The redirector should be transparent to the P2P file-sharing protocols.

2. **Decentralized request redirection:** today's P2P file-sharing systems often employ the use of *supernodes*, distinguished peers that index content on other peers. Current architectures such as Kazaa are locality-unaware, as our data shows. Through the use of topological distance estimation techniques such as GNP [25], IDMaps [13], or King [17], it may be possible to infuse supernodes with locality awareness, resulting in a fully distributed redirection architecture.

The following sections use trace-based simulation to assess the potential benefits of these locality-aware mechanisms.

5.2 Methodology

We use trace-based simulation to evaluate a locality-aware scheme in which all requests from clients in the University of Washington are redirected (when possible) to other university peers. Our simulated locality-aware mechanism is ideal, in that it has perfect knowledge about which peers are currently up and which objects each peer is willing to serve. We assume that: (1) all peers have infinite storage capacity, (2) once a peer downloads an object, it makes that object available to other peers when it is up, and (3) each peer can serve at most 12 concurrent downloads, a number chosen to approximate the behavior of many P2P file-sharing systems, including Kazaa and Gnutella. In our model, each peer has a finite upload bandwidth of 500 Kb/s that is shared across all of that peer's concurrent uploads; each external transfer has a bandwidth of 100 Kb/s. These values approximate the typical values we saw in our trace. However, we verified that the qualitative results and conclusions of our experiments do not differ across a wide range of simulated internal and external bandwidth settings.

Unlike a proxy cache, a locality-aware mechanism cannot directly control the availability of content. Instead, it relies on peers to make content available. As a result, if an object exists on only one local peer, the object becomes unavailable whenever that peer is disconnected. Effectiveness is therefore limited by *object availability*. Given the crucial importance of availability, we provide a lower bound on the benefits of locality awareness by making an extremely conservative assumption, namely, that peers are available as servers *only* during periods in which our trace shows them to be actively transferring objects. In reality, peers will be much more available than we simulate, so a locality-aware architecture should be able to achieve *at least* the performance benefits that we show below.

The rest of this section explores whether a locality-aware scheme using internal peers can realize object availability, and therefore external bandwidth savings, competitive with a dedicated, centralized proxy cache.

5.3 Benefits of Locality-Awareness

Figure 13 shows the external bandwidth savings that an ideal locality-aware scheme can obtain. The chart accounts for all bytes downloaded by internal peers, broken down according to whether they were hits (successfully redirected to internal peers) or misses (requiring external bandwidth). The chart shows that locality awareness obtains an impressive 68% byte hit rate for large objects and a 37% byte hit rate for small objects, saving about 22.3TB and 1.5TB of external traffic, respectively. These savings occur despite our extremely conservative estimates of peer availability.

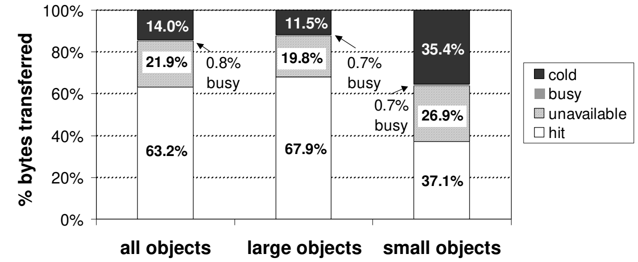

Figure 13: ***Bandwidth savings with ideal locality-aware request redirection.*** **This graph accounts for all bytes transferred to peers when using an ideal locality-aware scheme. A request hits if an available local host can serve it. Otherwise, it misses and downloads from an external host. Misses may be cold misses, busy misses (the object exists locally but all available hosts with it are busy), or unavailable misses (the object exists locally but no hosts with it are available).**

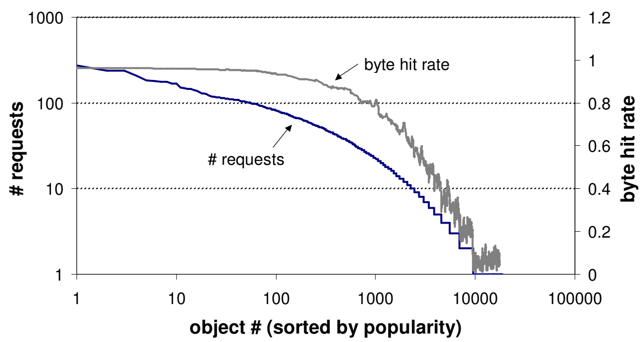

Figure 14: ***Correlating byte hit rate with popularity (large objects).*** **Hit rate and popularity appear to be related: popular objects are "naturally" replicated, leading to a high byte hit rate.**

Cold misses, which result in unavoidable external bandwidth consumption, account for 12% of large object bytes and 35% of small object bytes. Busy misses, in which the object exists on an available internal peer, but that peer is serving its maximum number of concurrent transfers, are insignificant. For both large and small objects, a substantial number of miss bytes are attributable to *unavailable objects*, i.e., the objects exist on local peers that are unavailable when a request occurs. If the locality-aware mechanism could avoid these misses using techniques like data replication or placement, it would reduce the overall miss rate by 62% for large objects and 43% for small objects.

5.4 Accounting for Hits and Misses

To understand which objects account for these unavailability misses, we first explored the relationship between the hit rate an object experiences and that object's popularity. Figure 14 shows the byte hit rate that each large object experiences, plotted on the same graph as object popularity; we see similar results for small objects. Popularity is related to byte hit rate: the more popular an object, the more that object is "naturally" replicated in a P2P file-sharing system.

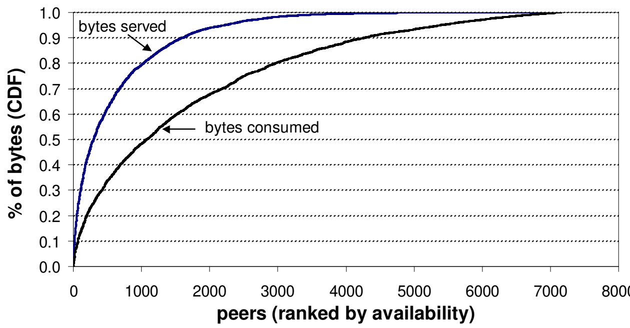

Figure 15: ***Highly available peers carry the load.*** **A CDF of bytes served and bytes consumed by the peers under a locality-aware mechanism, with peers sorted by availability, for large objects. Most of the bytes served and consumed come from highly available peers.**

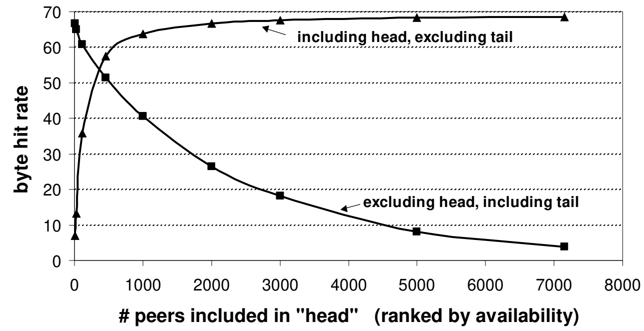

Figure 16: ***Spreading the load around.*** **The "including head, excluding tail" line shows locality-aware performance with load concentrated on the most available peers; the "excluding head, including tail" line shows performance if the most available peers were excluded from serving content. All data are for large objects only.**

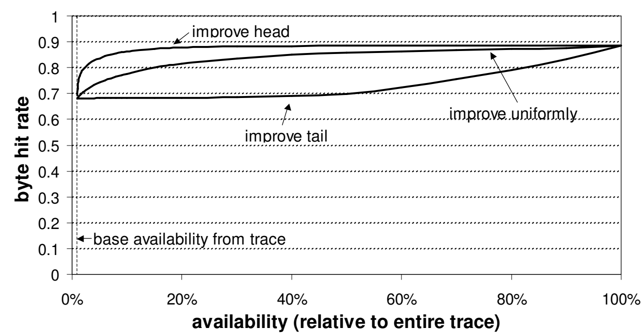

Figure 17: ***Hit rate vs. availability.*** **The effect of increased average peer availability on byte hit rate. The "improve head" line shows the effect of increasing the availability of the most available peers, the "improve tail" line shows the effect of increasing the availability of the least available peers, and the "improve uniformly" line shows the effect of increasing availability uniformly across peers. All data are for large objects only.**

As a result, popular objects enjoy higher hit rates.

The fact that a locality-aware scheme would work well in spite of our conservative peer availability model surprised us. To understand why this happened, we generated a CDF of the bytes served by peers, with peers sorted by availability (Figure 15). This graph shows that most of the bytes were served by the highly available peers, while few bytes were served by the less available peers. We also show a CDF of bytes consumed by peers on the same graph. As expected, the highly available peers tend to consume most of the bytes in the system, as well as serving them. Highly available peers have more objects than less available peers, which is another reason why they may end up serving more bytes.

It is intuitive that more available peers will serve more bytes. However, it is conceivable that the set of less available peers would also be able to provide adequate object availability. To evaluate this, we re-ran our simulation, attempting to "spread" the load to different subsets of the peer population. First, we concentrated the load on the most available peers: the "including head, excluding tail" line on Figure 16 shows the redirector hit rate as a function of the number of peers that we permitted to serve bytes, selecting highly available peers for inclusion in the group. Next, we concentrated the load on the least available peers; the "excluding head, including tail" line shows the hit rate as a function of the number of peers that we excluded from serving bytes, excluding highly available peers first.

Our results indicate that the *highly available peers are both necessary and sufficient* to obtain high hit rates. If only the top 1000 most available peers served bytes, we would still obtain a hit rate of 64%. However, if we excluded the top 1000 available peers and relied only on the least available 6153 peers, our hit rate would drop to 41%.

5.5 Benefits of Increased Availability

To explore the impact that our conservative estimates of availability had on our results, we re-ran our simulations, artificially augmenting the availability of the peers. To do this, we added a constant number of "hours" of availability to the population to increase the overall average peer availability, but we explored spreading this extra availability across the population in different ways.

First, we added the availability to the most available peers in the population, preferentially adding to the most available peer until that peer was 100% available, then adding to the next-most available peer, and so on; the results of this are shown as the "improve head" line in Figure 17. Next, we added availability to the least available peers, using a similar methodology; this is the "improve tail" line. Finally, we spread the availability around the population uniformly across peers; this is the "improve uniformly" line.

The results show that the impact on hit rate depends on which hosts are made more available. Adding an extra hour of availability to the most available host pays a higher hit rate dividend than adding that hour to the least available host. We believe this is because the most available hosts also have more files available, as shown in Figure 15.

5.6 Summary

This section demonstrates that there is a tremendous amount of untapped locality in the Kazaa workload. As a result, a large percentage (86%) of externally downloaded bytes in our workload could be avoided by using an organizational proxy. As an alternative to proxy caching, we used trace-driven simulation to explore locality-aware mechanisms that reduce external bandwidth consumption by maximizing the use of content stored at local peers. Our results show that even with very conservative availability assumptions, a locality-aware P2P file-sharing protocol or an organizational redirector can achieve significant bandwidth reductions in our environment.

6. RELATED WORK

Several measurement studies have characterized the basic properties of peer-to-peer file-sharing systems. Saroiu et al. [30] analyzed the behavior of peers inside the Gnutella and Napster file-sharing systems, showing that there is significant heterogeneity in peers' bandwidth, availability, and transfer rates. A study of AT&T's backbone traffic [32] confirmed these results, and revealed significant skew in the distribution of traffic across IP addresses, subnets, and autonomous systems. Bhagwan et al. [6] measured the availability of hosts in the Overnet file-sharing system to understand the impact of IP aliasing on measurement methodology. Leibowitz et al. [22] performed a cache study based on measurements of FastTrack-based P2P systems, including Kazaa. Several studies [21, 23] have explored how host dynamics within peer-to-peer networks affect performance and reliability. Our Kazaa trace is over a substantially longer time period than most other peer-to-peer file sharing studies, which allows us to draw conclusions about long-term behavior.

Because distributed systems and networks have complex behavior, many researchers have sought to find high-level trends or summary statistics that capture essential properties of their workloads. Breslau et al. [7] explore the impact of Zipf's law with respect to Web caching, showing that Zipf-like popularity distributions cause cache hit rates to grow logarithmically with population size, as well as other effects. In this paper, we perform a similar analysis, demonstrating that Kazaa traffic does not exhibit Zipf-like behavior, and that this has a resulting impact on caching. Crovella and Bestavros [9] argue that several factors converge to cause self-similarity in Web traffic, including document size distributions, caching, and user "think time." In a similar spirit, we show how fetch-at-most-once behavior leads to the flattening of the Zipf curve.

Many researchers have proposed models of Web and file-sharing systems. Barford and Crovella [5] proposed a generative model of Web traffic, based on ON-OFF behavior of Web clients. Ge et al. [15] proposed an analytical model of P2P file-sharing networks and used it to explore the impact of freeloaders on system performance; however, their model focuses on query characteristics only and is not trace-driven. Wolman et al. [38] derived an analytical model of Web systems to explore how Web caching performance scales with population size, and to demonstrate the limits of cooperative Web caching. In our work, we proposed a model of P2P file-sharing traffic based on fetch-at-most-once client behavior and the rate at which objects and clients join the system. While Wolman's study shows that Web caching is ultimately limited by the rate of change of documents, our study shows that file-sharing performance is ultimately limited by the birth rates of objects and clients.

Request redirection has been explored in the context of content-distribution networks, most recently by Wang et al. [36], who show how redirection strategies affect load balancing, locality, and proximity. We consider the effectiveness of request redirection at a different scale (within organizations) and to explore different goals (cache performance).

7. CONCLUSIONS

Peer-to-peer file sharing now dominates all other sources of traffic on the Internet, yet the basic forces that drive this workload are still poorly understood. In this paper, we analyzed a 200-day trace of Kazaa P2P file-sharing traffic collected at the University of Washington in order to dig deeper into the nature of file-sharing workloads. Our results show that P2P file-sharing workloads are driven by considerably different processes than the Web. Kazaa is a batch-mode system with extremely patient users who often wait days, or even weeks, for their objects to fully download. As Kazaa clients age, they demand less from the system, partially because of attrition. The objects that Kazaa users exchange are large, immutable video and audio objects, causing the typical client to fetch any given object at most once. The popularity of Kazaa objects changes over time: popularity tends to be short-lived, and popular objects tend to be recently born.

Based on these results, we conclude that client births and object births are the fundamental processes driving P2P file-sharing workloads; in contrast, the Web is largely driven by changes to objects. We demonstrated that the "fetch-at-most-once" behavior of clients causes the aggregate popularity distribution of objects in Kazaa to deviate substantially from Zipf curves we typically see for the Web.

We also demonstrated that there is significant locality in the Kazaa workload, and therefore substantial opportunity for caching to reduce wide-area bandwidth consumption. We evaluated the impact of topological proximity awareness on Kazaa by simulating an ideal version of the system in which nearby clients act as a distributed cache of Kazaa objects for each other. Even with extremely conservative trace-driven estimates of client availability, our simulation results in a 63% cache hit rate over the population. If deployed in an environment such as the university we traced, a distributed cache would achieve substantial traffic savings.

8. ACKNOWLEDGEMENTS

We wish to thank Brian Youngstrom, who helped us with our tracing infrastructure, and David Richardson, Art Dong, and the other members of the Computing and Communications organization at the University of Washington for their continued support. The guidance of our shepherd, John Wilkes, and our anonymous reviewers was invaluable. We also gratefully acknowledge Tom Anderson, Brian Bershad, Azer Bestavros, Jeff Chase, Mark Crovella, Peter Druschel, Anna Karlin, Scott Shenker, and Andrew Whitaker, whose feedback and discussions sharpened both our research and the presentation of our results. This material is based upon work supported by the National Science Foundation under Grants ITR-0121341 and CCR-0085670 and by a gift from Intel Corporation.

9. REFERENCES

[1] S. Acharya, B. Smith, and P. Parnes. Characterizing user access to videos on the World Wide Web. In *Proceedings of ACM/SPIE Multimedia Computing and Networking*, January 2000.

[2] E. Adar and B. Huberman. Free riding on Gnutella. In *First Monday, 5(10)*, October 2000. http://www.firstmonday.dk/issues/issue5_10/adar/.

[3] J. Almeida, J. Krueger, D. Eager, and M. Vernon. Analysis of educational media server workloads. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSDAV '01)*, Port Jefferson, NY, June 2001.

[4] V. A. F. Almeida, M. G. Cesario, R. C. Fonseca, W. M. Jr., and C. D. Murta. Analyzing the behavior of a proxy server in light of regional and cultural issues. In *Proceedings of the Third International WWW Caching Workshop*, Manchester, England, June 1998. http://hermes.wwwcache.ja.net/events/workshop/.

[5] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS '98*, Madison, WI, June 1998.

[6] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-peer Systems*, Berkeley, CA, December 2002.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM 1999*, March 1999.

[8] L. Cherkasova and G. Ciardo. Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSDAV '02)*, Miami Beach, FL, May 2002.

[9] M. E. Crovella and A. Bestavros. Self-similarity in world wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proceedings of ACM Multimedia 1994*, October 1994.

[11] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Dec. 1997.

[12] R. P. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat. The trickle-down effect: Web caching and server request distribution. In *Proceedings of the Sixth International Workshop on Web Caching and Content Delivery*, Boston, MA, June 2000.

[13] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y.Shavitt, and L. Zhang. IDMAPS: a global Internet host distance estimation service. In *IEEE/ACM Transactions on Networking*, October 2001.

[14] S. Gadde, J. Chase, and M. Rabinovich. Web caching and content distribution: A view from the interior. In *Proc. of the 5th International Web Caching and Content Delivery Workshop*, May 2000.

[15] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proceedings of INFOCOM 2003*, Santa Fe, NM, October 2003.

[16] C. Griwodz, M. Bar, and L. C. Wolf. Long-term movie popularity models in video-on-demand systems. In *Proceedings of ACM Multimedia 1997*, Seattle, WA, November 1997.

[17] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the Second SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, November 2002.

[18] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of ACM SIGCOMM 1997*, Cannes, France, September 1997.

[19] Kazaa. Homepage http://www.kazaa.com, July 2003.

[20] Keynote Systems Inc. Homepage at http://www.keynote.com, July 2003.

[21] J. Ledlie, J. Taylor, L. Serban, and M. Seltzer. Self-organization in peer-to-peer systems. In *Proceedings of the 2002 SIGOPS European Workshop*, St. Emilion, France, September 2002.

[22] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit. Are file swapping networks cacheable? Characterizing P2P traffic. In *Proc. of the 7th Int. WWW Caching Workshop*, August 2002.

[23] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer networks. In *Proceedings of 2002 ACM Conference on the Principles of Distributed Computing*, Monterey, CA, July 2002.

[24] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the Winter USENIX Conference*, pages 259–270, 1993.

[25] E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM 2002*, NewYork, NY, June 2002.

[26] Nielsen Netratings, Inc., August 2003. http://www.nielsen-netratings.com.

[27] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy Web site: Findings and implications. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[28] D. Plonka. University of Wisconsin-Madison, Napster traffic measurement, March 2000. Available at http://net.doit.wisc.edu/data/Napster, March 2000.

[29] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of internet content delivery systems. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[30] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN) 2002*, January 2002.

[31] J. Segarra and V. Cholvi. Distribution of video-on-demand in residential networks. *Lecture Notes in Computer Science*, 2158:50–61, 2001.

[32] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the Second SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, November 2002.

[33] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. Long-term streaming media server workload analysis and modeling. Technical Report HPL-2003-23, HP Laboratories, January 2003.

[34] The Internet Movie Database, August 2003. http://www.imdb.com.

[35] Video Store Magazine, March 2000. Published by Avanstar Communications, http://www.videostoremag.com.

[36] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[37] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based analysis of Web-object sharing and caching. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, Oct. 1999.

[38] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. The scale and performance of cooperative Web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, Dec. 1999.