Pushing Politely: Improving Web Responsiveness One Packet at a Time

(Extended Abstract)

Brian D. Davison
Department of Computer Science
Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854-8019 USA
davison@cs.rutgers.edu

Vincenzo Liberatore
UMIACS
University of Maryland
A. V. Williams Building
College Park, MD 20742 USA
vliberatore@acm.org

Abstract

The rapid growth of traffic on the World-Wide Web results in heavier loads on networks and servers and in increased latency experienced while retrieving web documents. This paper presents a framework that exploits idle periods to satisfy future HTTP requests speculatively and opportunistically. Our proposal differs from previous schemes in that speculative dissemination always gives precedence to on-demand traffic, uses ranged requests for improved performance, and can be implemented over a connectionless transport. The protocol uses bounded and little server state even as the workload was increased and it is resistant to erroneous estimates of available bandwidth. Substantial latency improvements are reported over pure on-demand strategies.

1 Introduction

Heavy Internet traffic and bandwidth constraints cause Web users to perceive significant latency. Moreover, data demand at a server is essentially bursty [20] and alternates lulls with spikes of extreme activity. On the negative side, these traffic spikes lead to significant delays and to difficult network design problems [5, 12] and contribute to the perception that the Internet is an unreliable system. On the positive side, traffic lulls give the chance to speculatively execute alternative or background activities. In this paper, we describe a protocol that exploits traffic lulls unobtrusively and speculatively in order to anticipate future HTTP requests. We simulated our architecture on multiple web traces, allowing us to calculate statistics including object and byte hit rates, but our emphasis is on latency experienced by clients (i.e. the amount of time between request and response). Our architecture demonstrated a speed-up of up to 3.35 of client-site latency compared to the traditional (on-demand) strategy.

A client can exploit periods of local inactivity to speculatively load documents that will be likely needed in the near future (*prefetching*) [5, 4, 10, 16, 14]. If prefetching is not aware of network load, it can lead to substantial increases in traffic burstiness and network delays [5]. Prefetching can be effective when restricted to documents that are very likely to be accessed [4, 10] or by using a transport-layer mechanism to limit datagram transmission rate [5]. However, all prefetching schemes thus far can issue prefetching requests even when the server/network is overloaded and, conversely, can refrain from prefetching even when the server/network is idle. In contrast, we explore a wide-area solution that is based on an intuitive *principle of unobtrusiveness*: speculative data dissemination should always give precedence to demand traffic. Our approach, the UDiD (Unobtrusive Distribution of Data) architecture, attempts to minimize the adverse effects of additional traffic. Thus, the contributions of this work include the following key features:

Unobtrusiveness. Our architecture carefully prioritizes resources against speculative data dissemination. In practice, a UDiD server proactively sends documents to selected clients via low-priority datagrams whenever the server is idle. As a result, speculative dissemination always gives precedence to demand traffic.

Use of ranged dissemination and requests. We break documents into approximately equal size ranges, push them independently, and use *ranged requests* as defined by HTTP 1.1 [9, 13] for retrieval of missing ranges.

Minimal server state. Servers need to maintain per-client state, but can bound the amount of state in order to avoid excessive overhead. A fundamental benefit of our approach is that it achieved the performance of prepushing even when servers maintained state for a small number of unique clients in the trace. Furthermore, performance was analyzed when state is kept constant and load was increased by folding several weeks of client activity into one week. In spite of the increased load, the full benefits of data dissemination were achieved when the server maintained state for the same small number of clients as before. A fundamental technique is for the server to allocate client state dynamically to the few most recently active clients.

Connectionless protocol. We describe an implementation over a connectionless transport. In most cases, resulting performance was always within 5% (and typically much closer) of that of an ideal reliable protocol, but a connectionless protocol makes it easier to manage server bandwidth at the application layer and has lower overhead since no actual connection needs to be established, maintained, or closed.

Robustness. Our connectionless protocol depends on estimates of bandwidth available between server and clients. However, protocol performance decreased by less than 15% even when these estimates are $\pm 70\%$ away from actual values.

In this paper, we pose some restrictions and simplifications to the dissemination environment so as to make an initial study of our approach feasible. These include:

- Client state at the server site is simple and can be maintained with little server computation and little communication with the client. State overhead *is* indeed quantified. However, trade-offs between increased state and improved performance are not considered.
- While estimates of available client bandwidth may be available in a complete implementation, we consider only static values of client bandwidth.
- Our architecture assumes the availability of low-priority datagrams. While many QoS schemes are possible, we focus on the textbook technique of priority load shedding [11, 18].
- Document cachability and updates are estimated from data in our traces. While these estimates may vary in accuracy, we believe that, because of the nature of the web sites recorded in our logs and the time frames represented, the actual percentage of uncachable responses is not appreciably larger.
- Our server-client approach is complementary and in no way antithetic to a proxy-client scheme. However, we do not investigate in this paper the interaction between server-based and proxy-based dissemination nor the effects of a proxy cache placed between a UDiD client and server.

The abstract is organized as follows: In section 2, the high-level architecture of the UDiD system is described. In section 3, simulation details are presented and results are reported in section 4. Section 5 concludes the paper. Because of space limitations, many details as well as related work are omitted, and can be found instead in the longer version [7].

2 System Architecture

In this section we provide an overview of our proposed architecture. As mentioned in the introduction, our design decisions are guided by the principle of unobtrusiveness. The implementation-specific details that we simulate are described in section 3. Here, however, we provide the ideals and high-level goals that we envision for any UDiD implementation.

Overview. The UDiD architecture augments the traditional pull mechanisms provided by HTTP by allowing web servers to push[1] documents to web clients. Servers attempt to minimize the side-effects of the push traffic by pushing only when there is sufficient bandwidth available to support it, by pushing data in small portions at a time, and by using a transport mechanism that has lower priority and can be dropped earlier than demand traffic.

Servers. Servers in the UDiD architecture are HTTP servers with additional functionality. When the server receives a request to begin the UDiD protocol, it creates an object (e.g. an 'agent') to represent this particular client. Initialization requests can be piggy-backed on ordinary HTTP requests and include, but are not limited to, client cache size and maximum available client bandwidth. In general, the client may wish to specify a complete policy that includes specific client characteristics and economic constraints so that client preferences are honored.

UDiD servers have some mechanism to select which document to push to a particular client. Typically this is done by maintaining statistics on pull requests for use in future predictions of demand, but the selection method is not particular to UDiD. When the bandwidth to a UDiD client is underutilized, the server selects an

[1] We define a *push* as a transfer of data that is initiated by the sender, and a *pull* as one initiated by the receiver. For example, the HTTP GET request is a pull, while the PUT request is a push.

Parameter	BaseValue	Description
CacheSize	512KB	size of client's extension cache
Transport	connectionless	
Prediction Strategy	Markov	First-order Markov chain [2, 3, 4, 5, 10, 16]
Server Queue	SRTT	Shortest Remaining Transmission Time [6, 15]
Training/Warm-up	trace-specific	time before performance is measured
Bandwidth	trace-specific	bandwidth available at the server site
ClientBw	trace-specific	bandwidth available at the client site
ClientBwError	0%	percent error in estimating client bandwidth
RTT	trace-specific	round trip time
PacketDropProbability	1%	probability a 1KB packet is dropped
AgentMax	64	maximum number of agents allowed by server

Table 1: Principal parameters used in simulations.

appropriate document and begins pushing it to the client. Typically, the server will also maintain an estimate of the expected contents of the UDiD client cache to prevent the waste of bandwidth by sending documents already present in the client cache.

Client agent-objects do not persist indefinitely; they can be replaced by future requests from the client. More importantly, the set of client objects is managed by the UDiD server, which can enforce a policy that is appropriate to the server. The server could, for example, retain the objects corresponding to the clients which have made requests within some time period or the most recently active clients. This allows for flexibility in the implementation so that server overhead for these structures can be minimal.

Clients. A UDiD client is any entity that issues HTTP requests to the server and can receive pushed data from the UDiD server. A cache is necessary to hold pushed documents. For the purposes of analysis, we assume the use of an *extension cache* [8] (i.e. a second level cache in addition to the standard cache for pulled documents), but extension caches may be integrated in a real-world implementation.

UDiD clients can change the characteristics used by the server by sending a new initialization request. Using standard HTTP/1.1 protocols, clients issue ranged requests for missing portions of desired documents, and can validate cached contents using standard consistency mechanisms.

Communication and Transport. The UDiD architecture depends on the traditional mechanisms provided by HTTP for requests and responses for pulled objects. Pushed documents should always give precedence to demand traffic. While packet priorities have often been disregarded in the past, we believe that some form of prioritization will be necessary in the future for differentiated quality of service on the Internet. In the UDiD architecture, we assume the availability of low-priority datagrams[2] in a connection-less protocol for the transport of pushed documents, and the use of the textbook priority load shedding [11, 18]. Thus, at intermediate network stages, such packets will give precedence to on-demand traffic whenever possible. Without such mechanisms, the UDiD architecture can still speculatively disseminate documents (such as by piggy-packing disseminated documents on responses to demand requests), but in a more obtrusive form that may affect demand traffic and overall network performance.

Pushed documents are partitioned into approximately equally-sized ranges (chosen to fit in one datagram) which are sequentially sent to clients in very low priority datagrams. Each datagram contains a document range, along with a header that contains a URI, document size, byte ranges, and an ETAG [9] if available. Thus, each datagram may be of value to the client, independent of the receipt of other datagrams.

Datagrams are pushed at a rate that does not exceed client-specified maximum bandwidth, available server bandwidth, nor dynamic estimates of available bandwidth between client and server that may be obtainable from the monitoring of on-demand HTTP traffic. Available bandwidth can be estimated from TCP [19] with good accuracy for time periods of a few minutes [1] to a few hours [17]. Thus, a client will not necessarily

[2] A low-priority datagram is basically filler traffic for the purpose of this paper. It can be implemented as an IPv6 datagram with priority 0 or 1, or, to some extent, as an `IPTOS_LOWCOST` IPv4 datagram.

Trace	Date	Requests	Hosts	Objects	**AvgBW** (B/s)	**WkBW** (B/s)	**Off Time** (s)	No-Push
EPA	8/29/95	47721	2333	4827	3609.54	6428.0	24	14.5%
NASA	7/3-14/95	981042	48346	5324	20034.5	29291.6	31	0.01%
cs.edu	3/18-23/99	123555	8276	12627	5058.5	8030.7	24	5.8%
ssdc	8-10/97	50464	3706	309	45.8	49.4	16	0.6%
DEC	9/2-20/96	731987	8530	7827	2276.6	3520.7	10	0.4%

Table 2: Characteristics of web traces. Requests is the trace total number of requests, Hosts is the total number of distinct hosts making requests, Objects is the number of distinct web documents in the trace, AvgBW is the average number of bytes requested per second, WkBW is the average number of bytes requested per second from 10AM to 6:30PM (server local time), OffTime is the average time between two requests from the same host (intervals longer than 5 minutes have been disregarded), and No-Push is the percentage of bytes in files that are marked as unsuitable for push over total traffic.

receive data at the rate initially specified by the client due to contention with other clients at the server or contention for network resources.

3 Simulation

To test the UDiD architecture, we have developed a trace-based simulator.[3] It simulates a UDiD server communicating with UDiD clients and their extension caches. Network performance is modeled with packet queuing, connection setup costs (and persistent connections that can sometimes avoid those setup costs), round-trip latency estimates, and packet-loss. However, at present we do not simulate all of TCP, ignoring slow-start effects, realistic loss detection and retransmit delays, and congestion avoidance for HTTP traffic. Consequently, reported performance values are an upper bound on actual TCP performance. Due to space limitation, trace characteristics and simulation parameters are summarized in table 1, and further details are postponed to the full paper [7]. Simulation is based on web server traces whose characteristics are reported in table 2.

Performance Measures. The major performance measure is the average request *latency*. The latency is the time for a client to receive the requested document, and it includes the *RTT*'s, the transmission time, and the time spent in the server queue. If a document is cached, clients can access it with no delay. Results will be reported for latency experienced by requests issued in the peak period (10AM to 6:30PM). We also measured *object hit rate* and *byte hit rate* in client caches.

[3] The simulator source code is publicly available at `http://www.umiacs.umd.edu/users/liberato/software/`.

4 Experimental Results

Latency. Figure 1 gives latency in seconds experienced by a client. For the sake of brevity, latency from two traces are reported and charts from other traces are postponed to the full paper [7]. Both average latencies and their distribution are reported. Latencies are presented for the following environments:

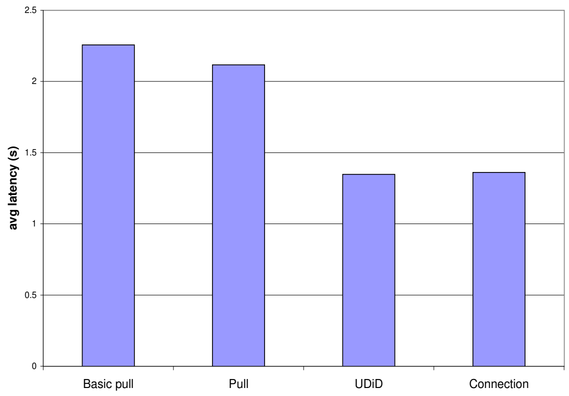
UDiD. The base push configuration with speculative dissemination, ranged requests, and unreliable transport.

Pull. On-demand HTTP protocol. The clients have *CacheSize* amount of memory, use persistent connections and ranged requests, but receive no pushed document.

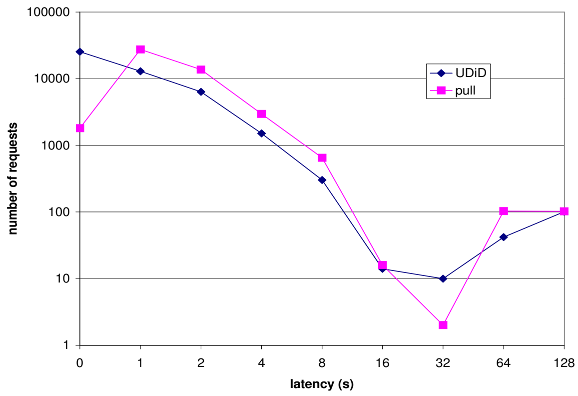
Basic pull. On-demand HTTP protocol with no support for ranged requests, connections are closed after each object transfer (no persistent connections), and clients have no extension cache.

Connection. The UDiD protocol over a reliable transport.

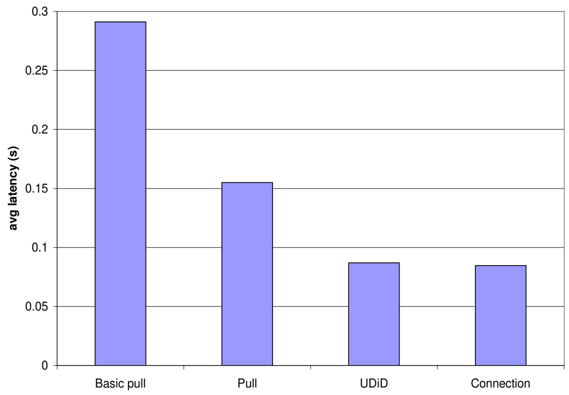
Speculative data dissemination significantly improved client-perceived latency. Speed-ups of the UDiD protocol over basic pull ranged from 1.33 (cs.edu) to 3.35 (DEC). Connection lower bounds are almost always comparable with base latencies, and a significant potential improvement is possible only on the NASA trace. The right column gives the latency distribution in the UDiD and pull configuration. Speculative dissemination increased the number of cache hits (zero latency requests) and reduced the number of low-latency requests. However, the number of higher latency requests is substantially the same in both configurations.



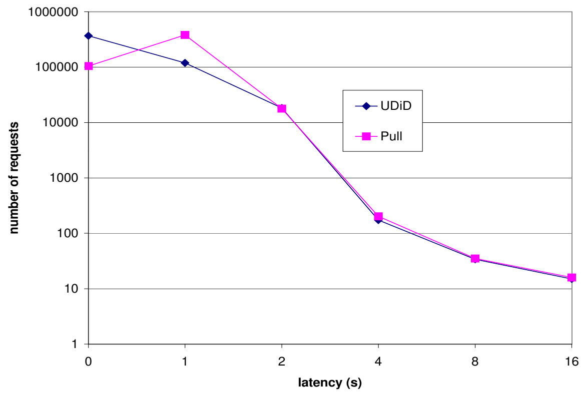
(a) ssdc: average latency



(b) ssdc: latency distribution



(c) DEC: average latency

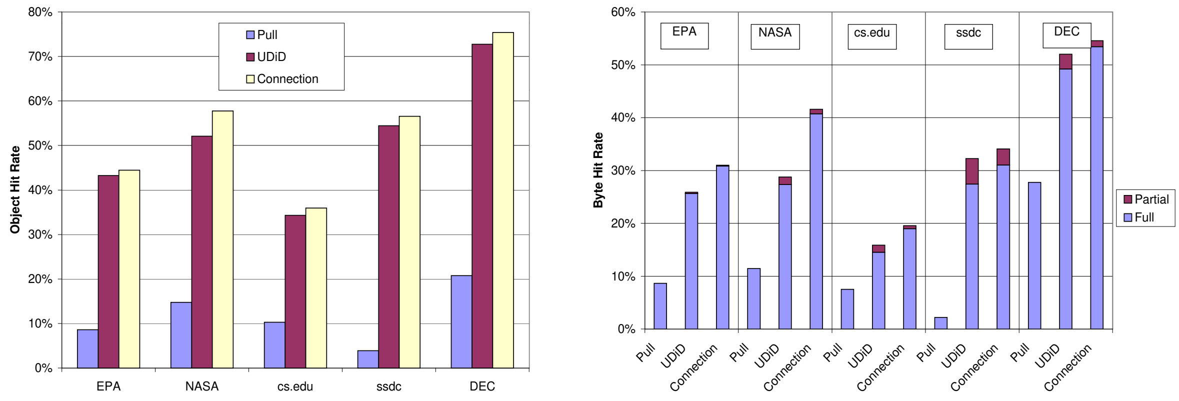


(d) DEC: latency distribution

Figure 1: Latency for the `ssdc` and `DEC` traces. Left column reports average latencies for four set-ups described in the text. Right column shows the distribution of latencies in a log-log scale.

Hit Rates. Although latencies are our fundamental performance metric, we additionally collected hit rate statistics. Figures 2(a) and (b) report object hit rates and byte hit rates for pull and push. Byte hit rates are further divided into byte hit rate for fully cached documents (*full byte hit rate*) and for partially cached documents (*partial byte hit rate*). In most traces, object hit rates were considerably improved by data dissemination (e.g. from 21% to 73% on `DEC`). Furthermore, hit rates were also improved as a result of error recovery used in connections, as a connection protocol avoids missing ranges except in the tail of a document when full document transmission cannot be completed before the client's next request. The connection-based protocol has also higher byte hit rates. Connectionless dissemination has larger values of partial hit rate (up to 5%) and thus benefits from ranged queries.

Server State. The server retains only the *AgentMax* most recently active agents. Conceivably, latency could deteriorate if *AgentMax* remains constant and load increases. To study this effect, we overlapped the three weeks of the `DEC` trace into one week of activity. Such transformation resulted in a 50% increase in the number of distinct clients and in a three-fold increase in the average number of bytes requested per second. The base UDiD configuration uses *AgentMax*=64 agents. The configuration with *AgentMax*=0 agents coincides with the pull set-up. Figure 3 reports latencies for values of *AgentMax* in the range from 0 to 64. In this picture, latencies are normalized to their value for the UDiD base configuration. As few as *AgentMax*=4 agents were needed to retain the speed-up of data dissemination. The *AgentMax*=4 threshold was not significantly affected by increasing server load.



(a) Object Hit Rate (b) Byte Hit Rate

Figure 2: Object hit rates and byte hit rates. Full byte hit rates are relative to documents fully found in the cache and partial byte hit rates to documents partially found in the cache.

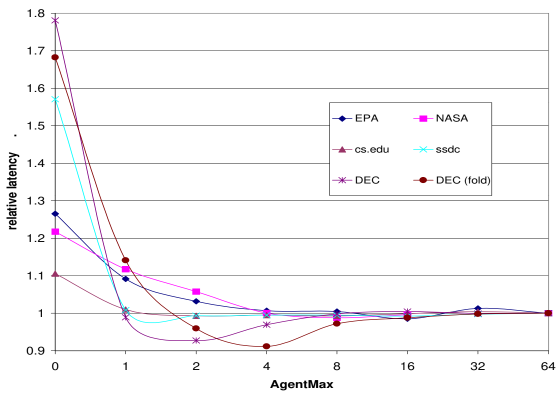


Figure 3: Latency as a function of *AgentMax*. Latencies are normalized to the value for the base case *AgentMax*=64.

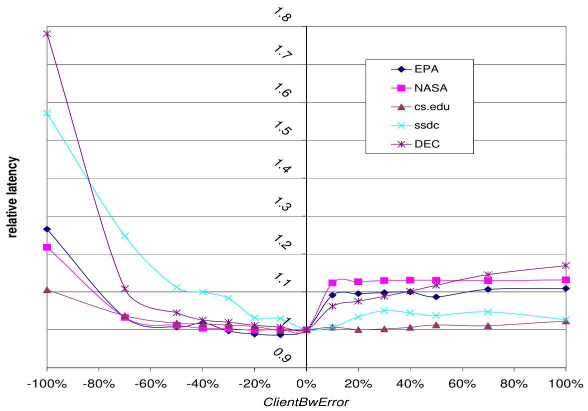


Figure 4: Effects of *ClientBwError* as speed-up of the base configuration (*ClientBwError* = 0) over experiments where *ClientBw* is incorrectly estimated (*ClientBwError* $\neq$ 0).

Imprecise bandwidth estimates. Although available bandwidth can be estimated with reasonable accuracy, servers will not necessarily have correct values. Relative latencies worsened by less than 15% even when *ClientBwError*$= \pm 70\%$ (figure 4).

Bandwidth utilization. In the `EPA` trace, pull uses less than 20% of the available bandwidth, while push operates at 75% of the maximum rate. Therefore, if network nodes have purchased fixed bandwidth, data dissemination more fully exploits available resources. In general, UDiD uses more bandwidth than on-demand protocols while reducing the percentage of bandwidth devoted to pull. Bandwidth utilization is similar for widely varying values of *AgentMax*. The reason is that agents exploit as much unused bandwidth as possible for push operations. Hence, fewer agents can expand to a comparatively large amount of push bandwidth.

Miscellaneous. Increased *PacketDropProbability* progressively worsens performance, but data dissemination shows significant speed-up over on-demand strategies even for $PacketDropProbability = 20\%$. In general, most parameters did not considerably affect push performance.

5 Conclusions

In this paper, the UDiD (Unobtrusive Dissemination of Data) architecture was described. Servers exploit local lulls to proactively push documents in low-priority datagrams to a client when the server predicts that the client will need those documents in the future. Our architecture has been simulated on multiple web traces, where latency speed-ups as high as 3.35 are reported compared to basic pull. The server load is tunable and full speed-up was achieved even when only few clients were allowed. Finally, our architecture can be implemented over a light-weight connectionless protocol and is robust even to gross error in bandwidth estimates.

Acknowledgments

We thank Mike Franklin, Samir Khuller, Thu Nguyen, Mark Nottingham, Craig Wills, and Xiao-Tong Zhuang, for helpful discussions. A number of the logs used here were collected and made available by others: the `EPA` log by Laura Bottomley of Duke University; the `NASA` log by Jim Dumoulin of the NASA Kennedy Space Center; and the `DEC` logs by Digital Equipment Corporation.

References

[1] Hari Balakrishnan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. Analyzing stability in wide-area network performance. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 97)*, Seattle WA, USA, June 1997.

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

[3] Azer Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time in distributed information systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, 1996.

[4] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *Proceedings of SIGCOMM 98*, 1998.

[5] Mark Crovella and Paul Barford. The network effects of prefetching. In *Proceedings of IEEE INFOCOM*, 1998.

[6] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in Web servers. Technical Report BUCS-TR-99-003, Boston University, April 1999.

[7] Brian D. Davison and Vincenzo Liberatore. Pushing politely: Improving web responsiveness one packet at a time. Technical Report DCS-TR-415, Department of Computer Science, Rutgers University, June 2000.

[8] Li Fan, Pei Cao, and Quinn Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1999.

[9] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk, L. Masinter, P. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616; http://ftp.isi.edu/in-notes/rfc2616.txt, June 1999.

[10] Zhimei Jiang and Leonard Kleinrock. Prefetching links on the WWW. In *Proceedings of the IEEE International Conference on Communications*, pages 483–489, 1997.

[11] Srinivasan Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley, Reading, MA, 1997.

[12] Leonard Kleinrock. *Queueing Systems*, volume 1. Wiley, 1975.

[13] Balachander Krishnamurthy, Jeffrey C. Mogul, and David M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. In *Proceedings of the Eighth International World Wide Web Conference*, pages 659–673, Toronto, Canada, May 1999.

[14] Thomas M. Kroeger, Darrell E. Long, and Jeffrey C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 13–22, December 1997.

[15] S. Muthukrishnan, Rajmohan Rajaraman, Anthony Shaheen, and Johannes E. Gehrke. Scheduling to minimize average stretch online. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, October 1999.

[16] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. *Computer Communications Review*, 26(3):22–36, 1996.

[17] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[18] Andrew S. Tanenbaum. *Computer Networks*. Prentice-hall International, Inc., 1996.

[19] Vikram Visweswaraiah and John Heidemann. Improving restart of idle TCP connections. Technical Report 97-661, University of Southern California, November 1997.

[20] Walter Willinger and Vern Paxson. Where Mathematics meets the Internet. *Notices of the AMS*, 45(8):961–970, September 1998.