In Proceedings of IEEE Infocom '98, San Francisco, CA

The Network Effects of Prefetching

Mark Crovella and Paul Barford
Computer Science Department
Boston University
111 Cummington St, Boston, MA 02215
{crovella,barford}@cs.bu.edu

Abstract— **Prefetching has been shown to be an effective technique for reducing user perceived latency in distributed systems. In this paper we show that even when prefetching adds no extra traffic to the network, it can have serious negative performance effects. Straightforward approaches to prefetching increase the burstiness of individual sources, leading to increased average queue sizes in network switches. However, we also show that applications can avoid the undesirable queueing effects of prefetching. In fact, we show that applications employing prefetching can significantly improve network performance, to a level much better than that obtained without any prefetching at all. This is because prefetching offers increased opportunities for traffic shaping that are not available in the absence of prefetching. Using a simple transport rate control mechanism, a prefetching application can modify its behavior from a distinctly ON/OFF entity to one whose data transfer rate changes less abruptly, while still delivering all data in advance of the user's actual requests.**

I. Introduction

Prefetching is an important technique for reducing latency in distributed systems. In distributed information systems like the World Wide Web, prefetching techniques attempt to predict the future requests of users based on past history, as observed at the client, server, or proxy [2], [11]. These techniques are speculative, in the sense that if predictions are incorrect then additional useless traffic is added to the network. Considerable previous work has evaluated the benefits of prefetching in various distributed and parallel systems; most of that work has focused on the addition of useless traffic to the network as the principal cost of prefetching.

In this paper we focus on a different cost of prefetching: increases in network delays. We show that even when prefetching adds no useless traffic to the network, it can have serious performance effects. This occurs because prefetching changes the pattern of demands that the application places on the network, leading to increased variability in the demands placed by individual sources, and in network traffic as a whole. Increases in traffic variability (or "burstiness") directly results in increased average packet delays, due to queueing effects. In general, the straightforward application of prefetching can be seen to increase the coefficient of variation of the arrival process of packets from a single source. This is because prefetching will increase the length of long packet interarrivals while increasing the number of short packet interarrivals. Increasing the coefficient of variation naturally increases queueing delays.

This work was supported in part by NSF grant CCR-9501822 and by Hewlett-Packard Laboratories.

We focus on the World Wide Web as our application of interest; prefetching for the Web is an active area of study that has considerable practical value. Starting from traces of Web client activity, we simulate the performance of a simple network as it satisfies user requests, using a detailed packet-level simulator that explicitly models the flow control algorithms of TCP Reno. In the first part of this paper we show that straighforward prefetching algorithms for the Web, even if they add no additional traffic to the network, can increase packet delay considerably. Our simulations indicate that aggressive prefetching can increase average packet delays by a factor of two to four, depending on network configuration.

For the case of the Web, these effects can be understood more precisely using the analytic framework provided by self-similar traffic models [10]. Such models have shown that if individual applications exhibit ON/OFF behavior in generating traffic, such that either ON or OFF periods are drawn from a heavy-tailed distribution with $\alpha < 2$, then the aggregate traffic can be expected to show self-similar scaling properties with scaling parameter $H > 1/2$ [15]. Recent evidence indicates that the Web in particular seems to exhibit heavy-tailed ON periods, and that this may be due to the sizes of files transferred via the Web [3]. We show evidence that the effect of prefetching in distributed information systems with heavy-tailed ON times is to lengthen the ON and OFF periods, resulting in increased burstiness at a wide range of scales.

However, in the second part of this paper we show that an application like the Web can avoid the undesirable queueing effects of prefetching. In fact, we show that applications employing prefetching in combination with a simple transport rate limiting mechanism can significantly *improve* network performance, to a level much better than that obtained without any prefetching at all. This technique is possible for the same reasons that prefetching is useful: there is typically some "idle" time between the completion of one data transfer and the user's initiation of the next transfer. This time corresponds to the OFF time of the application in the ON/OFF framework.

Using a transport rate limiting mechanism, an application (such as a Web browser) can prefetch in such a way so as to still obtain all data in advance of user requests, but in a manner that extends the data transfer over the entire inter-request interval. When the accuracy of prefetch prediction is high, this approach can radically decrease the application's contribution to queueing delay. The reason that this strategy works can again be understood in terms of the ON/OFF model for self-similarity in Web traffic. Using rate-controlled prefetching, an intelligent application can modify its behavior from a distinctly ON/OFF entity to one whose data transfer rate changes less abruptly, leading to a smoother overall traffic flow on the network, and significantly increased network performance.

We explore these ideas in the second half of the paper. Using our simulations, we show that network performance in terms of queueing delay could be much better than the no prefetching case, without missing a significant fraction of deadlines, if browsers employ rate-controlled prefetching; such performance improvements are based on perfect knowledge of the future sequence of requests to be made by users. We then show more realistic strategies that correspond to prefetching techniques that have been proposed in the literature, and we show that they can achieve results close to that of the ideal case. Finally, we evaluate the degree to which prefetching must be effective in order for our methods to succeed, and compare these with empirically measured prefetching effectiveness for Web browsers.

Our results suggest that while simple prefetching (as is likely to be implemented in Web browsers) can degrade network performance, rate-controlled prefetching has the potential to significantly smooth traffic due to the World Wide Web. Since such traffic appears to be quite bursty without such controls [3], an understanding of the effects of prefetching on burstiness is important, and in particular methods that allow Web applications to smooth their network demand are of considerable interest.

II. Background and Related Work

The literature on prefetching is large; a good example article is [14]. That article points out the performance risks of prefetching due to increased data transfer traffic.

More recent work on file prefetching [9] presents a method which uses access trees to represent file usage patterns. Using that approach, measurements show that future file accesses can be predicted with an accuracy of around 90%. Although we do not propose a mechanism for prefetching, one such as this could be used as a means for prefetching in distributed information systems like the Web. The high hit rate obtainable suggests that our rate-contolled prefetching policies could have significant payoff in practice.

A number of authors have presented prefetching methods specifically for the Web. In [4] the notion of prefetching by Web clients is presented. In that work, Markov chains are proposed as a mechanism for determining which files to prefetch. The Markov chains are constructed based on prior access patterns of individual users. Simulations suggest that this method can also provide relatively high prefetching hit rates—between 80% and 90%.

In [2] the notion of speculative prefetching by Web servers is presented in the context of replication services. This work shows that server information can also be profitably be used to increase prefetching hit rate. In addition, [11] simulates prefetching for the Web and shows that it can be effective in reducing latency.

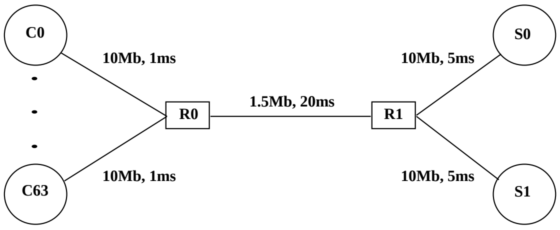

Fig. 1. Simulated Network Configuration

However, none of these proposals for Web prefetching consider the increase in burstiness of traffic caused by prefetching. Web traffic has been shown to be bursty at a wide range of scales ("self-similar") [3] and so this is an important issue. A number of previous papers have shown that the effects of self-similarity on network performance is primarily to increase queueing delays [5], [12] and so that is the network performance metric that we focus on in this paper. In addition, there have also been a number of studies of smoothing techniques for video streams, which are also bursty sources [7], [13].

The self-similarity of network traffic has been attributed to the behavior of individual sources. If sources follow an ON/OFF behavior, in which they transmit packets during an ON period and are silent during an OFF period, and if either ON or OFF periods (or both) are drawn from a heavy-tailed distribution (one whose tail follows a power-law with exponent less than 2) then the aggregation of many such sources will lead to self-similar traffic [15]. This model sheds light on why the rate-controlled prefetching techniques we propose can be effective at smoothing self-similar traffic: under rate-controlled prefetching, sources are no longer either ON or OFF but rather generate flows in which data constantly "trickles" into the application.

III. Experimental Environment

In this section we describe the simulation we used in our experiments.

A. Simulated Network

In order to assess the network effects of document transfers, we used the `ns` network simulator (version 1.0), developed at LBNL [6]. `Ns` is an event-driven simulator that simulates individual packets flowing over a network of hosts, links, and gateways. It provides endpoint agents that can use any of several types of TCP as their transport protocols; we used TCP Reno. In particular, `ns` simulates TCP Reno's flow control features: Slow Start, Congestion Avoidance, and Fast Retransmit/Recovery. A test suite describing validation results from the simulator can be found in [6].

This study used the simple network shown in Figure 1. The network nodes consist of 64 clients (C0 through C63), two routers (R0 and R1), and two servers (S0 and S1). Links between the servers and clients are configured to approximate Ethernet characteristics while the central link is configured to have less bandwidth, and therefore to be the bottleneck in all our simulations. The Figure shows the

Hour Number	Bytes Transferred	Files Transferred	Sessions	Avg. No. of Files/Session
Hour 1	48.9M	3007	48	63
Hour 2	47.5M	1304	43	30
Hour 3	39.2M	2011	89	23
Hour 4	25.1M	2126	46	46
Hour 5	24.1M	1799	80	23

TABLE I
CHARACTERISTICS OF HOURLY WEB CLIENT TRACES

baseline configuration of the bottleneck, which is 1.5Mbps; but some of the experiments we report used bandwidths of 750kps or 150kpbs for this link. Although this network is simple, it allowed us to simulate the effects of interest: the effects of flow control due to bandwidth limitations at the bottleneck, and the queueing of packets in router buffers.

In our simulations, clients make requests for files to be delivered from servers. As a result, the vast majority of traffic in the simulation flows from the servers to the clients, and therefore the buffers that exhibit important queueing effects are those in router R1. In the baseline configuration R1 has 32KB of buffer space; however some experiments increase this buffering, up to a maximum of 512KB. We do not simulate any resources internal to the servers; therefore the only limitation on server performance is the bandwidth of its outgoing link.

The default packet size is 1024 bytes, consisting of 1000 bytes of payload and 24 bytes of header. Some simulations vary the payload size, but all packets have 24 byte headers.

B. Workload Characteristics

The workload used to simulate user requests from the Web was a set of traces of Web sessions made at Boston University and available from the Internet Traffic Archives [8]. Each line of a trace file contains a session ID, the size of the file transferred, the start and finish time of transfer (measured to 10ms accuracy), and the URL of the file requested (which was not used in our simulation). We will refer to each line from a trace file as a *request* and to the set of all requests made during one execution of the browser as a *session.* Each trace file consists of the all the Web requests made in a local area network during the span of one hour. We studied five such hours, which are summarized in Table I.

To use these traces in our simulated network we assigned each session to a particular client-server pair. Sessions were distributed between servers to keep them in approximate balance, and were distributed among clients to keep them in approximate balance as well. Within each session, file transfers were initiated from the server to the client based on the size and time recorded in the trace.

Statistics were gathered during each experiment to measure the performance of the network. The principal performance metric of interest was mean queue size at the bottleneck router (R1); however we also tracked the number of bytes transferred over the bottleneck link and packet drops. In addition, the capability to track the completion of transmission of individual files was added to the system so that deadlines for the transmission of files could be established. Our intent was not to compare deadlines that were measured in the traces to those measured in the test network, since the environments are different, but simply to use the trace deadlines for the arbitrary deadines necessary when implementing rate controlled prefetching.

IV. PREFETCHING

In this section our goal is to show the general effects of prefetching on network performance as measured by mean queue size in the bottleneck router, and to explore the reasons behind the effects we observe.

Since our goal is to demonstrate the general effects of prefetching on network performance, we assume an idealized form of prefetching. At the start of each Web browser session, all files to be requested during the session are prefetched. This results in a transfer of a large number of files; the average number of files per session for each hour is shown in Table I and varies between 23 and 63. This type of prefetching is unrealistic in at least two aspects: first, it assumes that all files to be requested by a user can be predicted with perfect accuracy, and second, it assumes that all requests can be predicted based on the first request made by the user. The first aspect causes our assessment of network effects of prefetching to appear overly optimistic, because any incorrect predictions will add traffic to the network. The second aspect causes our assessment of network effects of prefetching to be pessimistic, since a more realistic policy would spread trasnsfers more evenly through the session. As a result, the absolute performance impacts of this policy are not our main focus in this section; rather we are concerned with demonstrating the effect and exploring the reasons behind it. In the next section we explore more realistic policies that 1) do not assume that more than one file is prefetched at a time and 2) do not assume perfect prediction of user behavior.

Our first results compare prefetching with the baseline request pattern for a typical configuration (bottleneck link speed = 1.5 Mbps, router buffer = 32 KB). We compare the average queue length in the router (R1) for all five of the trace hours, for both prefetching and for the baseline request pattern, as measured in bytes. The results are shown in Figure 2 (left). This figure shows that over a range of utilizations, and for all the sample traces we measured, that prefetching significantly increases the mean queue size, and thus the average delay experienced by a packet in transit.

To examine whether our results were particularly sensitive to network configuration, we varied network configuration along two dimensions: bandwidth and buffering.

First, we adjusted the speed of the bottleneck (central) link in the network. The results are shown in Figure 2 (middle), which plots the increase in mean queue size due to prefetching for a single hour (Hour 1), as the bandwidth of the the bottleneck link is varied.

Figure 2 (middle) shows that as the bottleneck bandwidth decreases, the relative effects of prefetching on queue size appear to moderate somewhat. This seems to occur

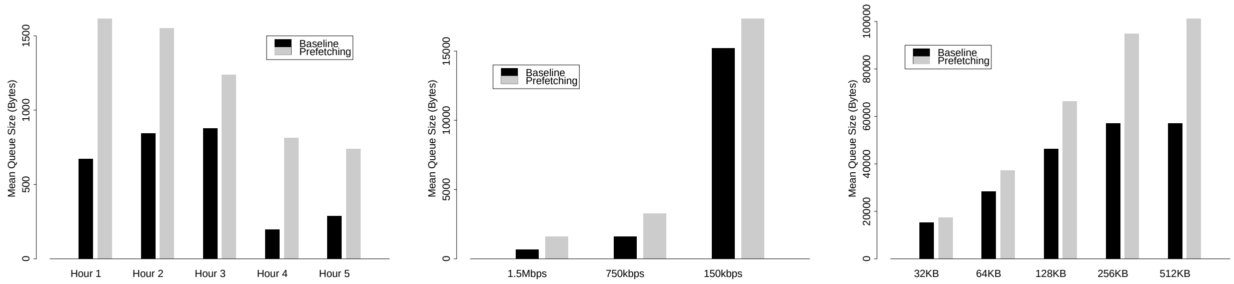

Fig. 2. Effect of Prefetching on Mean Queue Size for all hours (left), Varying Bottleneck Speeds (middle), and Varying Buffer Sizes (right)

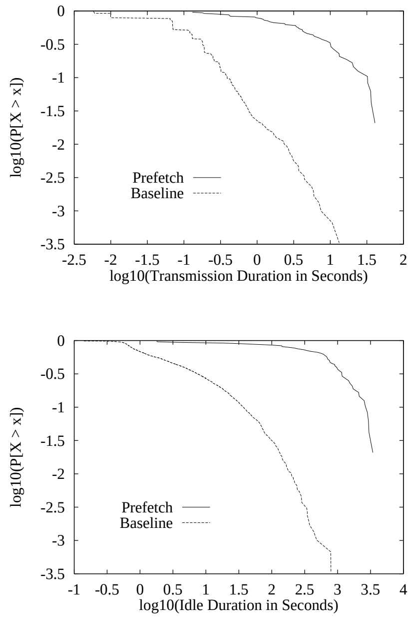

Fig. 3. Distributions of ON Times (upper) and OFF Times (lower) For Baseline and Prefetching

because router buffers are kept more uniformly full at a higher average level. Note that in the case of the 150kbps, average queue length is about half of the total amount of buffer space available (32KB); as a result when a burst of packets arrives, instead of being stored, packets must be dropped and so do not contribute to average queue length.

Second, using a 150kpbs bottleneck link, we adjusted the amount of buffering at the bottleneck link from 32KB up to 512KB. The results are shown in Figure 2 (right). This figure shows that the detrimental effects of prefetching return as we increase the buffer size at the bottleneck router, even with a relatively slow link. Thus, both the network modification experiments support the notion that the effects of prefetching will be greatest for network configurations with either relatively low utilization or relatively large amounts of buffering.

To understand the reasons behind the increased queueing delays when prefetching is in effect we examined the individual sessions as ON/OFF sources. Previous work has suggested that the sessions present in these traces show heavy-tailed ON/OFF behavior [3]. Figure 3, shows evidence of long, power-law tails in the distributions for both ON and OFF times. Durations of transfers of files or sets of files (when prefetching) correspond to ON times in the ON/OFF framework, idle periods when no transfers are taking place correspond to OFF times. Figure 3 shows how strongly aggressive prefetching affects the distribution of ON and OFF times of the browser. In both cases the distributions are shifted strongly toward larger values. The median ON time changes from 0.12 seconds for the baseline case to 4.2 seconds for prefetching; while the median OFF time increases from 2.48 seconds to 755 seconds.

These results show that individual sources are becoming much burstier as a result of prefetching: typical ON and OFF times are drastically increasing. Since ON and OFF times show heavy-tailed distributions the increase in burstiness is evident at a wide range of scales, and affects queueing in the router in a significant way.

V. Rate Controlled Prefetching

The preceding section showed that the effect of prefetching on network performance is to increase queueing in routers, and thus average packet delays. This may seem to decrease the incentive to use prefetching in distributed information systems like the Web. However, in this section we show that prefetching offers opportunities to improve network performance to a level much better than that obtained without prefetching, by using *rate-controlled* prefetching. In particular, our results suggest that rate-controlled prefetching might profitably be added to Web browsers to significantly smooth the burstiness of Web traffic.

The feasibility of rate-controlled prefetching is based on the observation that when prefetching a data object, it is not necessary to transfer the object at the maximum rate supported by the network; rather, it is only necessary to transfer it at a rate sufficient to deliver it *in advance of the user's request.* Thus the central idea of rate-controlled prefetching is to lower the transfer rate during prefetching to a level such that (ideally) the prefetch is initiated as early as possible, while the last byte of the data object is delivered just before the object is requested.

The success of rate-controlled prefetching is based on the fact that in distributed information systems like the Web, user-induced delays are common between document transfers. User-induced delays will be typically be quite long relative to document transfer times; as a result, prefetching rates can be drastically reduced below the network's maximum. For example, in Hour 1 of the traces we used, the aggregate ON time of all sessions was 651 seconds, while the aggegate OFF time was 46886 seconds. The goal of rate-controlled prefetching is to spread the 651 seconds of data transfers as smoothly as possible over the much larger span of 46886 seconds without missing any deadlines for document delivery to the user.

Rate-controlled prefetching adds a new requirement to the prefetching algorithm: in addition to a prediction of the next document to be requested, the algorithm must make some estimate of the time until the next request will be made (OFF time duration). We think this is feasible; previous research on prefetching has focused on predicting the identity of the next transfer, with reasonable success. We expect that such algorithms could be likewise reasonably successful at addressing the related problem of predicting OFF time duration. Explicitly evaluating the potential for predicting OFF time duration is treated in Section VI. We also show later in this section that our results are not dependent on exact prediction of OFF time duration; approximate predictions (within a factor of 2, for example) are quite sufficient to achieve significant performance gains using rate-controlled prefetching.

In this section we assume a prefetching algorithm that is more realistic than was used in the previous section. First, in all of the results in this section, documents are prefetched one at a time — that is, all available knowledge of prior user requests is assumed to be used before deciding on a document to prefetch. Thus, after each document is delivered to the user, the system attempts to prefetch only the single next document that the user will request. Second, we do not assume that the prefetching algorithm can predict future requests with perfect accuracy. Instead we evaluate all of our results as a function of varying hit rate of the prefetching algorithm. The hit rate of the prefetching algorithm is the percent of predictions that the algorithm makes correctly. If a prediction is incorrect, then at the point the user makes the next request the correct document is transferred without applying any rate controls (that is, at the maximum rate the network can deliver at the time).

Evaluating varying hit rates means that network performance (as measured by mean queue size) can degrade for two reasons: increased burstiness of the traffic process (as before), and increased network utilization due to wasted traffic—caused by incorrect prefetch predictions. In our simulations, incorrect predictions are made in the same way regardless of whether rate control is in effect. Thus, in comparing rate-controlled prefetching with simple prefetching, the increase in network traffic due to hit rates less than 100% is the same in both cases.

A. Window Based Rate-Controlled Prefetching

To bound the rate at which transfers take place during prefetching we employed a simple method: limiting the maximum window size that can be used by TCP. More sophisticated methods are possible, but were not needed to demonstrate our results. This is a method that could be implemented in practice by causing the client to advertize a limited window at connection establishment.

We used a simple rule for moderating transfer rate via window size. We wish to approximate a transfer rate determined by P/T where P is the number of packets necessary to transfer the document, and T is the time between the end of the previous document request and the initiation of the next document request. The window size W is then determined by

$$W = \lceil P \cdot R/T \rceil \tag{1}$$

where R is the round-trip time of the path between client and server. In general, R is unknown at the start of the connection, but can be estimated quickly once the connection is established. In practice we did not model such an adaptive approach; instead we used a simpler method of setting R based on the product of the known path round-trip delay and a factor that corrects for congestion effects. Use of this factor also allowed us to test the method's senstitivity to accurate prediction of OFF time, because increasing the congestion factor caused transfers to terminate more quickly, simulating increased burstiness due to an inaccurate OFF time prediction.

Thus, in our simulation, rate-controlled prefetching works as follows. At the completion of each document transfer, the next document transfer is started, with maximum TCP window size W determined by Equation 1. At the time when the next request is made by the user, the simulation determines probabilistically whether that request will be considered a prefetch hit. That is, the most recently prefetched document is considered to be the correct document with probability equal to a predetermined prefetch hit rate. If the prediction is incorrect, the correct document is then transferred at maximum rate as would be the case in the absence of prefetching. If the prediction is correct, the simulation begins prefetching the next document.

The effects of rate-controlled prefetching on mean queue length can be seen in Figure 4. In each plot we have shown the effects of three policies: Baseline (no prefetching at all), Simple Prefetching (prefetching at unlimited rate), and Rate-Controlled Prefetching. The figure shows that rate-controlled prefetching is always better than simple prefetching. Interestingly, it also shows that rate-controlled prefetching is usually better than no prefetching at all, even though rate-controlled prefetching generally adds additional traffic to the network (that is, when prefetching hit rate is less than 100%). Note that the minimum value for the simple prefetching case corresponds to the baseline case: when prediction accuracy is 100%, simple prefetching adds no additional traffic to the network and does not change traffic burstiness significantly.[1] In fact, over all the hours we studied, if prefetching hit rate is above 80%

[1]Unlike the previous section, in which multiple files were prefetched, which increased the burstiness of individual sources as well as overall traffic.

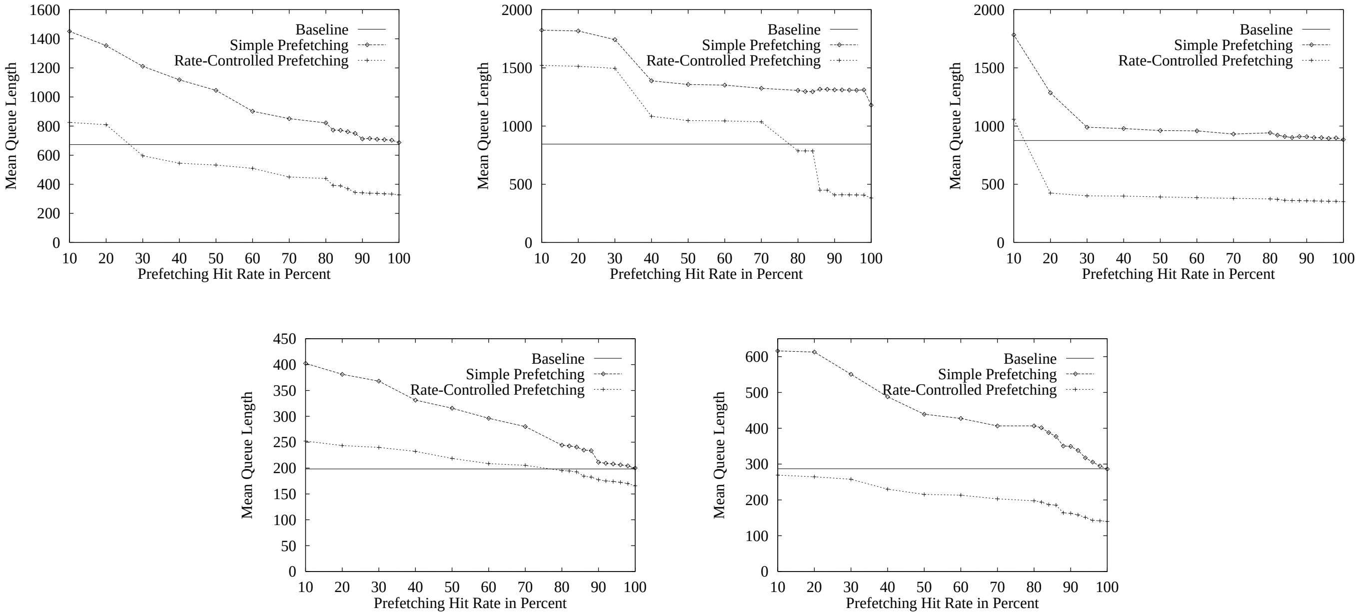

Fig. 4. Effect of Rate-Controlled Prefetching on Mean Queue Length for all five Hours

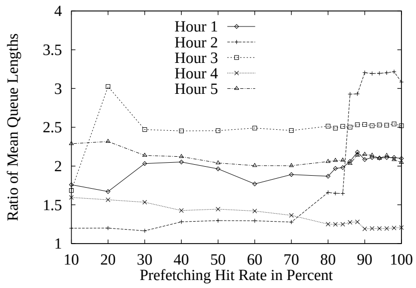

Fig. 5. Improvement Due to Rate Control in Prefetching

then queueing delay is lower for rate-controlled prefetching than for no prefetching at all. In addition, for three out of the five hours we studied, queueing delay is lower for rate-controlled prefetching even for very low prefetching hit rates—as low as 10-30%.

In general, the simple rate-controlled prefetching scheme we describe here seems to be able to reduce mean queue size significantly. In Figure 5 we plot the ratio of mean queue size without rate control to mean queue size with rate control, as a function of prefetching hit rate, for all five hours. The figure shows that rate control seems to always improve the performance of prefetching, usually by a factor of at least two, and usually fairly consistently over a range of prefetching hit rates.

These results suggest that even relatively inaccurate prefetching, if rate-controlled, may be a desirable feature of an application that is interested in reducing network induced delays, and reducing the variability of such delays. In addition, these results suggest that if prefetching is to be implemented in an application like a Web browser, then rate-control is a very desirable feature.

B. Payload Based Rate-Controlled Prefetching

One observation that can be made about the window-based rate control described in the last subsection is that there is a minimum transfer rate for prefetching that is determined by D/R where D is the data payload of a single packet. That is, transmission rate cannot easily be reduced below that of one packet per round trip. However it may be the case that ideal smoothing of prefetched data requires a transfer rate less than this minimum.

To test whether even slower transfer rates are helpful, we implemented an alternative rate-control scheme, based on modifying the size of each packet's payload. In this scheme, we do not limit window sizes, but the payload of each packet is scaled to reduce throughput. Just as in the window-based case, the scaling is performed based on the estimated time until the next user request. This method provides the opportunity for finer control of transport rate, and for reducing transport rate to a level much lower than the window-based method.

The results of this approach are shown in Figure 6. Note in this figure that the hit rates studied have been restricted to the range 80% to 100%. The figure shows that for high prefetching hit rates, payload based rate control has the potential to significantly improve network queueing delay—to an almost negligible level. This occurs because the packets being queued are quite small and occupy much less buffer space per packet. However the figure also shows that for slightly lower hit rates—in many cases, less than about 90%, that payload based rate control performs worse than the window based approach.

The reason that the queueing characteristics of payload based rate control degrade so quickly is that the payload

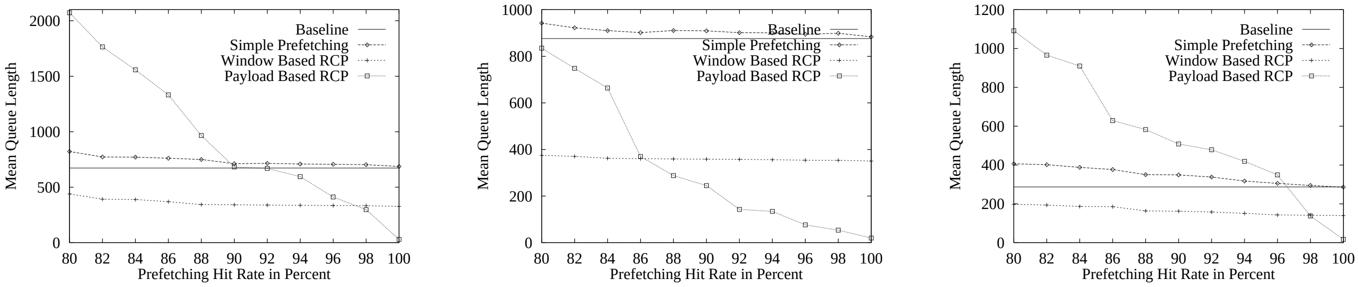

Fig. 6. Comparison of Payload and Window Based Rate Control in Prefetching for Hours 1 (left), 3 (middle) and 5 (right)

method adds significant additional traffic to the network, even when prediction is perfect. This is because the ratio of header data to payload data increases drastically for transfers with highly limited rates. In our simulations, payload based rate control can as much as double the total byte traffic on the network; as a result, network queues can build up quickly when predictions are not perfectly accurate. Thus we conclude that while payload based rate control can result in better queueing performance at high prefetching hit rates, the additional traffic added by the technique makes its use undesirable.

VI. Estimating Transfer Rates

The keys to being able to use our rate throttling ideas is the implementation of: 1) a document prediction mechanism within a Web browser, and 2) a transfer rate prediction mechanism within a Web browser. A number of mechanisms for document prediction have been proposed; see Section II for examples. Therefore in this section we concentrate on determining the feasibility of predicting necessary transfer rates. The best transfer rate for the purpose of prefetching is dependent on the predicted time between document requests (OFF times). OFF times in network traffic have been analyzed in [3], [15], [1]; however these studies have not related OFF times to an independent variable which can be used for predictive purposes.

Our approach to the transfer rate prediction problem is to attempt to relate OFF times to the size of the file transfered immediately before the start of an OFF period. The document prediction mechanism could be used to extract file size information; once it is known we could then predict the length of the ensuing OFF period. Using the five busy hours studied in this work, we generated a data set consisting of file sizes and the OFF periods which followed their accesses. A simple least squares regression on this data set results in the following model:

$$OFF = 9.78 \times 10^{-5} \cdot size + 24.9 \qquad (2)$$

This formula gives an OFF time prediction in seconds based on file size in bytes. The regression yielded an R^2 of 0.14 so clearly there is very low correlation between OFF times and file sizes. However, since there is some correlation, there may be an opportunity for prediction. Therefore, we tested the effectiveness of this method for the purpose of throttling data in simulation. Since our goal is to transfer documents with minimal network impact, we evaluate how well we can reduce queue sizes using this formula for OFF time prediction without missing document transfer deadlines.

In this set of simulations, instead of considering each file independently (as was done in prior simulations) we consider transfers of files in terms of groups as follows. If the measured OFF times between consecutive transfers was less than a threshold value (one second in our case), then we consider the aggregate of bytes as a single transfer. A deadline is the actual time at which the next file transfer was requested (preceded by an OFF time greater than the threshold). This was done in order to more accurately model OFF times as the time that users "think" between file accesses (an attempt to exclude latency between inline document transfers). Since the R^2 for the regression was so low, we decided to evaluated the effectiveness of the OFF time prediction model by including scaling factors in the analysis. Each OFF time predicted by the model is simply divided by the scaling factor. The effect of scaling is as follows: when the scaling factor is increased, it will result in a shorter projected OFF time and thus would increase the TCP window (or payload) size for prefetching (see equation 1). We would expect the effect of compressing the time for transfer as the scaling factor is increased would be to increase queue sizes. However, the benefit of the higher scaling factor should be to reduce the number of transfer deadlines missed. Simulations were run testing 1-ahead prefetching using projected OFF times with a 90% document hit rate accuracy. We compare these results with those for 1-ahead prefetching with a 90% hit rate when using measured OFF times as a baselines for these simulations.

Figure 7 shows the effect of varying scaling factor for predicted OFF times versus mean queue size. Note that in this figure, "Baseline" refers to the case in which off times are known exactly, as in Section V. For all five hours there is still a significant reduction in mean queue size using predictive values for OFF times regardless of scaling factor used. As might be anticipated, when the scaling factor is increased, the mean queue size generally increases since the TCP window size is also increased. The scaling factor has an inverse effect on missed deadlines as can be seen in Figure 7. The baseline case shows what might be expected

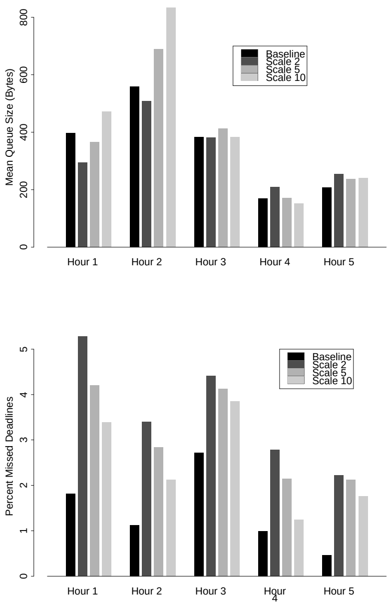

Fig. 7. Effect of Predicted OFF times on Mean Queue Size (top) and Missed Deadlines (bottom)

as the best case in terms of percentage of missed deadlines when actual OFF times are used to set the TCP window size. In all of the predictive model experiments, the percentage of missed deadlines declines as the scaling factor is increased (which again would be anticipated given our formula for setting TCP window size). In absolute terms, the number of deadlines missed when using the predictive model for OFF times is still a very small percentage of the overall number of files transferred.

Based on these results, we feel that a routine which predicts OFF times based on file sizes could be easily implemented in a browser and using a scaling factor of 2 and an off time threshold of 1 second. This would provide the best queue reduction effect while only missing a minimal number of deadlines. This along with a mechanism for predicting document requests could then effectively be used to implement rate throttling.

VII. Conclusions

In this paper we've shown how prefetching in a distributed information system like the World Wide Web can affect the queueing behavior of the network it uses. We started by showing that prefetching as it is usually implemented—that is, the transfer of multiple files together in advance of request—can create an undesirable increase in burstiness of individual sources. Because such sources in a distributed information system like the World Wide Web may exhibit heavy-tailed ON/OFF behavior, increases in source burstiness result in increases in variability of aggregate traffic at a wide range of scales. This makes straightfoward approaches to prefetching, even when their predictions are quite accurate, less attractive from the standpoint of network performance.

However, we have also shown that prefetching offers an opportunity for traffic shaping that can improve network performance. The periods between document transfers at individual sources may often be very long compared to the durations of transfers themselves. By prefetching documents at a controlled rate during the in-between time, applications can exploit an opportunity to *decrease* their individual burstiness as compared to the non-prefetching case. As a result, applications employing rate-controlled prefetching can have the best of both worlds: data transfer in advance of user request, and better network performance than is possible without prefetching.

References

[1] M. Arlitt and C. Williamson. Web server workload characterization: The search for invarients. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 126–138, May 1996.

[2] Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of CIKM'95: The* 4^{th} *ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[3] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 160–169, May 1996.

[4] C. R. Cunha. *Trace Analysis and its Applications to Performance Enhancements of Distributed Information Systems.* PhD thesis, Boston University, Boston, Massachusetts, 1997.

[5] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, April 1996.

[6] Sally Floyd. Simulator tests. Available at ftp://-ftp.ee.lbl.gov/papers/simtests.ps.Z. ns is available at http://-www-nrg.ee.lbl.gov/nrg., July 1995.

[7] M. Garrett and W. Willinger. Analysis, modeling and generation of self-similar vbr video traffic. In *ACM SIGCOMM*, pages 269–280, August 1994.

[8] Internet Town Hall. The internet traffic archives. Available at `http://town.hall.org/Archives/pub/ITA/`.

[9] Hui Lei and Dan Duchamp. An analytical approach to file prefetching. In *USENIX Annual Technical Conference*, Anaheim CA, January 1997.

[10] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

[11] Venkata N. Padmanabhan. Improving world wide web latency. Technical Report CSD-95-875, Computer Science Department, University of California at Berkeley, May 1995.

[12] Kihong Park, Gi Tae Kim, and Mark E. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the Fourth International Conference on Network Protocols (ICNP'96)*, pages 171–180, October 1996.

[13] James Salehi, Zhi-Li Zhang, James Kurose, and Don Towsley. Supporting stored video: Reducing rate variability and ene-to-end resource requirements through optimal smoothing. In *ACM SIGMETRICS*, Philadelphia, PA, May 1996.

[14] A. Smith. Disk cache - miss ratio analysis and design considerations. *ACM Transactions on Computer Systems*, 3(3):161–203, August 1985.

[15] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. In *Proceedings of ACM SIGCOMM '95*, pages 100–113, 1995.