NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-04-37
March 9th, 2004

Looking at the Server Side of Peer-to-Peer Systems

Yi Qiao, Dong Lu, Fabián E. Bustamante and Peter A. Dinda

Abstract

Peer-to-peer systems have grown significantly in popularity over the last few years. An increasing number of research projects have been closely following this trend, looking at many of the paradigm's technical aspects. In the context of data-sharing services, efforts have focused on a variety of issues from object location and routing to fair sharing and peer lifespans. Overall, the majority of these projects have concentrated on either the whole P2P infrastructure or the *client*-side of peers. Little attention has been given to the peer's *server*-side, even when that side determines much of the everyday user's experience. In this paper, we make the case for looking at the server side of peers, focusing on the problem of scheduling with the intent of minimizing the average response time experienced by users. We start by characterizing server workload based on extensive trace collection and analysis. We then evaluate the performance and fairness of different scheduling policies through trace-driven simulations. Our results show that average response time can be dramatically reduced by more effectively scheduling the requests on the server-side of P2P systems.

Keywords: Peer-to-peer, server-side, scheduling, SRPT, uploading/downloading.

Looking at the Server Side of Peer-to-Peer Systems

Yi Qiao, Dong Lu, Fabián E. Bustamante and Peter A. Dinda
Department of Computer Science
Northwestern University, Evanston, IL 60201, USA.
emails: {yqiao,donglu,fabianb,pdinda}@cs.northwestern.edu

March 9th, 2004

Abstract

Peer-to-peer systems have grown significantly in popularity over the last few years. An increasing number of research projects have been closely following this trend, looking at many of the paradigm's technical aspects. In the context of data-sharing services, efforts have focused on a variety of issues from object location and routing to fair sharing and peer lifespans. Overall, the majority of these projects have concentrated on either the whole P2P infrastructure or the *client*-side of peers. Little attention has been given to the peer's *server*-side, even when that side determines much of the everyday-user's experience. In this paper, we make the case for looking at the server-side of peers, focusing on the problem of scheduling with the intent of minimizing the average response time experienced by users. We start by characterizing server workload based on extensive trace collection and analysis. We then evaluate the performance and fairness of different scheduling policies through trace-driven simulations. Our results show that average response time can be dramatically reduced by more effectively scheduling the requests on the server-side of P2P systems.

1 Introduction

The popularity and tremendous success of peer-to-peer systems have motivated considerable research on many of the paradigm's technical aspects. In the context of data-sharing services, a number of projects have explored a wide variety of issues including more scalable object location and routing protocols, fair resource sharing, and high churn-resilient systems, just to name a few. The majority of these projects have, so far, concentrated on either the whole P2P infrastructure or the *client* side of a peer. Little attention has been given to the peer's *server*-side, although that side determines much of the everyday user's experience.

After determining alternative sources for a desired object, a peer must initiate the object download from a subset of possible providers; each party effectively adopting *client* and *server* roles. Looking at the client-side of peers, we are only aware of Bernstein et al. [7] where the authors propose using machine learning for the construction of server peer selection strategies for faster download speed. The server-side, on the other hand, remains mostly ignored.

However, recent studies suggest that servers in a P2P data-sharing system often turn out to be a performance bottleneck. From the analysis of P2P traffic collected at border routers at the University of Washington, Saroiu et al. [15] report that a small number of Kazaa [3] servers are responsible for serving the majority of requests for content. Their traces indicate that over 80% of all download requests are rejected because of the saturation of server capacity. Similarly, another study of P2P workload by the same group [11] shows that object downloading in Kazaa can be extremely slow, with 50% of all requests for large objects ($>$100MB) taking more than one day and nearly 20% taking over one week to complete!

These results clearly argue for taking a closer look at the server-side of peers, and this paper reports on our initial steps. We focus here on the scheduling problem, and our goal is to design efficient and fair scheduling algorithms for P2P servers that result in a lower average response time (a.k.a. sojourn time) for client peers. Despite the similarity in purpose with research on scheduling algorithms for web servers [5, 9, 6, 12], a closer look at the characteristics of P2P request traces indicate that many findings from the web context are not directly applicable to our problem:

- The *fetch-at-most-once* behavior of P2P client makes the distribution of object popularity decidedly *not* conform to Zipf or other power-laws [11].

- Requests to P2P servers are often not for the whole object, but instead for only a small chunk (with the remaining parts downloaded from other servers). In fact, as our traces show, the amount of data actually served is often just a fraction of the requested size.

- While web servers can reasonably assume full control over resources, P2P servers are commonly configured with quite conservative upper bounds for resource consumption to control their impact on their users' other tasks.

- Although web servers often experience high load [1], close to 1, they are typically not overloaded. Popular P2P servers, on the other hand, normally oper-

[1]In this paper, the load is defined as mean job arrival rate over mean service rate, as is the standard definition for load in queuing theory [17].

ate overloaded [15] due in part to low resource availability and, on average, large object sizes.

We start by characterizing server workload through trace collection and analysis. Our traces of download requests were collected from a set of P2P servers behind 100Mbps and cable modem connections. To the best of our knowledge, ours is the first attempt at characterizing server workload on P2P systems.

We study the performance and fairness of different scheduling policies using our workload characterization and trace-driven simulations. Our results show that average response time can be dramatically reduced by scheduling jobs on the server-side of P2P systems using policies based on preemtive *Shortest-Remaining-Processing-Time (SRPT)*.

We describe our trace collection methodology in Section 2 and characterize different aspects of server workload in Section 3. Section 4 presents our trace-based evaluation of various scheduling policies for P2P servers. We summarize our results, conclude and indicate directions for future work in Section 5.

2 Trace Collection

In order to collect a large number of client requests we built a *honey-pot*, a peer offering a large set of popular files selected based on query-related traffic observed through passive monitoring. To create our shared directory, we modified an open-source Gnutella client [4] that passively monitors all query and query-hit strings routed through it, analyzes the popularity of different objects, and automatically initiates the downloading of randomly chosen objects based on the popularity distribution. We run our honey-pot with a total of 1,533 different objects, most of them highly popular, and for each incoming request, we record request arrival time, object name, size of requested and served data chunk and transfer finish time for further analysis.

To avoid potential bias in data collection, we constructed multiple honey-pots at different hosts, each serving its own collection of objects, and each configured with different upper bounds for outgoing bandwidth and number of threads serving requests. To ensure we capture the behavior of busy server peers, most of these limits were set much higher than their default settings. In order to capture potential differences due to bandwidth classes, we also collected traces using a peer behind a cable connection. Some key parameters of our traces are summarized in Figure 1.

Connection Type	Number of Threads	Number of Objects	Number of Requests
100Mbps Ethernet	200	1,533	300,000
100Mbps Ethernet	100	1,533	150,000
100Mbps Ethernet	50	500	80,000
Cable Modem	20	1,533	40,000

Figure 1: Key parameters of collected traces from P2P servers. *Number of Threads* is the number of available server threads.

3 Server Workload Characterization

Server workload characterization forms the basis for any work on scheduling policies. In this section we address the following questions for the case of data-sharing P2P servers. We use the terms “job” and “request” interchangeably.

- What is the distribution of job interarrival time?
- Are the job arrivals independent?
- What is the distribution of job service time[2]?
- What is the likely performance bottleneck? To understand which of the P2P server’s resources needs to be scheduled, we need to understand which one is the bottleneck.
- What are the implications of our findings on P2P system scheduling?

3.1 Job Arrivals Form a Poisson Process

We characterized job interarrival times for P2P servers based on our collected traces. Figure 2 gives the complementary cumulative distribution function (CCDF) of job interarrival time at a P2P server for a typical trace. Notice that the vertical axis is logarithmic; the straight line of the CCDF curve strongly indicates that the arrival process can be modeled by an exponential. The least-squares curve-fitting using an exponential function, indicated by the dash-line in Figure 2, with coefficient of determination $R^2 = 0.9943$ quantifies our argument.

We tested the independence of job arrivals by computing the serial correlation of their interarrival times, as shown in Figure 3. Clearly, the correlation between any two separate interarrival times is effectively nil. Since each of our traces

[2]In this paper, we define job service time as the wall clock time it takes a server to finish sending data to a client over the Internet given the bounded outgoing bandwidth for the job. Similarly, the response time of a job is the sum of its service time and its total waiting time in the queue.

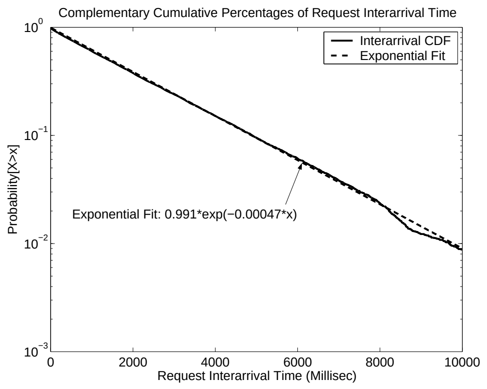

Figure 2: CCDF of interarrival time of requests to P2P server

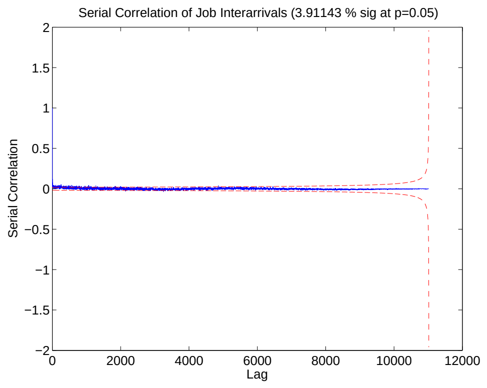

Figure 3: Serial correlation of interarrival time of requests to P2P server

exhibits similar behavior, job interarrivals for a P2P server can be well modeled as independent of each other, clearly a significant difference from the web server case. Exponentially distributed, independent interarrival times are the definition of a Poisson process.

Previous research [13, 10] has shown that Poisson processes are valid for modeling the arrival of user-initiated TCP sessions such as TELNET and FTP connections. HTTP arrivals, on the other hand, have been shown not to be Poisson. Deng et al [10] point out that the aggregated interarrival times of HTTP requests can better be modeled by a heavy-tailed Weibull distribution. This is because HTTP document transmissions are not entirely initiated by the user; some are automatically generated by the browser (requesting embedded files), resulting in a more bursty process.

Although P2P server requests, like web requests, are not solely initiated by the users, there are some interesting peculiarities of client peers that may explain the observed differences. For example, a client searching for a given object collects a set of candidate servers from which it later initiates parallel downloads. In addition, clients can abandon (switch) servers in the middle of a download, after finding an alternative source with higher available bandwidth [7].

3.2 Job Sizes are Pareto

Job size is another important property for queuing models. Interestingly, for P2P scheduling, there are three different possible definitions for job size: *full object size*, *requested data chunk size*, and *served data chunk size*. While the full object size is usually very large, most requested data chunks are small, covering only a small fraction of the whole object. More importantly, there is usually also a significant difference between the requested data chunk size and the actual served data chunk size. We discuss some possible explanations for this difference in Section 4.

The CCDFs of the three job sizes are depicted in log-log scale in Figure 4. As it is clear from the graph, the three often differ by several orders of magnitude. This clearly distinguishes P2P server requests from web requests and supports the argument for taking a closer look at the server side of P2P systems. For the remainder of this paper, we focus mainly on the requested and served data chunk sizes as these two are the main determinants of P2P server performance. For all of our traces, the distribution of these sizes can be modeled as a Pareto with high R^2 values (0.9293 and 0.9452 for the example in Figure 4).

Given the relatively limited number of requests (up to 300,000) in our traces, one may question if these results could not be strongly related to our particular traces. While this is certainly possible, the fact that we see similar results in each trace gives at least some confidence that we have captured general behavior. We

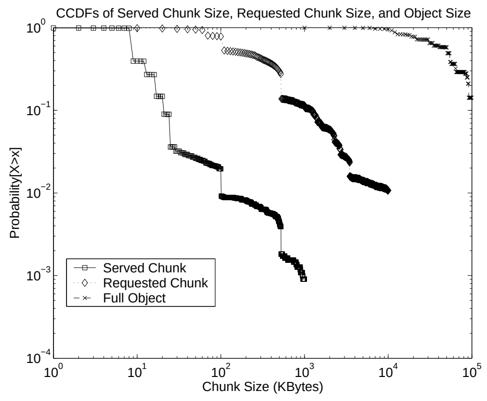

Figure 4: CCDFs of served data chunk size, requested data chunk size, and full object size

are working to expand the set of traces.

As we did for job arrivals in subsection 3.1, we also performed time series analysis for arrived job sizes. Our traces show that there are no correlations between arrived job sizes.

3.3 Server Resource Utilization Is Low

Despite their apparent similarities with web servers, the resource utilization of P2P servers could be quite different. Web servers typically try to serve requests as quickly as possible and, as it has been shown, their bottleneck resource is commonly the limited bandwidth of the outgoing link [6]. P2P servers, on the other hand, are normally run on the background of common users' machines and are thus more conservative in their use of resources.

To understand resource utilization on P2P servers, we instrumented our honey-pots [3] to periodically (every three seconds) record different metrics such as CPU and memory usage. Our traces show that even when our servers support 200 concurrent downloads and use up to 2 MBytes/second of bandwidth, CPU utilization is always between 1.2% and 20%, and memory usage is consistently below 20 MBytes. Thus, unlike the web server case, neither CPU, memory, nor bandwidth turn out to be the performance bottleneck for a P2P server, not even for the most popular of our honey-pots.

The service capacity of most P2P servers is limited by user-defined bandwidth

[3]Each of our servers is a dual 1 GHz Pentinum III machine with 1 GB RAM and two 30GB IDE disks running Red Hat Linux 7.3.

usage and concurrent server thread upper bounds. It is clear that the bottleneck resource we want to schedule is the number of server threads on a server, i.e., the number of concurrent jobs that a server can serve. Thus, our scheduling problem can be formulated as following: Given the total number of concurrent jobs that a server can take, how should we schedule the incoming jobs so that their mean response time is minimized?

4 Early results: Evaluation of Scheduling Policies

4.1 Scheduling Policies

A good scheduling policy should minimize the average waiting time without starving any jobs. More generally, fairness has several metrics, with the most recent work [6] using slowdown – defined as a request's response time divided by the time it would require if it were the sole request in the system.

The most commonly used scheduling polices are *Processor Sharing (PS)* and *First Come First Serve (FCFS)*. PS is commonly employed for CPU scheduling and in the current Apache web server, while FCFS is used by common Gnutella implementations such as Mutella, the implementation we use [4]. Neither of these policies makes use of other available information, such as size of a job, to improve performance. [4]

Shortest Remaining Processing Time (SRPT) has been studied since the 1960s [17]. For a general queuing system (G/G/1) Schrage [16] proved that SRPT is optimal in the sense that it yields – compared to any other conceivable strategy – the smallest mean value of occupancy and thus also of minimum waiting and delay time. Perera [14] and Harchol-Balter, et al [6] evaluated SRPT in term of fairness. Perera [14] studied the variance of delay time in $M/G/1/SRPT$ queuing systems and concluded that the variance is lower than FIFO and LIFO [14], while in [6] the authors proved that SRPT also outperforms PS in terms of mean slowdown, their fairness metric. SRPT has been successfully applied to a number of application areas. Bux [8], for example, introduced SRPT into packet networks using the message size as the service time. More recently, Harchol-Balter et al. [6] proposed the use of SRPT in web servers, relying on file sizes as the estimator of service time.

In this paper we introduce SRPT into P2P server-side scheduling. The adoption of SRPT faces some challenges, however. To begin with, ideal SRPT requires knowledge of requests' service times, something not available a priori. In addition, while it may be possible to estimate it [8, 6], the estimation is in itself challenging

[4]However, some P2P systems (such as eDonkey [2]) consider reputation (scores) as part of their scheduling policy.

due to the dynamic characteristics of P2P systems we have discussed.

4.2 SRPT Scheduling in P2P Systems

Since a typical P2P download request is for a specific chunk of the whole object, as described in Section 3, we could use the requested chunk size as a rough estimate of service time, and as the metric for SRPT scheduling. Unfortunately, as Figure 5 shows, there are only weak correlations between requested chunk size and either served chunk size or the real service time, which implies that requested chunk size may not be a good estimate of service time. This discrepancy between the requested and served chunk sizes could compromise the performance of SRPT [12].

Statistics	Service Time	Served Chunk Size	Requested Chunk Size
Service Time	1.0000	0.7023	0.2833
Served Chunk Size	0.7023	1.0000	0.2339
Requested Chunk Size	0.2833	0.2339	1.0000

Figure 5: Correlation coefficients between service time, served chunk size and requested chunk size.

Several characteristics of the P2P environment could help explain the weak correlation:

- A client can exit at any time during the data transmission.
- As already discussed, a P2P client can switch servers for a given data chunk before the request is completed. We speculate that the more popular the object, the more likely this switching is.
- Although each downloading process is supposed to share equal outgoing bandwidth from the P2P server, bandwidth bottlenecks along the path to the destination can make the individual download speed vary.

Figure 5 also shows a much stronger correlation between served chunk size and service time, indicating that chunk size can be a very good estimate for service time.

Despite the aforementioned discrepancies between requested and served chunk sizes and the weak correlation between requested chunk size and service time, it may be still interesting to evaluate SRPT performance using requested chunk size as its scheduling metric. Lu et al. [12] have studied the behavior of size-based schedulers such as SRPT with different correlation coefficient R between actual service time and estimated service time, concluding that SRPT is very robust in

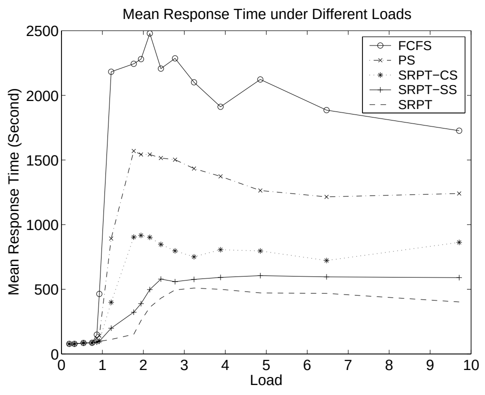

Figure 6: Mean Response time for different scheduling policies under varying load

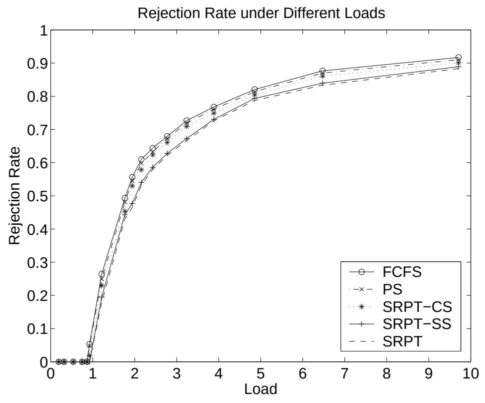

Figure 7: Rejection rate for different scheduling policies under varying load

the face of inaccurate job service time information and outperforms PS as soon as $R > 0.15$.

We explored how SRPT performs when using requested chunk size (CS) and served chunk size (SS) as the scheduling metric. For comparison purposes, we will also present the scheduling performance for ideal SRPT. The three scheduling policies are denoted as SRPT-CS, SRPT-SS, and SRPT, respectively. Notice that SRPT-CS can be directly implemented with current tools, while SPRT-SS would require an accurate estimator.

4.3 Performance Analysis

We built a general purpose queuing simulator to evaluate the performance of different policies, including PS, FCFS, SRPT-CS, SRPT-SS and ideal SRPT. All simulations were driven by our server-side request traces. For all of our simulations we set queue capacity to 500. A time slice of 0.01 seconds is used for PS. Besides our own work [12], we are not aware of other previous research addressing SRPT performance with inaccurate job size information.

Figure 6 gives the mean response time of the five scheduling policies handling all requests for a P2P server, with the system load varying between 0.1 to 10. The advantages of the three SRPT-based policies over PS and FCFS are clear, especially when the load is close to or above 1[5]. When the load is 1.76, for instance, mean response time is 2244.08 seconds under FCFS and 1569.89 seconds under PS. For SRPT-CS, SRPT-SS, and SRPT, however, the number drops to 903.61 seconds, 322.621 seconds, and 151.451 seconds, respectively. This confirms our expectations of SRPT performance.

Similar to what we have observed for web servers [12], even with only a weak correlation between requested chunk size and actual service time, SRPT-CS achieves considerable performance gains over both PS and FCFS. As would be expected, due to the strong correlation between served chunk size and service time, SRPT-SS performs significantly better and even approaches the performance of ideal SRPT under several different system loads.

The actual served chunk size, upon which SRPT-SS relies, is not known until the request is completed. However, we belive it should be possible to predict it fairly accurately. One possible way of achieving this, which we are currently exploring, is by finding correlations between object popularity and the level of discrepancy between the requested and served data chunk size. Another approach could be based on both the prediction of served chunk size and the download speed of the client. For the latter, the connection type of the client and the network distance of the client, such a domain-based scheduling [12], could be potentially applied.

[5]In this paper we are mostly interested in the case where server load is larger than 1, which is normal for a popular P2P server. Moreover, since job arrivals form a Poisson process and lack burstiness, the queue length shrinks abruptly when the load drops below 1. In all scheduling policies we evaluated, for example, the mean queue length drops to around 0.10 when system load is 0.75. As can be seen in Figure 6, SRPT-based scheduling policies still outperforms FCFS and PS when the load is smaller than 1, as long as there are jobs waiting in the queue.

4.4 Fairness Concerns

One major concern with SRPT scheduling is that it is possible to design an adversarial workload in which SRPT leads to starvation of large jobs. That is, SRPT can be made to behave unfairly. Fortunately, previous research on M/G/1 queuing systems with comparable workloads have shown that the starvation does not occur [14, 6]. Perera [14] proves that the variance of delay time of ideal SRPT is smaller than that of FIFO and LIFO, while Harchol-Balter [6] shows that the mean slowdown of ideal SRPT is actually smaller than that of FCFS and PS. In the context of P2P server scheduling, we consider fairness issues of a scheduling policy from three different aspects: mean slowdown of $large$ jobs, rejection rate of requests, and distribution of rejected job size.

Figure 7 shows the rejection rates for the five policies under various system loads. We can see that SRPT actually results in the lowest rejection rate; SRPT-CS and SRPT-SS also reject fewer jobs than FCFS and PS. Our simulations also demonstrate that the distribution of rejected job size is almost identical for all evaluated scheduling policies. Moreover, under various system loads, SRPT-based scheduling policies yield lower mean slowdown for large jobs. When the system load is two, for instance, the mean slowdown for the top 10% largest jobs in the system are: 15.496 (FCFS), 25.615 (PS), 10.723 (SRPT-CS), 8.741 (SRPT-SS), and 7.707 (SRPT).

5 Conclusions and Future Work

The server-side of P2P systems often turns out to be the performance bottleneck. Surprisingly, it has received little attention from the research community. In this paper, we start this exploration by looking at the problem of download request scheduling. We collected trace data of P2P download requests experienced by individual P2P servers and performed analysis and modeling of this server workload. We proposed two SRPT-based scheduling policies and show their advantages through trace-driven simulations.

Analysis of several inherent characteristics of P2P server requests also reveals considerable room for improvement in estimating request service time, which would let us approach the performance of ideal SRPT. Two possible approaches for estimating service time we plan to explore include: predicting served data chunk size based on object popularity and requested chunk size, and predicting transfer rate based on client type and Internet path characteristics.

We also identified other interesting directions of future work in P2P server-side scheduling:

- Deeper and more thorough analyses of fairness issues for various scheduling policies.
- P2P server trace collection from different hosts, increasing both the geographical and connection variety.
- Modeling and scheduling for cooperative uploading/downloading, as employed in [2, 1].
- Implementation and evaluation of various scheduling models in P2P software and its evaluation on large-scale Internet testbeds.

References

[1] Bittorrent homepage, 2004. http://bitconjurer.org/BitTorrent.

[2] eDonkey homepage, 2004. http://www.edonkey2000.com.

[3] Kazaa homepage, 2004. http://www.kazaa.com.

[4] Mutella homepage, 2004. http://mutella.sourceforge.net.

[5] ALMEIDA, J., DABU, M., MANIKUTTY, A., AND CAO, P. Providing differentiated quality-of-service in web hosting services. In *Proc. 1st WISP* (1998).

[6] BANSAL, N., AND HARCHOL-BALTER, M. Analysis of SRPT scheduling: Investigating unfairness. In *Proc. ACM SIGMETRICS* (2001).

[7] BERNSTEIN, D. S., FENG, Z., LEVINE, B. N., AND ZILBERSTEIN, S. Adaptive peer selection. In *Proc. 2nd IPTPS* (2003).

[8] BUX, W. Analysis of a local-area bus system with controlled access. *IEEE TC 32*, 8 (1983), 760–763.

[9] CROVELLA, M., FRANGIOSO, R., AND HARCHOL-BALTER, M. Connection scheduling in web servers. In *Proc. USENIX USITS* (1999).

[10] DENG, S. Empirical model of WWW document arivals at access links. In *Proc. IEEE ICC* (1996).

[11] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. 19th ACM SOSP* (2003).

[12] LU, D., SHENG, H., AND DINDA, P. A. Effects and implications of file size/service time correlation on web server scheduling policies, in submission, 2004.

[13] PAXSON, V., AND FLOYD, S. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM TN 3*, 3 (1995), 226–244.

[14] PERERA, R. The variance of delay time in queueing system M/G/1 with optimal strategy SRPT. *Archiv fur Elektronik und Uebertragungstechnik 47*, 2 (1993), 110–114.

[15] SAROIU, S., GUMMADI, K. P., DUNN, R. J., GRIBBLE, S. D., AND LEVY, H. M. An analysis of internet content delivery systems. In *Proc. 5th USENIX OSDI* (2002).

[16] SCHRAGE, L. E. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research 16* (1968), 678–690.

[17] SCHRAGE, L. E., AND MILLER, L. W. The queue M/G/1 with the shortest remaining processing time discipline. *Operations Research 14* (1966), 670–684.