Traffic Modeling and Proportional Partial Caching for Peer-to-Peer Systems

Mohamed Hefeeda, *Member, IEEE*, and Osama Saleh

***Abstract*—Peer-to-peer (P2P) file sharing systems generate a major portion of the Internet traffic, and this portion is expected to increase in the future. We explore the potential of deploying proxy caches in different Autonomous Systems (ASes) with the goal of reducing the cost incurred by Internet service providers and alleviating the load on the Internet backbone. We conduct an eight-month measurement study to analyze the P2P traffic characteristics that are relevant to caching, such as object popularity, popularity dynamics, and object size. Our study shows that the popularity of P2P objects can be modeled by a Mandelbrot–Zipf distribution, and that several workloads exist in P2P traffic. Guided by our findings, we develop a novel caching algorithm for P2P traffic that is based on object segmentation, and proportional partial admission and eviction of objects. Our trace-based simulations show that with a relatively small cache size, a byte hit rate of up to 35% can be achieved by our algorithm, which is close to the byte hit rate achieved by an off-line optimal algorithm with complete knowledge of future requests. Our results also show that our algorithm achieves a byte hit rate that is at least 40% more, and at most triple, the byte hit rate of the common web caching algorithms. Furthermore, our algorithm is robust in face of aborted downloads, which is a common case in P2P systems.**

***Index Terms*—Internet measurement, network protocols, peer-to-peer systems, traffic modeling, traffic analysis.**

I. Introduction

PEER-TO-PEER (P2P) file-sharing systems have gained tremendous popularity in the past few years. More users are continually joining such systems and more objects are being made available, enticing even more users to join. Currently, traffic generated by P2P systems accounts for a major fraction of the Internet traffic [1], and it is expected to increase [2]. The sheer volume and expected high growth of P2P traffic have negative consequences, including: (i) significantly increased load on the Internet backbone, hence, higher chances of congestion; and (ii) increased cost on Internet Service Providers (ISPs) [3], hence, higher service charges for all Internet users. A potential solution for alleviating those negative impacts is to *cache* a fraction of the P2P traffic such that future requests for the same objects could be served from a cache in the requester's autonomous system (AS).

Manuscript received October 26, 2006; revised June 16, 2007, and October 28, 2007. First published June 10, 2008; current version published December 17, 2008. Approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Nain. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant #313083 and RTI Grant #344619.

M. Hefeeda is with the School of Computing Science, Simon Fraser University, Surrey, BC, V3T QA3 Canada. He is also with Mansoura University, Mansoura, Egypt (e-mail: mhefeeda@cs.sfu.ca).

O. Saleh is with Eyeball Networks Inc., Vancouver, BC, Canada.

Digital Object Identifier 10.1109/TNET.2008.918081

Caching in the Internet has mainly been considered for web and video streaming traffic, with little attention to the P2P traffic. Many caching algorithms for web traffic [4] and for video streaming systems [5] have been proposed and analyzed. Directly applying such algorithms to cache P2P traffic may not yield the best cache performance, because of the different traffic characteristics and caching objectives. For instance, reducing user-perceived access latency is a key objective for web caches. Consequently, web caching algorithms often incorporate information about the cost (latency) of a cache miss when deciding which object to cache/evict. Although latency is important to P2P users, the goal of a P2P cache is often focused on the ISP's primary concern; namely, the amount of bandwidth consumed by large P2P transfers. Consequently, the byte hit rate, i.e., the number of bytes served from the cache to the total number of transferred bytes, is more important than latency. Moreover, P2P objects tend to be larger than web objects [1] reducing the number of complete objects that can be held in a cache.

Furthermore, although objects in P2P and video streaming systems share some characteristics, e.g., immutability and large size, streaming systems impose stringent timing requirements. These requirements limit the flexibility of caching algorithms in choosing which segments to store in the cache. Therefore, new caching algorithms that consider the new traffic characteristics and system objectives need to be designed and evaluated.

In this paper, we first develop a deeper understanding of the P2P traffic characteristics that are *relevant to caching*, such as object popularity, popularity dynamics and object size. We do that via an eight-month measurement study on a popular file-sharing system. Then, we design and evaluate a novel P2P caching algorithm for object admission, segmentation and replacement. Similar to web proxy caching, our algorithm would be used by caches deployed at the gateway routers of ASes or ISP networks that opt to use caching to reduce the burden of P2P traffic. No cooperation among caches is assumed in this paper.

Specifically, our contributions can be summarized as follows. First, we develop new models for P2P traffic based on our measurement study. We show that the popularity of P2P objects can be modeled by a Mandelbrot–Zipf distribution, which is a generalized form of Zipf-like distributions with an extra parameter. This extra parameter captures the flattened head nature of the popularity distribution observed near the lowest ranked objects in our traces. The flattened head nature has also been observed by a previous study [1], but no specific distribution was given. We also quantify the degree of popularity dynamics and the multi-workload nature of P2P traffic. Second, we analyze the impact of the Mandelbrot–Zipf popularity model on caching and

show that relying on object popularity alone may not yield high hit rates/byte hit rates. Third, we design a new caching algorithm for P2P traffic that is based on segmentation, partial admission and eviction of objects.

We perform trace-based simulations to evaluate the performance of our algorithm and compare it against common web caching algorithms, such as LRU, LFU and GDSP [6], [7], and a recent caching algorithm proposed for P2P systems [8]. Our results show that with a relatively small cache size, a byte hit rate of up to 35% can be achieved by our algorithm, which is close to the byte hit rate achieved by an off-line optimal algorithm with complete knowledge of future requests. Our results also show that our algorithm achieves a byte hit rate that is at least 40% more, and at most triple, the byte hit rate of the common web caching algorithms.

The rest of this paper is organized as follows. In Section II, we summarize the related work. Section III describes our measurement study, presents a new model for object popularity, and analyzes the effect of this model on cache performance. Our P2P caching algorithm is described in Section IV. We evaluate the performance of our algorithm using trace-based simulation in Section V. Section VI concludes the paper.

II. Related Work

We first summarize previous P2P measurement studies, justifying the need for a new study. Then, we contrast our caching algorithm with other P2P, web, and multimedia caching algorithms.

Several measurement studies have analyzed various aspects of P2P systems. Gummadi *et al.* [1] study the object characteristics of P2P traffic in Kazaa and show that P2P objects are mainly immutable, multimedia, large objects that are downloaded at most once. The study demonstrates that the popularity of P2P objects does not follow Zipf distribution, which is usually used to model the popularity of web objects [9]. The study provides a simulation method for generating P2P traffic that mimics the observed popularity curve, but it does not provide any closed-form models for it. Sen and Wang [10] study the aggregate properties of P2P traffic in a large-scale ISP, and confirm that P2P traffic does not obey Zipf distribution. Their observations also show that few clients are responsible for most of the traffic. Klemm *et al.* [11] use two Zipf-like distributions to model query popularity in Gnutella. Because the authors are mainly interested in query popularity, they do not measure object popularity as defined by actual object transfers.

While these measurement studies provide useful insights on P2P systems, they were not explicitly designed to study caching P2P traffic. Therefore, they did not focus on analyzing the impact of P2P traffic characteristics on caching. The study in [1] highlighted the potential of caching and briefly studied the impact of traffic characteristics on caching. But the study was performed in only one network domain.

The importance and feasibility of caching P2P traffic have been shown in [12] and [3]. The study in [12] indicates that P2P traffic is highly repetitive and responds well to caching. Whereas the authors of [3] show that current P2P protocols are not ISP-friendly, because they impose unnecessary traffic on ISPs. The authors suggest deploying caches or making P2P protocols locality-aware. Neither [12] nor [3] provide any algorithm for caching.

The closest work to ours is [8], where two cache replacement policies for P2P traffic are proposed. These two policies are: MINRS (Minimum Relative Size), which evicts the object with the least cached fraction, and LSB (Least Sent Byte), which evicts the object that has served the least number of bytes from the cache. Our simulation results show that our algorithm outperforms LSB, which is better than MINRS according to the results in [8]. In addition, a few commercial P2P caching products have already made it to the market, such as CacheLogic [13], PeerCache [14], and Sandvine [15]. This highlights the importance and timeliness of the problem addressed in this paper. It is not possible, however, to compare our work against these commercial products, because we have no access to their algorithms or implementations.

Partial and popularity-based caching schemes for web caching, e.g., [6], and video streaming, e.g., [16], [17] have been proposed before. [6] proposes a popularity-aware Greedy-Dual-Size algorithm for caching web traffic. Because the algorithm focuses on web objects, it does not consider partial caching, which is critical for P2P caching due to large sizes of objects. Jin *et al.* [16] consider partial caching based on object popularity, encoding bit rate, and available bandwidth between clients and servers. Their objectives are to minimize average start-up delays and to enhance stream quality. In contrast, our partial caching approach is based on the number of bytes served from each object normalized by its cached size. This achieves our objective of maximizing the byte hit rate without paying much attention to latency. A partial caching algorithm for video-on-demand systems is proposed in [17], where the cached fraction of a stream is proportional to the number of bytes played back by all clients from that stream in a time slot. Unlike our algorithm, the algorithm in [17] periodically updates the fractions of *all* cached streams, which adds significant overhead on the cache. The algorithms in [16], [17] are not usable in caching P2P traffic because they require several inputs that are not available in P2P systems.

Finally, our caching algorithm is designed for P2P systems, which contain multiple workloads corresponding to various types of objects. This is in contrast to the previous web and streaming caching algorithms which are typically optimized for only one workload.

III. Modeling P2P Traffic

We are interested in deploying caches in different autonomous systems (ASes) to reduce the WAN traffic imposed by P2P systems. Thus, our measurement study focuses on measuring the characteristics of P2P traffic that would be observed by these *individual* caches, and would impact their performance. Such characteristics include object popularity, popularity dynamics and object size. We measure these characteristics in several ASes of various sizes. In this section, we describe our measurement methodology and present our findings.

A. Measurement Methodology

We conduct a *passive* measurement study of the Gnutella file-sharing network [18]. For the purposes of our measurement, we modify a popular Gnutella client called Limewire [19]. We choose to conduct our measurement on Gnutella because: (i) it has gained a lot of popularity in recent years and is considered to be one of the top-three most popular P2P systems [20], (ii) it supports the super-peer architecture which facilitates non-intrusive passive measurements by observing traffic passing through super peers, and (iii) it is easier to modify since it is an open source protocol.

Previous studies show that Gnutella is similar to other P2P systems. For example, early studies on the fully-distributed Gnutella and the index-based Napster systems found that clients and objects in both systems exhibit very similar characteristics such as the number of files shared, session duration, availability and host uptime [21]. Another study on BitTorrent [22] made similar observations regarding object characteristics and host uptime. Also the non-Zipf behavior of object popularity in Gnutella (as we show later) has been observed before in Kazaa [1]. Therefore, we believe that the Gnutella traffic observed and analyzed in our study is representative of P2P traffic in general.

According to the Gnutella protocol specifications peers exchange several types of messages including PING, PONG, QUERY and QUERYHIT. A QUERY message contains search keywords, a TTL field and the address of the immediate neighbor which forwarded the message to the current peer. Query messages are propagated to all neighbors in the overlay for a hop distance specified by the TTL field. A typical value for TTL is seven hops. If a peer has one or more of the requested files, it replies with a QUERYHIT message. A QUERYHIT message is routed on the reverse path of the QUERY message it is responding to, and it contains the name and the URN (uniform resource name) of the file, the IP address of the responding peer, and file size. Upon receiving replies from several peers, the querying peer chooses a set of peers and establishes direct connections with them to retrieve the requested file.

Gnutella has two kinds of peers: *ultra peers*, characterized by high bandwidth and long connection periods, and *leaf peers* which are ordinary peers that only connect to ultra peers. We run our measurement node in ultra-peer mode. It passively records the contents of all QUERY and QUERYHIT messages passing through it without injecting any traffic into the network. Although we deploy only one ultra peer, we configure it to reach most of the Gnutella network as follows. We increase the number of concurrent connections that it can maintain to be up to 500. A regular ultra peer allows up to 16 connections to other ultra peers and up to 30 to leaf peers. Effectively, our peer is worth more than 20–30 regular ultra peers. In many times, our peer was connected to more than 350 other ultra peers. Let us assume that each of these 350 ultra peers connect to other 10 ultra peers on average, each of them connect to other 10, and so on. Given that queries in Gnutella are forwarded up to 7 hops among ultra peers, our peer was able to capture traffic from a huge number of peers. In addition, our peer ran continuously for eight months, while other peers joined and left the network. This means that the 200–300 other peers connected to our peer were continuously changing, which allowed our peer to reach different and larger portions of the Gnutella network.

TABLE I
SUMMARY STATISTICS OF THE MEASUREMENT STUDY

Measurement period	16 Jan. 2006 – 16 Sep. 2006
Number of QUERY messages	287,875,754
Number of QUERYHIT messages	133,952,125
Number of unique objects	16,687,320
Number of IP addresses	38,952,881
Number of unique ASes	17,369
Number of ASes with more than 100,000 downloads	127
Total traffic volume exchanged in all ASes	6,262 terabytes

The measurement study was conducted between 16 January 2006 and 16 September 2006. Our measurement peer was located at Simon Fraser University, Canada. But since the Gnutella protocol does not favor nodes based on their geographic locations [11], we were able to observe peers from thousands of ASes across the globe. During the eight months of the measurement, we recorded more than 288 million QUERY messages and 134 million QUERYHIT messages issued from approximately 38 million peers distributed over more than 17 thousand different ASes. Table I summarizes relevant measurement statistics. The large scale of our measurement study enables us to draw solid conclusions about P2P traffic. The measurement data is stored in several trace files with a total size of approximately 20 gigabytes. The trace files are available to the research community at [23].

B. Measuring and Modeling Object Popularity

In this section, we explain how we measure object popularity in different ASes. Then, we present and validate a simple, and fairly accurate, popularity model for objects in P2P systems.

The popularity of an object is defined as the probability of requesting that object relative to other objects. Object popularity is critical for the performance of the cache. Intuitively, storing the most popular objects in the cache is expected to yield higher hit rates than storing any other set of objects.

Since we are primarily interested in the performance of individual caches, we measure the popularity of objects in *each* AS. To measure the popularity of an object in a specific AS, we count the number of replicas of that object in the AS considered. The number of replicas indicates the number of downloads that were completed in the past. This means that if a cache were deployed, it would have seen a similar number of requests. This assumes that most of the downloads were supplied by peers from outside the AS, which is actually the case because peers in most current P2P networks have no sense of network proximity and thus do not favor local peers over non-local peers. In fact, previous studies [1] have shown that up to 86% of the requested P2P objects were downloaded from peers outside the local network even though they were locally available.

To count the number of replicas of a given object, we extract from our trace all QUERYHIT messages which contain the unique ID (URN) of that object. QUERYHIT messages contain the IP addresses of the responding nodes that have copies of the requested object. We can determine the number of replicas by counting the number of unique IP addresses. Then, we map

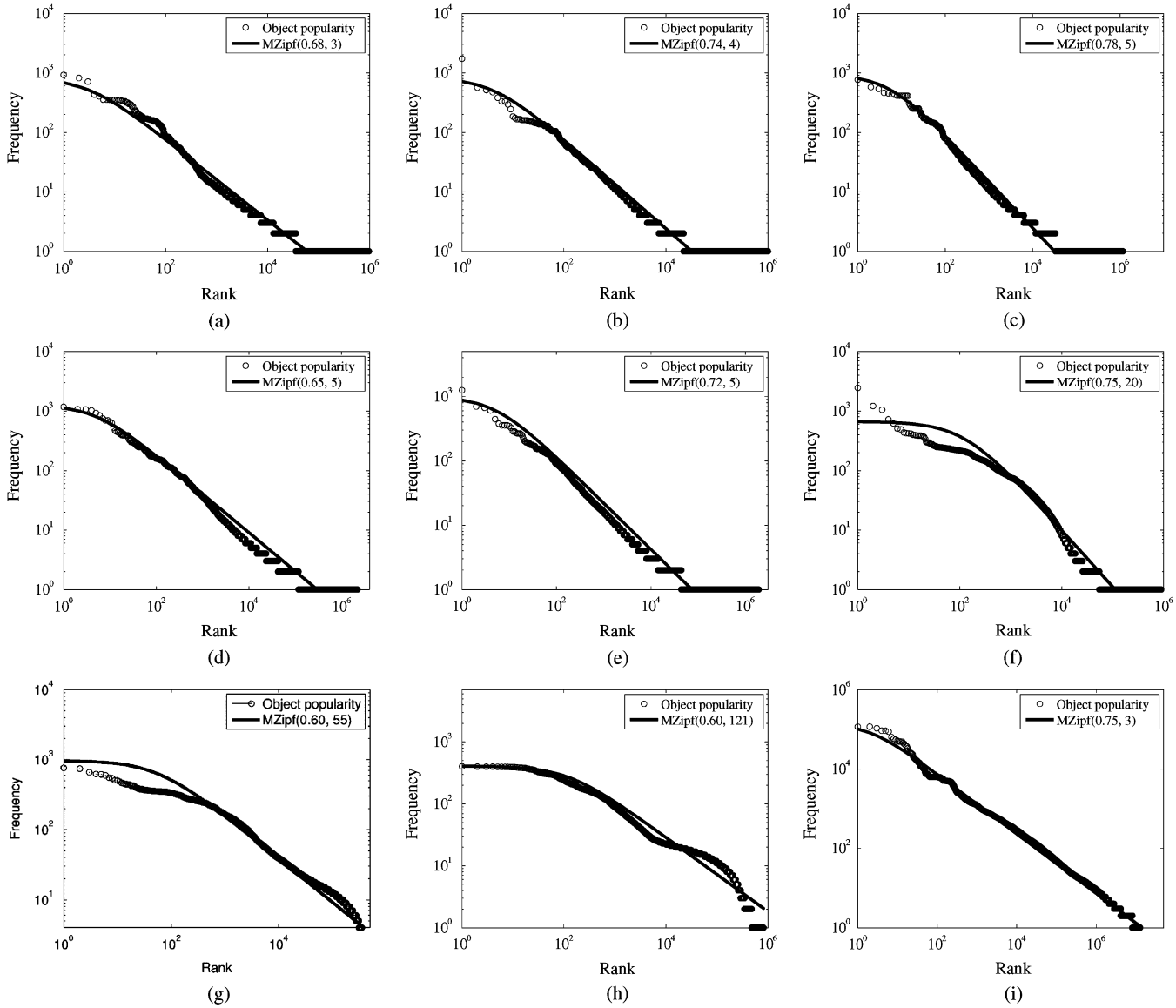


Fig. 1. Object popularity in P2P traffic: in different ASes (a)–(h), and across all ASes (i). The popularity can be modeled by Mandelbrot–Zipf Distribution. Plots are sorted from left to right based on the q values. (a) AS 223, (b) AS 1782, (c) AS 2120, (d) AS 2161, (e) AS 1859, (f) AS 9406, (g) AS 14832, (h) AS 18538, (i) All ASes.

these unique IP addresses to their corresponding ASes by using the GeoIP database [24].

We compute the popularity of each object in each of the top 18 ASes (in terms of sending and receiving messages). These top ASes contribute around 43% of the total traffic seen by our measurement node. We also compute the popularity across all ASes combined. We rank objects based on their popularity, and we plot popularity versus rank. Fig. 1 shows a sample of our results. Similar results were obtained for other ASes. As shown in the figure, there is a *flattened* head in the popularity curve of P2P objects. This flattened head indicates that objects at the lowest ranks are not as popular as Zipf-like distributions would predict. This flattened head phenomenon could be attributed to two main characteristics of objects in P2P systems: immutability and large sizes. The immutability of P2P objects eliminates the need for a user to download an object more than once. This download at most once behavior has also been observed in previous studies [1]. The large size of objects, and therefore the long time to download, may make users download only objects that they are really interested in. This is in contrast to web objects, which take much shorter times to download, and therefore, users may download web objects even if they are of marginal interest to them. These two characteristics reduce the total number of requests for popular objects.

Fig. 1 also shows that, unlike the case for web objects [9], using a Zipf-like distribution to model the popularity of P2P objects would result in a significant error. In log-log scale, the Zipf-like distribution appears as a straight line, which can reasonably fit most of the popularity distribution except the left-most part, i.e., the flattened head. A Zipf-like distribution would greatly overestimate the popularity of objects at the lowest ranks. These objects are the most important to caching mechanisms, because they are the good candidates to be stored in the cache.

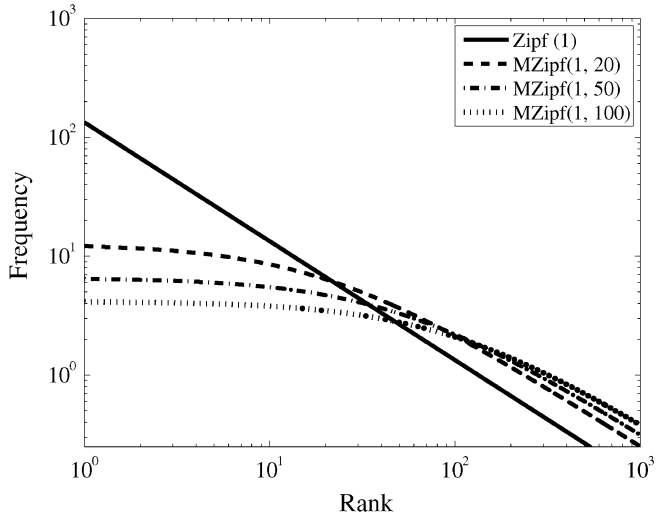


Fig. 2. Zipf versus Mandelbrot–Zipf for different q (plateau factor) values.

We propose a new model that captures the flattened head of the popularity distribution of objects in P2P systems. Our model uses the Mandelbrot–Zipf distribution [25], which is the general form of Zipf-like distributions. The Mandelbrot–Zipf distribution defines the probability of accessing an object at rank i out of N available objects as

$$p(i) = \frac{K}{(i+q)^{\alpha}} \tag{1}$$

where $K = 1/(\sum_{i=1}^{N} 1/(i+q)^{\alpha})$, α is the skewness factor, and $q \geq 0$ is a parameter which we call the *plateau* factor. q is so called because it is the reason behind the plateau shape near to the left-most part of the distribution. Notice that the higher the value of q, the more flattened the head of the distribution will be. When $q = 0$, Mandelbrot–Zipf distribution degenerates to a Zipf-like distribution with a skewness factor α. Fig. 2 compares Zipf distribution versus Mandelbrot–Zipf distribution for different q values. Notice that, there is about an order of magnitude difference in frequency between the two distributions at the lowest ranks.

To validate this popularity model, we fit the popularity distributions of objects in each of the top 18 ASes to Mandelbrot–Zipf distribution using the Matlab distribution fitting tool. Our results, some of them are shown in Fig. 1, indicate that Mandelbrot–Zipf distribution models the popularity of P2P objects reasonably well. The significance of the plateau factor q is that it controls the left-most part of the distribution; i.e., the fraction of total requests received by objects at the lowest ranks. As such, q serves as an important indication of the feasibility of caching and the achievable byte hit rate; the larger the value of q the lesser the benefit of caching especially for caching schemes that store entire objects. We elaborate more on the impact of the Mandelbrot–Zipf model on caching in the following section.

The finding that object popularity in P2P systems follows a Mandelbrot–Zipf model is important in its own right, because it gives a simple formula for modeling the P2P traffic behavior observed in our traces as well as by other researchers. e.g., [1]. This model can be used for example to: (i) analytically analyze the performance of P2P systems in general, and (ii) generate more accurate synthetic traces for P2P traffic.

C. The Effect of Mandelbrot–Zipf Popularity on Caching

In this section, we analyze the impact of the Mandelbrot–Zipf popularity model on the cache hit rate and byte hit rate using simple analysis and simulation.

We start with a simple analysis of an LFU (Least Frequently Used) policy. Under LFU, the C most popular objects are stored in the cache. For simplicity, we assume that all objects have the same size and the skewness parameter α is 1. We are mainly interested in exploring the impact of the plateau factor q on the hit rate. The hit rate H of an LFU cache is given by

$$H = \sum_{i=1}^{C} p(i) = \sum_{i=1}^{C} \frac{K}{(i+q)} \approx \int_{i=1}^{C} \frac{K}{(i+q)} di = K \ln\left(\frac{1+C/q}{1+1/q}\right). \tag{2}$$

Equation (2) implies that increasing q results in a decrease in hit rate. When $q \to \infty$, i.e., the head is very flat, the hit rate approaches zero. In contrast, for a Zipf-like popularity distribution ($q = 0$), the hit rate is $H = K \ln C$. To further illustrate the impact of Mandelbrot–Zipf on the cache performance, we plot in Fig. 3(a) the *relative loss* in hit rate between Zipf and Mandelbrot–Zipf distributions. The relative loss in hit rate is computed as $(H^{\mathrm{Zipf}} - H^{\mathrm{MZipf}})/H^{\mathrm{MZipf}}$, where H^{Zipf} and H^{MZipf} are the hit rates achieved by an LFU cache if the popularity follows Zipf and Mandelbrot–Zipf distributions, respectively. As the figure shows, significant loss in hit rate could be incurred because of the flattened-head nature of the Mandelbrot–Zipf popularity distribution. The loss in hit rate is higher for smaller relative cache sizes and larger values of q.

Next, we consider an LRU (Least Recently Used) cache. We use simulation to study the impact of the popularity model on the hit rate. We generate synthetic traces as follows. We consider 4 000 equal-sized objects and randomly generate requests for these objects according to the Zipf and Mandelbrot–Zipf distributions. We run the traces through an LRU cache with a relative cache size that varies between 0 and 100%. We compute the hit rate in each case. The results are shown in Fig. 3(b). As the figure indicates, the situation is even worse under LRU: higher drops in hit rates are observed.

Finally, we use traces from the first three months of our measurement study and compute the maximum achievable byte hit rate in two different ASes. We pick two ASes from our traces with similar α values but different q values: AS397 with $q = 8$ and AS14832 with $q = 55$ (as observed in the first three months). We use an optimal off-line algorithm which looks at the trace of each AS and stores in the cache the objects which will serve the most number of bytes, hence, achieves the highest byte hit rate. We perform trace-based simulation and compute the byte hit rate under various cache sizes for both ASes. As can be seen from Fig. 3(c), with a cache size of 400 GB, a byte hit rate of 24% is achieved under AS397, while only 9% byte hit rate is achieved under AS14832 using the same cache size. This means that the top popular objects which can fit in a cache size

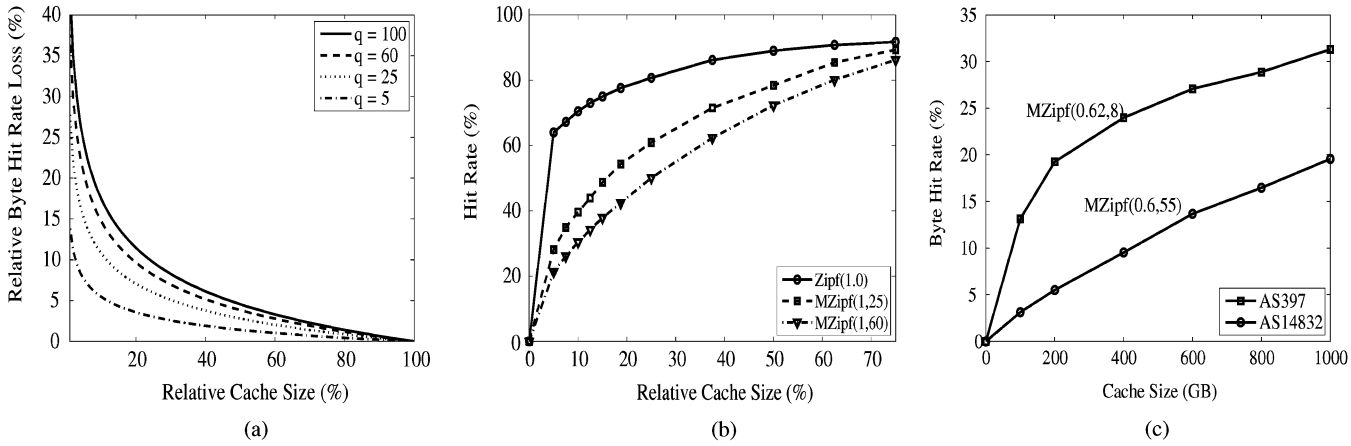


Fig. 3. Effect of Mandelbrot–Zipf popularity distribution on the cache performance. (a) Hit rate loss under LFU (analytic). (b) Hit rate under LRU (simulation). (c) Byte hit rate under an offline optimal policy (trace-based).

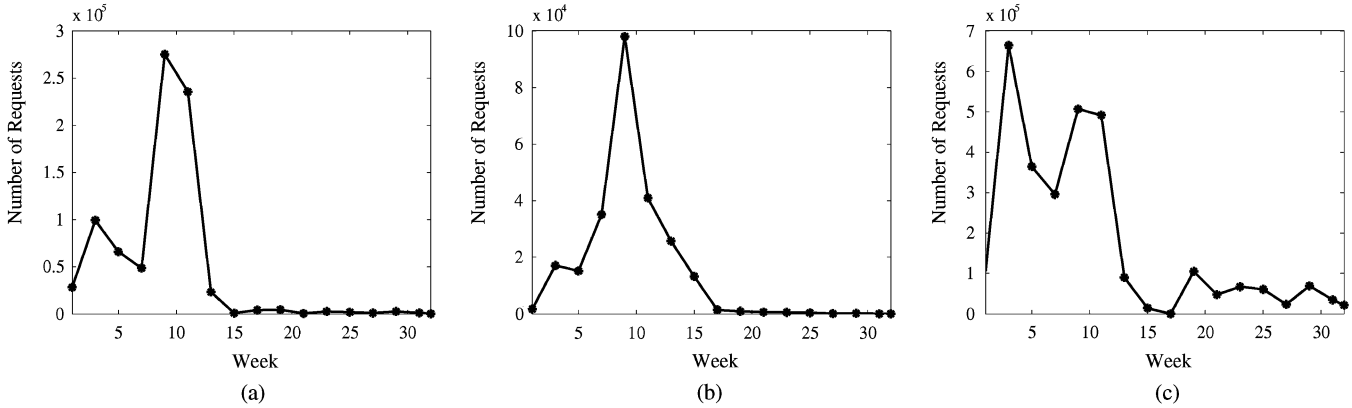


Fig. 4. Popularity dynamics in P2P systems. The figure shows popularity of the 100 most requested objects in the top two ASes (a)–(b), and in all ASes (c). (a) Top first AS. (b) Top second AS. (c) All ASes.

of 400 GB receive 24% of the total outgoing requests in AS397, in comparison to 9% of the total outgoing requests in AS14832.

These observations and experiments imply that caching schemes that capitalize on object popularity *alone* may not yield high hit rates/byte hit rates and may not be very effective in reducing the ever-growing P2P traffic.

D. Popularity Dynamics

In designing dynamic caching algorithms, it is important to understand the timescale at which the popularity of objects changes, so that we can maintain in the cache the most popular objects. This is particularly important in P2P systems since the total amount of traffic far exceeds the cache size.

To measure the turnover of popularity in P2P traffic, we perform the following experiment. We choose the top 100 most popular objects during the third month of our measurement as seen in the top first AS, top second AS and all ASes. We trace the popularity of those objects by counting the number of requests they receive per week for the entire eight months of our measurement study. Fig. 4 shows that popular objects gain popularity in a relatively short timescale reaching their peak in about 5–10 weeks. The popularity of those objects drops dramatically after that. As the figures show, we observe as much as a sixfold decrease in popularity in a matter of 5–10 weeks. This means that if we store objects in the cache based on their frequency of requests, they will be retained in the cache long after they lose their popularity. Thus, frequency-based caching policies, e.g., LFU, may not be very effective in caching P2P traffic. Also notice that the popularity of objects is not very short-lived. As shown in the figure, a popular object enjoys about 3 months of popularity before its request frequency dies out. This indicates that incremental partial caching of objects is beneficial since there is enough time for the cache to build popularity profiles, identifying which objects are truly popular and incrementally caching them until they are completely available in the cache. If the time scale were small, i.e., in the order of days, full caching of objects would be a better approach. We discuss incremental partial caching versus full caching further in Sections IV and V-E.

E. Measuring and Modeling Object Size

Many web caching algorithms use object size in their decision to evict objects. Other algorithms, such as GreedyDual-Size [6], [7], incorporate size with other characteristics to maximize cache performance. With the growing amount of P2P traffic and the large size of exchanged objects, cache size is a very important resource, and any effective policy must strive to achieve the

objective of placing in the limited-size cache the best objects that will maximize the byte hit rate.

To understand object size distributions, we collect information about objects in two popular P2P file-sharing systems: BitTorrent and Gnutella. In BitTorrent, there are a number of web servers that maintain metadata (*torrent*) files about objects in the network. These servers are known as torrent sites. We developed a script to contact four popular torrent sites and download random torrent files. We downloaded 100 thousand torrent files, 49.3% of which are unique. Each torrent file contains information about the shared object including object size, number of segments in the object and the size of each segment. For the Gnutella system, we extract from our traces the sizes of all unique objects seen during our measurement.

The histogram of the size distribution in both systems exhibits several peaks, where each peak corresponds to a different workload (figures are not shown due to space limitations, they are available in [26]). For example, a peak around 700 MB corresponds to most shared CD movies; another peak around 300 MB corresponds to high quality TV video series and some software objects, while a peak around few megabytes corresponds to video and audio clips. The location of those peaks are almost identical in both Gnutella and BitTorrent. However, BitTorrent has two more peaks: one peak around 1.5 GB corresponding to high quality avi movies, and another smaller peak at 4.5 GB corresponding to DVD movies. We have checked the validity of the peak-to-content-type mapping by randomly sampling many files from the content and checking the type of the sampled files. The similarity of workloads in Gnutella and BitTorrent leads us to believe that similar distributions exist for shared content on other file-sharing P2P systems.

The existence of multiple workloads has several consequences on the cache design. For example, audio files tend to be requested more often than large video files. Thus, using an LFU policy would be biased against large objects. On the other hand, using object size as a replacement criterion, i.e., evicting objects with the smallest size, would be biased against smaller objects. Worse yet, this might result in a scenario where we evict tens of popular mp3 objects to make space for a not-so-popular large video object that would be requested only once. Therefor, any P2P caching algorithm will have to consider the intrinsic properties of each workload and the fact that P2P object size extends from few kilobytes to few gigabytes.

Understanding object size and the multiple workloads that exist guided the design of our P2P caching algorithm. We show in the Section IV how our algorithm leverages the existence of multiple workloads to divide objects of each workload into equal-sized segments to improve cache performance.

F. Summary of the Measurement Study

We designed our measurement study to analyze the P2P traffic that would be observed by individual P2P caches deployed in different ASes. We found that popular P2P objects are not as highly popular as their web counterparts. The popularity distribution of P2P objects has a flattened head at the lowest ranks, and therefore, modeling this popularity as a Zipf-like distribution yields a significant error. We also found that a generalized form of the Zipf distribution, called Mandelbrot–Zipf, captures this flattened head nature and therefore is a better model for popularity of P2P objects. Furthermore, we found that the Mandelbrot–Zipf popularity has a negative impact on hit rates and byte hit rates of caches that use the common LRU and LFU policies. In addition, our study revealed the dynamic nature of object popularity in P2P systems, which should be considered by the caching algorithm. Finally, we found that objects in P2P systems have much larger sizes than web objects, and they can be classified into several categories based on content types.

IV. P2P Caching Algorithm

With the understanding of the P2P traffic we developed, we design and evaluate a novel P2P caching scheme based on partial caching. We start with an overview of the algorithm, then we discuss various implementation issues.

A. Overview

Our caching algorithm accounts for the Mandelbrot–Zipf popularity, large object sizes, and multiple workloads characteristics of the P2P traffic revealed by our measurement study. The flattened head of the Mandelbrot–Zipf popularity (see Figs. 1 and 2) implies that most of the requests are not directed towards a small set of highly-popular objects, unlike the case for web objects which follow the Zipf popularity model. Rather, the requests are spread out over the relatively larger number of objects that constitute the flattened head of the popularity distribution. To achieve high byte hit rate, the cache should store as many as possible of the objects at the flattened head. However, as shown by our measurements, object sizes are fairly large. This means that the cache can only store a few objects in their entirety. Storing an entire object upon a request may waste a large cache space, especially if this object is unpopular. This may reduce the chances of popular objects at the head of the distribution to get into the cache. Therefore, P2P caching should take a conservative approach towards admitting new objects into the cache to reduce the cost of storing unpopular objects. To achieve this objective, our algorithm divides objects into small segments and incrementally admits more segments of an object to the cache as the object receives more requests.

Our algorithm performs partial caching of objects and it incrementally increases the stored fraction of each object as the object becomes more popular. Partial caching is beneficial in P2P systems for an additional subtle reason: aborted downloads. According to [1], many download sessions are not completed. This means that even if there is a hit, the whole object may not be downloaded. Therefore, having more partial objects is better than having few full objects, because it increases the chances of hitting fragments of objects in the cache. Our experimental results (see Section V-E) verify this intuition.

To account for the multiple workloads nature of P2P traffic, our algorithm maintains the average object size for each workload. Denote the average object size in workload w as μ_w. Further let γ_i be the number of bytes served from object i normalized by its cached size. That is, γ_i can be considered as the average number of times each byte in this object has been served from the cache. The cache ranks objects according to their γ values, such that for objects ranked from 1 to n, we have

Proportional Partial Caching Algorithm

```
/* upon a request for object i */
1.  if object i is not in the cache
2.      add one segment of i to cache, evicting if necessary
3.  else
4.      hit = cached range ∩ requested range
5.      γ_i += hit / cached size of i
6.      missed = (requested range - hit)/segment size
7.      k = min[missed, max(1, (γ_i/γ_1)μ_w)]
8.      if cache does not have space for k segments
9.          evict k segments from the least valued object(s)
10.     add k segments of object i to the cache
11. return
```

Fig. 5. A Proportional Partial Caching Algorithm for P2P traffic.

$\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_n$. We refer to an object at rank i simply as object i. When an object is seen for the first time, only one segment of it is stored in the cache. If a request arrives for an object of which at least one segment is cached, the cache computes the number of segments to be added to this object's cached segments as $(\gamma_i/\gamma_1)\mu_w$, where μ_w is the mean of the object size in workload w which object i belongs to. Notice that this is only the number of segments the cache could store of object i. But since downloads can be aborted at any point during the session, the number of segments *actually* cached upon a request, denoted by k, is given by

$$k = \min\left[\text{missed},\ \max\left(1, \frac{\gamma_i}{\gamma_1}\mu_w\right)\right] \tag{3}$$

where $missed$ is the number of requested segments not in the cache. This means that the cache will stop storing uncached segments if the client fails or aborts the download, and that the cache stores at least one segment of an object. Notice that the segment size is fixed across workloads to facilitate cache management, while the number of stored segments is proportional to the object value (γ) and the mean object size (μ) of the workload to which the object under consideration belongs.

The pseudo-code of our P2P caching algorithm appears in Fig. 5. At a high level, the algorithm works as follows. The cache intercepts client requests and extracts the object ID and the requested range. If no segments of the requested range are in the cache, the cache stores at least one segment of the requested range. If the entire requested range is cached, the cache will serve it to the client. If the requested range is partially available in the cache, the cache serves the cached segments to the client, and decides how many of the missing segments to be cached using equation (3). In all cases, the cache updates the average object size of the workload to which the object belongs and the γ value of the requested object.

The algorithm uses a priority queue data structure to store objects according to their γ_i values. When performing eviction, segments are deleted from the least valued objects first. The algorithm needs to perform $O(\log N)$ comparisons with every hit or miss, where N is the number of objects in the cache. Since objects are large, this is a reasonable cost considering the small number of objects the cache will contain.

B. Deployment and Implementation Issues

The proposed caching algorithm is to be used by caches deployed at the gateway routers of ASes or ISP networks that choose to employ caching to reduce the burden of P2P traffic. Caches in different ASes work independent from each other. The algorithm does not require cooperation among caches. Based on our trace-based simulation results, caches with storage in the order of several hundreds gigabytes would yield a byte hit between 20–30% using our algorithm, which amounts to significant savings in WAN bandwidth given the enormous amount of P2P traffic. There are several implementation issues that need to be addressed to develop a cache for P2P traffic. We briefly discuss some of them in the following in light of our ongoing work to develop a fully-functioning prototype cache.

In order to take full advantage of a deployed cache while avoiding modifying the source code of P2P client software, the P2P cache should be transparent. This is similar to web caching, where the gateway router detects HTTP requests and forwards them to a web cache. Detecting P2P traffic, however, is a bit more involved because many P2P systems use dynamic ports and some of them even encrypt control packets. Nonetheless, there have been several works on identifying P2P traffic using techniques such as application signatures [27] and connection patterns [28]. The traffic identification method is orthogonal to the operation of the cache itself. Also to ensure cache transparency, the cache participates in the P2P protocol. This means that when a cache serves a hit, it acts as if it were a regular peer in the network sending to the requesting peer. In addition, when there is a miss and the caching algorithm decides to store additional segments, the cache requests these segments using the P2P protocol. Therefore, our algorithm does not require changing the underlying P2P protocol.

Another important issue in designing the cache is storing and serving fragments of objects. This is done as follows. Object IDs are stored in a heap based on their γ values. Each node of an object contains information on how many bytes are currently stored of that object. If the range of bytes is contiguous, only two fields are needed: start byte and end byte. Otherwise, a linked list is maintained, where each node in the list represents a contiguous range of bytes and has a pointer to the next range. Byte ranges are combined, and their associated nodes are merged once the missing bytes become available in the cache. Node merging accelerates serving byte ranges in case of a hit, as will be described shortly. To enhance contiguity of stored byte ranges, when the caching algorithm decides to store additional segments of an object, it prefers to fill the gap right after the first stored byte range.

Requests in P2P systems are typically issued for byte ranges. Upon receiving a request for an object, one of the following scenarios will occur. First, if the entire requested byte range is found in the cache, it is served to the client. This can be checked easily using the start and end bytes of the requested byte range and comparing them against stored byte ranges. Second, if the byte range is not in cache and the caching algorithm decides that it is not worth storing locally, the cache will forward the

query to the P2P network, as if it were an intermediate node in the P2P network. If the requested byte range is found at some peer(s) in the network, it will be sent directly to the original requesting peer, not to the cache. Direct transmission of bytes to the requesting peer reduces the load on the cache. Third, if part of the requested range is found in the cache, this part is served to the client. Then, the caching algorithm decides whether to store more segments of the requested object or not. In the former case, the cache constructs a query with the missed part of the byte range—with the cache itself as the source—and sends it to the P2P network. While receiving the missed part of the requested byte range, the cache serves it to the client and stores it locally. In the latter case, the cache constructs a query for the missed part with the requesting peer as the source and sends that query to the P2P network.

Finally, an interesting issue that we are currently exploring is how to utilize a single cache to serve requests from multiple different P2P protocols. That would further increase the byte hit rate and save WAN bandwidth. To support cross-systems caching, several aspects need to be handled, such as different IDs for the same object in different systems and different object segmentation strategies.

V. Evaluation

In this section, we use trace-driven simulation to study the performance of our P2P caching algorithm under several scenarios, and compare it against several common web caching algorithms and a recent caching algorithm proposed for P2P systems.

A. Experimental Setup

Traces and Performance Metrics. We use traces obtained from the first three months of our measurement study to conduct our experiments. Based on our discussion on measuring object popularity, we count the number of replicas of each object from QUERYHIT messages returned by peers in a particular AS. We assume that these replicas were downloaded sometime in the past, and a cache would have seen a sequence of requests for these objects if it had been deployed in that AS. Thus, we construct the sequence of requests from the unique QUERYHIT messages, i.e., the sequence has one request for each replica downloaded by a peer. Peers that replied earlier with QUERYHITs for an object are assumed to have downloaded the object earlier. Notice that, from the cache perspective, the exact time when the object was downloaded is not important. It is the relative popularity of objects and the distance between similar requests in the trace that matter. These two issues are captured by our sequences. In addition, in some of our simulations (Section V-D), we explicitly study the effect of various degrees of temporal locality using synthetic traces. We do this by controlling the temporal correlations between objects using the LRU stack model in [29]. Thus we complement our collected traces with synthetic traces to evaluate all aspects of caching.

Our objective is to study the effectiveness of deploying caches in several ASes. Thus, we measure the byte hit rate that could be achieved if a cache were to be deployed in that AS. We use the byte hit rate as the performance metric because we are mainly interested in reducing the WAN traffic. In all experiments, we use the ASes which have the most amount of traffic seen by our measurement node. This ensures that we have enough traffic from an AS to evaluate the effectiveness of deploying a cache for it. In addition, we study the impact of the plateau factor q and the skewness parameter α of the Mandelbrot–Zipf distribution on the performance of our P2P caching algorithm.

Algorithms Implemented. We run several AS traces through the cache and compare the byte hit rate achieved using several caching algorithms. We implemented six algorithms in total. In addition to our algorithm (P2P), we implemented the Least Recently Used (LRU), Least Frequently Used (LFU), popularity-aware GreedyDual-Size (GDSP) [6] and Least Sent Bytes (LSB) [8] algorithms. We also implemented the off-line optimal (OPT) algorithm to use it as a benchmark for comparison. LRU capitalizes on the temporal correlation in requests, and thus replaces the oldest object in the cache. LFU sorts objects based on their access frequency and evicts the object with the least frequency first. We compared against LRU and LFU because they are simple and widely used for web caching. Thus our comparison will answer the natural question of what if we were to use the currently deployed web caches for P2P traffic. GDSP, an enhancement on the GreedyDual-Size algorithm [7], is an elaborate algorithm designed for web caching, which accounts for several aspects including: size, recency, and popularity [6], [7]. We used object size as the cost function in GDSP to maximize byte hit rate as indicated by [6], [7]. LSB is designed for P2P traffic caching and it uses the number of transmitted bytes of an object as a sorting key. The object which has transmitted the least amount of bytes will be evicted next. LSB is the only other algorithm designed for P2P caching that we are aware of. OPT looks at the *entire* stream of requests off-line and caches the objects that will serve the most number of bytes from the cache.

Evaluation Scenarios. We evaluate the algorithms under several scenarios. First, we consider the case where objects requested by peers are downloaded entirely, that is, there are no aborted transactions. Then, we consider the case where the downloading peers prematurely terminate the downloads during the session, which is not uncommon in P2P systems [1].

We also analyze the effect of temporal locality on caching algorithms. Recall that temporal locality is defined as the likelihood of requesting an object in the near future [6], [30]. Temporal locality has two components: popularity and temporal correlations [30]. Popularity of an object is the number of requests it receives relative to other objects. Temporal correlation means that requests to the same object tend to be clustered or correlated together in time. Temporal correlation is usually measured in the number of intervening requests between two consecutive requests to the same object. A shorter sequence of intervening requests implies higher temporal correlations. We study the effect of each component of temporal locality in isolation from the other. We also study the combined effect of the two components. To isolate the popularity component, we randomly shuffle our traces to eliminate the effect of temporal correlations. For the temporal correlations component, we generate synthetic traces with various degrees of temporal correlations (and fixed popularity), and run the synthetic traces through a cache running our algorithm. For the combined effect of the two components, we

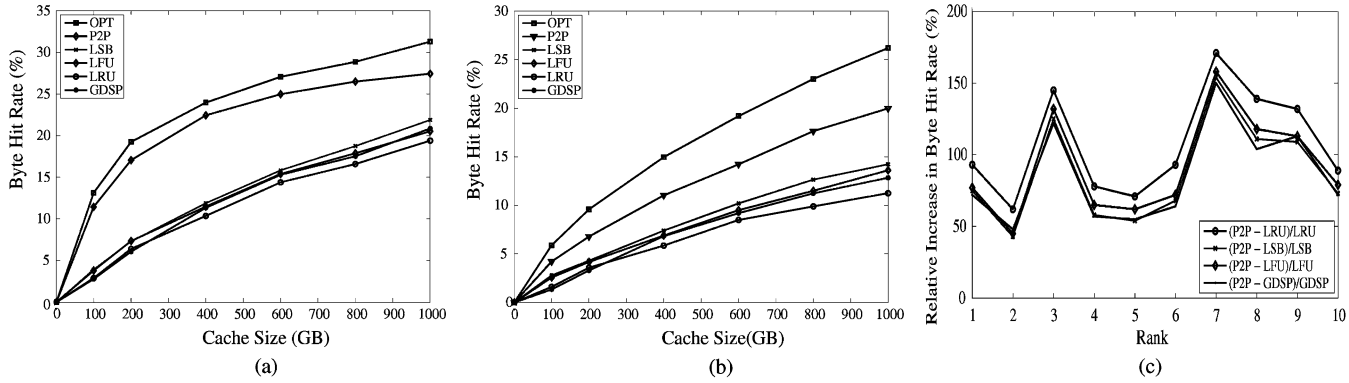


Fig. 6. Byte hit rate for different caching algorithms. No aborted downloads. (a) AS397, 48% of traffic is cacheable. (b) AS95, 54% of traffic is cacheable. (c) Top ten ASes.

use the original traces with preserved popularity and temporal correlations.

Finally, we vary the cache size between 0 and 1000 GB.

B. Caching Without Aborted Downloads

Fig. 6 shows the byte hit rate for two representative ASes with different characteristics. These two ASes have different maximum achievable byte hit rates, which is defined as the fraction of traffic downloaded more than once, i.e., cacheable traffic, over the total amount of traffic. As shown in the figure, our policy outperforms other policies by as much as 200%. For instance, in AS397 (Fig. 6(a)) with a cache of 600 GB, our policy achieves a byte hit rate of 24%, which is almost double the rate achieved by LRU, LFU, GDSP, and LSB policies. Moreover, the byte hit rate achieved by our algorithm is about 3% less than that of the offline optimal algorithm. Our traces indicate that the amount of traffic seen in AS397 is around 24.9 terabytes. This means that a reasonable cache of size 600 GB would have served about 6 terabytes locally using our algorithm; a significant saving in the WAN bandwidth.

We believe that traditional policies perform poorly for P2P traffic due to the effect of unpopular objects. For example, one-timer objects are stored entirely under traditional policies on a miss. Under our policy, however, only one segment of each one-timer will find its way into the cache, thus minimizing their effect. The same could be said about second timers, third timers and so on. Thus, our algorithm strives to discover the best objects to store in the cache by incrementally admitting them. Similar results were obtained for the other top ten ASes. Our policy consistently preforms better than traditional policies. Fig. 6(c) summarizes the relative improvement in byte hit rate that our policy achieves over LRU, LFU, GDSP, and LSB for the top ten ASes, with a cache of size 500 GB. The relative improvement is computed as the difference between the byte hit rate achieved by our policy and the byte hit rate achieved by another policy normalized by the byte hit rate of the other policy. For instance the improvement over LRU would be (P2P-LRU)/LRU. The figure shows a relative improvement of at least 40% and up to 180% can be achieved by using our algorithm. That is a significant gain given the large volume of the P2P traffic. We notice that the relative gain our policy achieves is larger in ASes with a substantial fraction of one-timers.

We also observe that the achievable byte hit rate is between 15% and 40% with reasonable cache sizes. This is similar to the achievable byte hit rate for web caching, which is practically in the range 20%–35% (CISCO technical paper [31]), or as other sources indicate 30%–60% [32]. But due to the large size of P2P objects, a small byte hit rate amounts to savings of terabytes of data.

As a final comment on Figs. 6(a) and (b), consider the byte hit rates achieved under our algorithm and the optimal algorithm. Notice that although the percentage of cacheable traffic in AS397 is less than that of AS95, the byte hit rate is higher in AS397. This is because popular objects in AS95 do not get as many requests as their counterparts in AS397. That is, the popularity distribution of AS95 has a more flattened head than that of AS397. We computed the skewness factor α and the plateau factor q of the Mandelbrot–Zipf distribution that best fits the popularity distributions of these two ASes. We found that AS95 has $\alpha = 0.6$ and $q = 50$, while AS397 has $\alpha = 0.62$ and $q = 8$. Smaller q values mean less flattened heads, and yield higher byte hit rates. Section V-F elaborates more on the impact of α and q on the byte hit rate.

C. Caching With Aborted Downloads

Due to the nature of P2P systems, peers could fail during a download, or abort a download. We run experiments to study the effect of aborted downloads on caching, and how robust our caching algorithm is. Following observations from [1], we allow 66% of downloads to be aborted anywhere in the session. To achieve this, we use the same traces used for the previous experiments with the same number of objects plus two aborted download sessions for each object in the trace. While our policy is designed to deal with aborted downloads, web replacement policies usually download the entire object upon a miss, and at times perform pre-fetching of objects. This is reasonable in the web since web objects are usually small, which means they take less cache space. But in P2P systems, objects are larger, and partial downloads constitute a large number of the total number of downloads. Fig. 7 compares the byte hit rate with aborted downloads using several algorithms. Compared to the scenario of caching under full downloads (Section V-B), the performance of our algorithm improves slightly while the performance of other algorithms declines. The improvement in our policy could

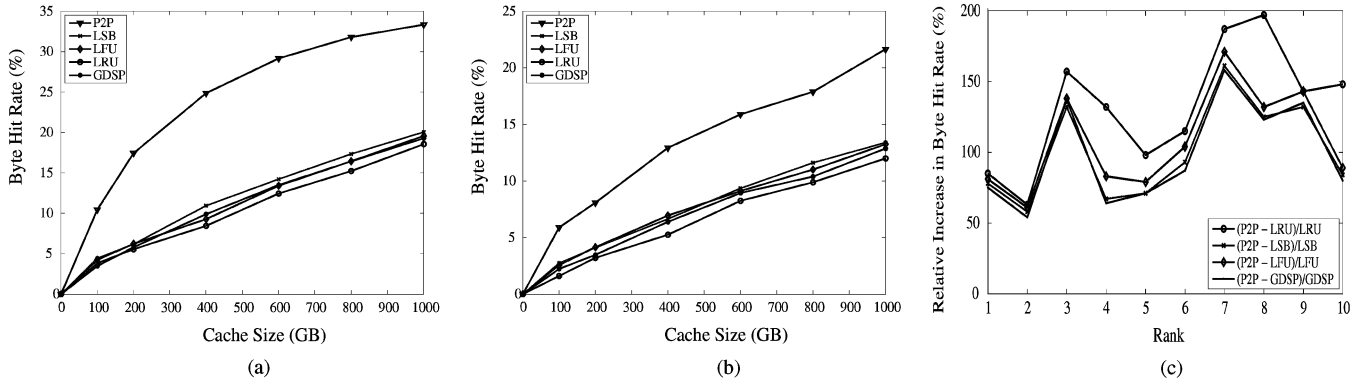


Fig. 7. Byte hit rate for different caching algorithms using traces with aborted downloads. (a) AS397. (b) AS95. (c) Relative byte hit rate improvement.

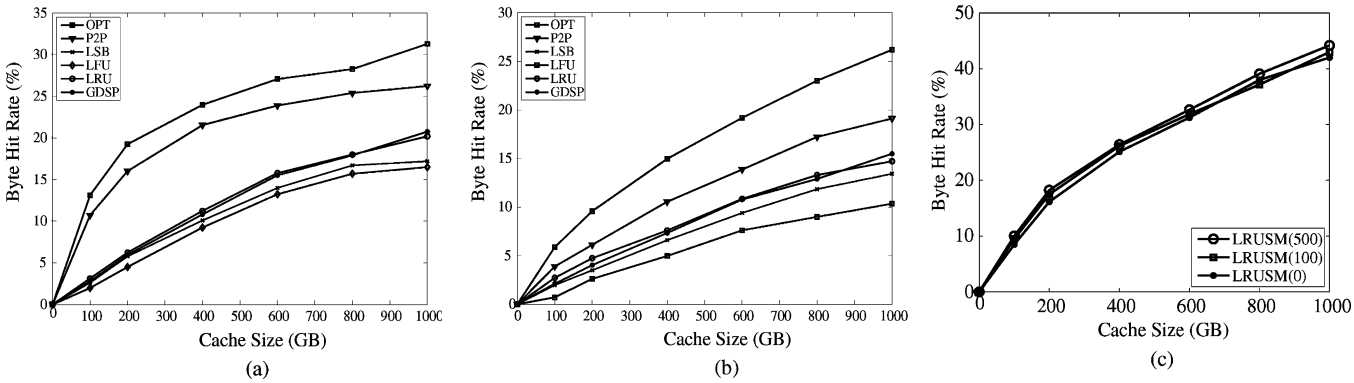


Fig. 8. Byte hit rate for different caching algorithms using traces with temporal correlations. (a) AS397, (b) AS95, (c) Using model from [29].

be explained by the fact that fewer bytes are missed in case of a failure.

The performance of LRU, LFU, GDSP, and LSB declines because they store an object upon a miss regardless of how much of it the client actually downloads. Hence, under aborted download scenarios, the byte hit rates for traditional policies suffer even more than under full download scenarios. Similar results were obtained for the top ten ASes with a cache of size 500 GB. Our policy consistently outperforms LRU, LFU, LSB and GDSP with a wide margin in all ten ASes. Fig. 7(c) shows that the relative improvement in byte hit rate is at least 50% and up to 200%.

D. Sensitivity of the P2P Caching Algorithm to Temporal Locality

As we discuss in Section V-A, temporal locality is caused by popularity of objects and temporal correlations of requests. In the previous experiments we isolated the effect of popularity from the effect of temporal correlations by shuffling the traces. In this section, we first study the combined effect of the two components of temporal locality. Then we isolate temporal correlation and study its effect on caching while fixing popularity.

For the combined effect of popularity and temporal correlations, we use the original unshuffled traces, with preserved popularity and temporal correlations. Figs. 8(a) and (b) show the byte hit rate in two ASes: AS95 and AS397, respectively. LRU and GDSP perform slightly better than LSB and LFU because they make use of temporal correlation between requests. However, the achieved byte hit rate under the four algorithms is still low compared to the byte hit rate achieved by our algorithm. Note that the byte hit rate under our algorithm is slightly smaller than in the previous experiments, where we only used popularity. This is because our algorithm does not capitalize on temporal correlation. However, this reduction is small, less than 3%. The fact that the performance of our algorithm does not suffer much under temporal correlation and still outperforms other algorithms (e.g., LRU and GDSP) could be explained as follows. We believe that object size is the dominant factor in caching for P2P systems, because the maximum cache size we used (1000 GB) is still small compared to the total size of objects, less than 5%–10% in most cases. As a consequence, object admission strategy is a key element in determining the byte hit rate, which our algorithm capitalizes on. We obtained similar results for other ASes.

Now, we fix popularity and study the effect of temporal correlations. Since we cannot control the degree of correlation in our traces, we generate synthetic traces with various degrees of temporal correlations. This is done by using the LRU Stack Model [29], which generates correlated requests by using a stack of depth n to keep an ordered list of the last requested n objects such that the subset of objects in the stack have a higher probability of being accessed again than they would if they were not in the stack. The stack depth reflects the degree of temporal correlations in the generated traces: higher depths indicate stronger temporal correlations.

The authors of [29] provide an open-source trace generation tool called ProWGen. We modify ProWGen to use Mandelbrot–Zipf distribution and provide object sizes from our traces. We fix the popularity by using $q = 20$ and $\alpha = 0.6$. To study

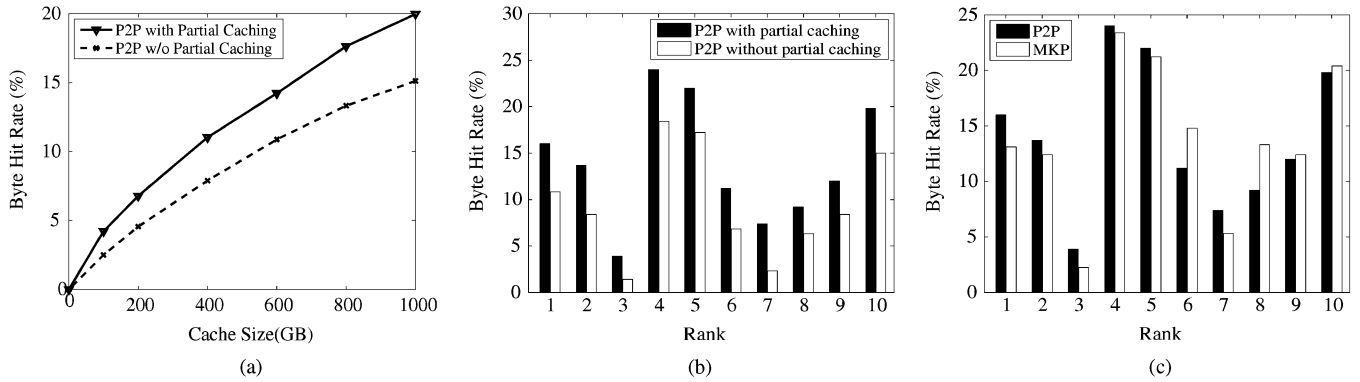


Fig. 9. The importance of partial caching in P2P traffic. The figure shows the byte hit rate achieved using our P2P algorithm with and without partial caching in AS95 (a), and in top ten ASes (b). (c) shows the byte hit rates achieved by our *online heuristic* partial caching algorithm (P2P) and by the MKP *offline optimal* in top ten ASes. (a) AS95. (b) P2P algorithm. Cache size 500 GB. (c) Top ten ASes. Cache Size 500 GB.

the sensitivity of our algorithm to temporal correlations, we vary depth of the LRU stack between 0 and 500. Fig. 8(c) shows that our P2P caching algorithm is not sensitive to the degree of temporal correlations, and the achieved byte hit rate stays fairly constant across wide range of temporal correlations. This shows the robustness of our algorithm in face of different traffic patterns with various degrees of temporal correlations.

E. Partial Caching Versus Full Caching

As we show in previous experiments, the performance of traditional policies suffers when they are used to cache P2P traffic. This is mainly because they cache entire objects, thus running the risk of storing in the cache large unpopular objects. Our algorithm takes a conservative approach by incrementally caching objects in proportion to their request frequency. To study how much partial caching contributes to the performance of our P2P caching algorithm, we perform the following experiment. We run the traces of the top 10 ASes through a 500 GB cache that runs a modified version of our algorithm without the partial caching capability. That is, we use a utility function based on the number of bytes served from an object normalized by its cached size, and evict the object with the least utility value. Fig. 9(a) shows the results for a sample AS where the performance of our algorithm clearly suffers when partial caching is not used. In Fig. 9(b), we show the byte hit rate achieved using our algorithm with and without partial caching capability for the top ten ASes using a cache of size 500 GB. As the figure shows, in some ASes, partial caching contributes as much as 60% of the byte hit rate achieved by our algorithm.

To further investigate the importance of partial caching for P2P traffic, we compare our algorithm versus an *offline* algorithm that looks at the *entire* trace and fills the cache with the most popular objects. We call this algorithm the Most K-Popular (MKP) algorithm. MKP is optimal in terms of object popularity, i.e., it achieves the optimal hit rate, not the optimal byte hit rate. In Fig. 9(c), we plot the byte hit rate of our *online heuristic* algorithm versus that of the *offline optimal* MKP algorithm in the top ten ASes. The figure shows that the P2P caching algorithm outperforms MKP in six of the top ten ASes. In two of the remaining four ASes, at ranks 9 and 10, MKP outperforms our algorithm with a very small margin $(<1\%)$, and by a small margin $(<4\%)$ in the other two ASes at ranks 6 and 8. From further inspection, we found out that these four ASes are characterized by a small plateau factor q (less flattened head) in their popularity curve, and that their popular objects tend to have large sizes. The results in Fig. 9 imply that partial caching contributes significantly toward achieving high byte hit rates in caching of P2P traffic.

In addition, as shown in Fig. 6 the byte hit rate achieved by P2P caching algorithm (which does partial caching) is close to the optimal byte hit rate in most cases. Therefore, based on the results in Figs. 6 and 9, we believe that partial caching is crucial in P2P traffic. It remains an open question, though, to prove whether partial caching is absolutely *necessary* to achieve high byte hit rates in P2P traffic.

F. Effect of α and q on P2P Caching

As we mention in Section III, P2P traffic can be modeled by a Mandelbrot–Zipf distribution with two parameters: a skewness parameter α and a plateau factor q. In this section, we study the effect of α and q on the byte hit rate of our algorithm via simulation. We did not use our traces because they may not capture the performance of our algorithm for all possible values of α and q. We randomly pick 100 000 objects from our traces and generate their frequencies using Mandelbrot–Zipf with various values for α and q. We fix the cache size at 1 000 GB and we assume a no-failure model where peers download objects entirely.

To study the effect of different α values, we fix q and change α between 0.4 and 1. As shown in Fig. 10, the byte hit rate increases as the skewness factor α increases. This is intuitive since higher values of α mean that objects at the lowest ranks are more popular and caching them yields higher byte hit rate. Thus ASes whose popularity distribution is characterized with high α values would benefit more from caching than those with low α values.

Another parameter that determines the achievable byte hit rate is the plateau factor q of the popularity distribution. q reflects the flattened head we observed in Section III. Fig. 11 shows that the byte hit rate decreases as q increases. The reason for this is that a higher value of q means popular objects are requested less often. This has a negative impact on a cache that

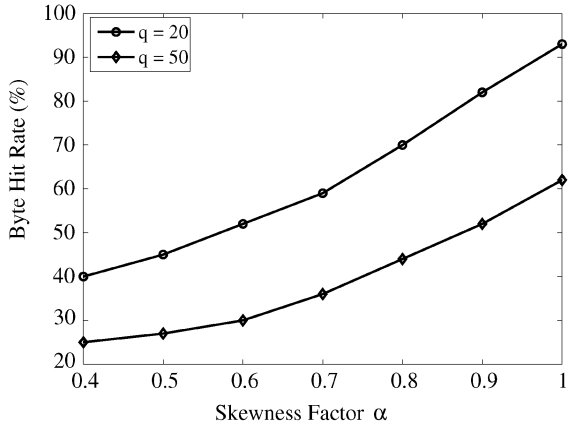


Fig. 10. The impact of α on caching.

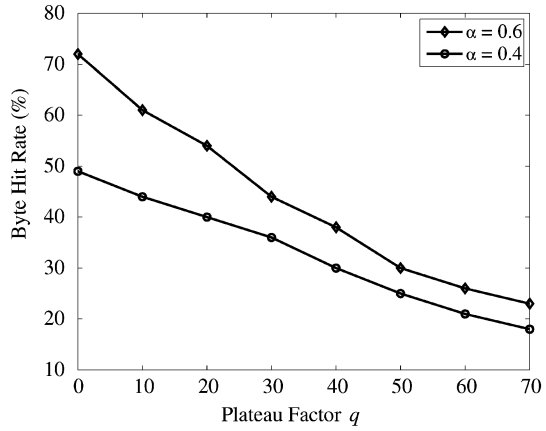


Fig. 11. The impact of q on caching.

stores those popular objects, because they receive less downloads resulting in a decreased byte hit rate. Notice, though, that we are only exploring how changing the Mandelbrot–Zipf parameters impacts the caching performance and not suggesting using specific values for q and α. This means that there exists some ASes for which Mandelbrot–Zipf accurately captures the popularity model by a small value of α and a large value of q resulting in popularity being spread out among objects more evenly, i.e., approaching a uniform distribution. But in such model all caching algorithms will suffer since the portion of requests received by popular objects decreases drastically.

VI. Conclusion

In this paper, we conducted an eight-month measurement study on the Gnutella P2P system. Using real-world traces, we studied and modeled characteristics of P2P traffic that are relevant to caching, such as popularity, object size and popularity dynamics. We found that the popularity distribution of P2P objects cannot be captured accurately using Zipf-like distributions. We proposed a new Mandelbrot–Zipf model for P2P popularity and showed that it closely models object popularity in several AS domains. The finding that object popularity in P2P systems follows a Mandelbrot–Zipf model could be of interest in its own right, since it gives a simple formula for modeling the P2P traffic behavior observed in our traces as well as by other researchers, e.g., [1]. This model can be used for example to: (i) analytically analyze the performance of P2P systems in general, and (ii) generate more accurate synthetic traces for P2P traffic. Our measurement study also revealed that the P2P traffic is a mix of multiple workloads, and that objects in P2P systems have much larger sizes than web objects.

We used simple analytic analysis and trace-based simulations to study the impact of the P2P traffic characteristics on caching. We found that: (i) the Mandelbrot–Zipf popularity has a negative effect on byte hit rates of caching schemes that store entire objects such as LRU, LFU, and popularity-aware Greedy-Dual-Size (GDSP) [6], and (ii) object admission strategies in P2P caching are critical to the performance of P2P caches.

Guided by our findings from the measurement study, we designed and evaluated a new P2P proportional partial caching algorithm that is based on segmentation and incremental admission of objects according to their popularity. Using trace-driven simulations, we showed that our algorithm outperforms traditional algorithms by a significant margin and achieves a byte hit rate that is up to triple the byte hit rate achieved by other algorithms. Our results indicate that partial caching is crucial to the performance of P2P caching algorithms in P2P systems. We showed that our algorithm is robust against various workloads with different degrees of temporal correlations and against aborted downloads which are common in P2P systems.

References

[1] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. 19th ACM Symp. Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, Oct. 2003, pp. 314–329.

[2] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos, "Is P2P dying or just hiding?," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM'04)*, Dallas, TX, Nov. 2004, pp. 1532–1538.

[3] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should internet service providers fear peer-assisted content distribution?," in *Proc. 5th ACM SIGCOMM Conf. Internet Measurement (IMC'05)*, Berkeley, CA, Oct. 2005, pp. 63–76.

[4] S. Podlipnig and L. Bszrmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 347–398, Dec. 2003.

[5] J. Liu and J. Xu, "Proxy caching for media streaming over the Internet," *IEEE Commun. Mag.*, vol. 42, no. 8, pp. 88–94, Aug. 2004.

[6] S. Jin and A. Bestavros, "Popularity-aware GreedyDual-size web proxy caching algorithms," in *Proc. IEEE Int. Conf. Distributed Computing Systems (ICDCS'00)*, Taiwan, May 2000.

[7] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. USENIX Symp. Internet Technologies and Systems*, Monterey, CA, Dec. 1997, pp. 193–206.

[8] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak, "Cache replacement policies revisited: The case of P2P traffic," in *Proc. 4th Int. Workshop on Global and Peer-to-Peer Computing (GP2P'04)*, Chicago, IL, Apr. 2004, pp. 182–189.

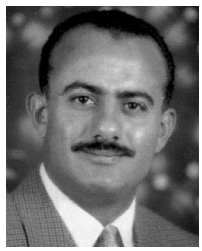
[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM'99*, New York, NY, Mar. 1999, pp. 126–134.

[10] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. Networking*, vol. 12, no. 2, pp. 219–232, Apr. 2004.

[11] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst, "Characterizing the query behavior in peer-to-peer file sharing systems," in *Proc. ACM/SIGCOMM Internet Measurement Conf. (IMC'04)*, Taormina, Sicily, Italy, Oct. 2004, pp. 55–67.

[12] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit, "Are file swapping networks cacheable? Characterizing P2P traffic," in *Proc. 7th Int. Workshop on Web Content Caching and Distribution (WCW'02)*, Boulder, CO, Aug. 2002.

[13] Home Page of CacheLogic. [Online]. Available: http://www.cachelogic.com/
[14] Home Page of PeerCache. [Online]. Available: http://www.joltid.com/
[15] Home Page of Sandvine. [Online]. Available: http://www.sandvine.com/
[16] S. Jin, A. Bestavros, and A. Iyengar, "Network-aware partial caching for internet streaming media," *ACM Multimedia Syst. J.*, vol. 9, no. 4, pp. 386–396, Oct. 2003.
[17] S. Park, E. Lim, and K. Chung, "Popularity-based partial caching for VOD systems using a proxy server," in *Proc. 15th Int. Parallel and Distributed Processing Symp. (IPDPS'01)*, San Francisco, CA, Apr. 2001.
[18] Gnutella Home Page. [Online]. Available: http://www.gnutella.com
[19] Limewire Home Page. [Online]. Available: http://www.limewire.com/
[20] S. Zhao, D. Stutzbach, and R. Rejaie, "Characterizing files in the modern Gnutella network: A measurement study," in *Proc. SPIE/ACM Multimedia Computing and Networking (MMCN'06)*, San Jose, CA, Jan. 2006.
[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," *ACM/Springer Multimedia Syst. J.*, vol. 9, no. 2, pp. 170–184, Aug. 2003.
[22] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P file-sharing system: Measurements and analysis," in *Proc. 4th Int. Workshop on Peer-To-Peer Systems (IPTPS'05)*, Ithaca, NY, Feb. 2005, pp. 205–216.
[23] Network Systems Lab Home Page. [Online]. Available: http://nsl.cs.sfu.ca
[24] GeoIP Database Home Page. [Online]. Available: http://www.maxmind.com
[25] Z. Silagadze, "Citations and the Zipf-Mandelbrot's law," *Complex Syst.*, vol. 11, no. 487–499, 1997.
[26] O. Saleh, "Modeling and caching of peer-to-peer traffic," Master's Thesis, Sch. Comput. Sci., Simon Fraser Univ., Surrey, BC, Canada, Dec. 2006.
[27] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of P2P traffic using application signatures," in *Proc. 13th Int. World Wide Web Conf. (WWW'04)*, New York City, NY, May 2004, pp. 512–521.
[28] T. Karagiannis, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proc. ACM/SIGCOMM Internet Measurement Conf. (IMC'04)*, Taormina, Sicily, Italy, Oct. 2004, pp. 121–134.
[29] M. Busari and C. Williamson, "ProWGen: A synthetic workload generation tool for simulation evaluation of web proxy caches," *J. Comput. Netw.*, vol. 38, no. 6, pp. 779–794, Apr. 2002.
[30] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in web traces and implications for designing replacement policies," *J. Comput. Netw.*, vol. 45, no. 4, pp. 379–398, Jul. 2004.
[31] G. Huston, "Web Caching," *The Internet Protocol J.*, vol. 2, no. 3, Sep. 1999.
[32] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Reading, MA: Addison-Wesley, 2002.



Mohamed Hefeeda (S'01–M'04) received the B.Sc. and M.Sc. degrees from Mansoura University, Egypt, in 1994 and 1997, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2004.

He is an Assistant Professor in the School of Computing Science, Simon Fraser University, Surrey, BC, Canada, where he leads the Network Systems Lab. His research is funded by Canadian funding agencies and industry through several grants. His research interests include multimedia networking, peer-to-peer systems, and wireless sensor networks.

Dr. Hefeeda is a member of the ACM Special Interest Groups on Data Communications (SIGCOMM) and Multimedia (SIGMM).

Osama Saleh received the Masters degree in computing science from Simon Fraser University, Surrey, BC, Canada, in December 2006.

He is currently a Software Engineer at Eyeball Networks Inc., Vancouver, BC, Canada. His research interests include peer-to-peer systems, Internet traffic modeling, Internet caching algorithms and VoIP.