Pushing Politely: Improving Web Responsiveness One Packet at a Time[*]

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`

June 2000

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 [WP98] 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 [CB98, Kle75] 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*) [CB98, CKR98, JK97, PM96, KLM97]. If prefetching is not aware of network load, it can lead to substantial increases in traffic burstiness and network delays [CB98]. Prefetching can be effective when restricted to documents that are very likely to

[*]An extended abstract of this paper is available as a workshop paper [DL00].

be accessed [CKR98, JK97] or by using a transport-layer mechanism to limit datagram transmission rate [CB98]. 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 [FGM$^+$99, KMK99] 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 impose 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 [Kes97, Tan96].

- 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 paper 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. In section 5, related work is summarized and section 6 concludes the paper.

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.

2.1 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.

2.2 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 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

[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.

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.

2.3 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* [FCJ99] (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.

2.4 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 connectionless protocol for the transport of pushed documents, and the use of the textbook priority load shedding [Kes97, Tan96]. 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 [FGM$^+$99] 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 [VH97] with good accuracy for time periods of a few minutes [BSSK97] to a few hours [Pax99]. Thus, a client will not necessarily receive data at the rate initially specified by the client due to contention with other clients at the server or contention for network resources between the client and server.

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

Table 1: 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.

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. The rest of this section describes further detail of our simulator, the algorithms used (especially for UDiD-specific activities), and the particular settings used for the experimental results described in section 4.

3.1 Traces

Simulation is based on web server traces. Trace characteristics are reported in table 1. The `cs.edu` trace has been logged by the computer science web server at Rutgers University. The `ssdc` trace is the log of Internet requests made to a small software development company that is connected through a 33.6kbps modem link. The `DEC` trace is the log of all requests made to the most popular server from a large proxy trace of all web requests made by DEC.

Most traces report only the HTTP request, the host making the request, the time of the request, the reply code, and the reply size. The table gives the trace collection dates, the total number of requests, the total number of distinct hosts making requests, the number of distinct web objects in the trace, the average number of bytes requested per second (AvgBW), the average number of bytes requested per second from 10AM to 6:30PM server local time (WkBW), and the average time (Off Time) between two requests from the same host (intervals longer than 5 minutes have been disregarded). It also reports the percentage of bytes in documents that are marked as unsuitable for caching and push (see below) over total traffic. The client trace length distribution is similar in the four traces, and, although there are occasionally some very long client traces, almost 70% of the clients issued less than 20 requests per session (a session is defined as a sequence of request that are received no more than 90 seconds apart).

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

In our simulations, a document is marked as suitable for caching and pushing unless either (1) its request URI contains the substring "cgi", or (2) it is invoked through a POST, or (3) it is invoked with a question mark, or (4) it provokes a non-200, non-304 reply. While for some workloads this limited cachability test would significantly under-report true cachability numbers [Dav99], we believe that the actual percentage of uncachable responses is not appreciably larger for these web sites and time frames. Most documents are considered suitable for caching and pushing, as reported in table 1. "No-Push" resources have *not* been deleted from the original trace and *do* participate in the prediction model. Most of our traces do not allow us to determine whether a client has a cache and of which size. Hence, our logs could be the trace of client cache misses. We assume that each client has an extension cache (like that used in [FCJ99]) of moderate size (see below) to keep pushed documents. Likewise, since traces typically do not include last-modification-time, we assume that we have a "hit" if, on a 200 reply, the URL and the number of bytes transferred matched what was in the cache (as in [KS98, WAS$^+$96]).

3.2 Access Pattern Prediction

We use a prediction scheme based on maximum likelihood first-order Markov chains. Similar schemes have been used in much of the previous work on prediction for prefetching [Bes95, Bes96, CKR98, CB98, JK97, PM96]. The server maintains a counter f_{pq} of the number of transitions from document p to document q and increments it whenever a client request for document q was immediately preceded by a request for document p. The counter f_{pq} contains a value that at all steps is proportional to the maximum likelihood estimate of the probability that a client accesses q after p. Servers send document q to a client that had previously requested p when the value of f_{pq} is the highest among all such competing documents. More complex prediction schemes are possible (as in [CKR99, FCJ99]), but are applicable in many situations, not just UDiD, and so are not considered further here.

3.3 Simulation Parameters

Clients. The parameters that describe our simulator are in table 2. Each client has an extension cache of size *CacheSize*. The extension cache keeps pulled and pushed documents in LRU order. Henceforth, such extension cache will be simply called client cache. If a client has kept a version of a web document in its cache longer than the *CacheTimeOut* period, the client considers that version invalid.

Connectivity. Servers are connected through links with a certain available *Bandwidth* and we performed experiments for several bandwidth values with *Bandwidth* > AvgBW. A client is connected to the server by *ClientBw* available bandwidth. The server can communicate with several clients simultaneously. In particular, a server can initiate several push operations to different clients. However, a client will not necessarily receive data at a *ClientBw* rate (as described in section 2.4), so a *ClientBw* represents available bandwidth, but not the actual communication rate.

Although *ClientBw* can be estimated with fairly good accuracy [Pax99], estimates will not necessarily be perfect. Consequently, speculative dissemination may be performed at an erroneous rate when connectionless protocols are used. Bandwidth estimates are assumed to be *ClientBwError* away from the true value *ClientBw*. If *ClientBwError* > 0, bandwidth is overestimated and pushed packets that exceed *ClientBw* are dropped. Conversely, if *ClientBwError* < 0, bandwidth is underestimated, and the server pushes at a rate lower than the maximum possible. Eventually,

Parameter	**BaseValue**	**Description**
CacheSize	512KB	size of client's extension cache
CacheTimeOut	1hr	time before an extension cache entry is considered invalid
Transport	connectionless	
Prediction Strategy	Markov	see section 3.2
EffectiveProbability	5%	minimum probability of a pushed page
Server Queue	SRTT	see end of section 3.3
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
PushTimeOut	90s	time before a server stops pushing documents to idle clients
PushLimit	100%	maximum amount of pushed data per request as a percentage of *CacheSize*

Table 2: Parameters used in simulations.

if *ClientBwError* $= -100\%$, the server does not push at all, even though there would be available bandwidth. The combination of *ClientBw* and *ClientBwError* gives the server an estimate of available bandwidth that can be used with a connectionless transport for open-loop flow control.

Network latency is simulated by the parameter RTT (see table 3) that represents the round trip time of a packet. We model persistent connection with early close [BC99] by waiving an RTT handshake time to those requests issued within fifteen seconds of the time when the previous request was completed for the same client.

The parameter *PacketDropProbability* is the probability that a 1KB packet is dropped by an intermediate network router. In our simulations, pulled documents are fragmented into 1KB packets transparently of the server and of the client, and pushed documents are fragmented into 1KB ranges.

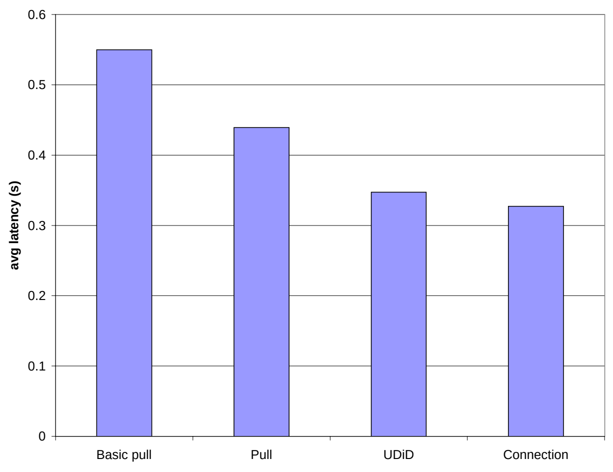
We have investigated two flow control strategies. The first strategy simulates a connectionless protocol and bases transmission rates solely on agent recommendations. The second strategy is to have connections between clients and servers. Both strategies are correct when load is shed by priorities [Kes97], but their performance differs.

Server. The server handles its queue in *shortest-remaining transmission time* (SRTT) order, which has been recently proven to lead to improved response time [CFHB99, MRSG99]. We assume that the only server load is to satisfy HTTP requests.

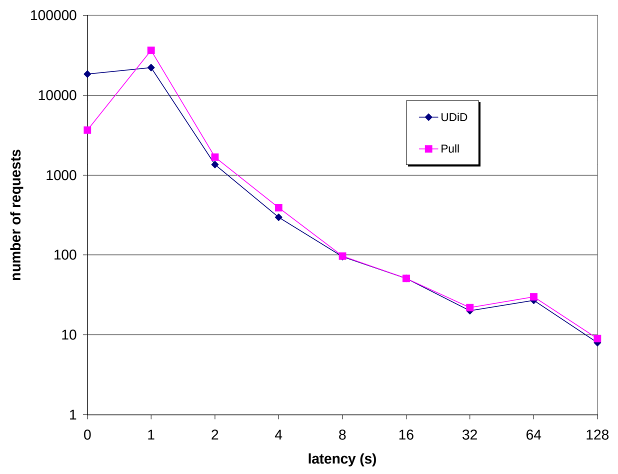
In our simulations, all clients attempt to set up their agents at the server site. However, the server will not allow more than *AgentMax* agents to be simultaneously live. The server creates space for a new agent by killing the agent that has been inactive for the longest time.

3.4 Performance Measures

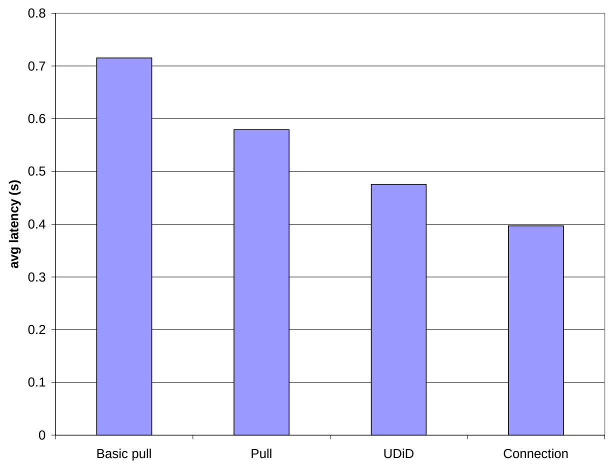
The major performance measure is the average request *latency*. The latency is the time for a client to receive the requested document, and it includes the RTT's, the transmission time, and the time spent in the server queue. If a document is cached, clients can access it with no delay. Results will



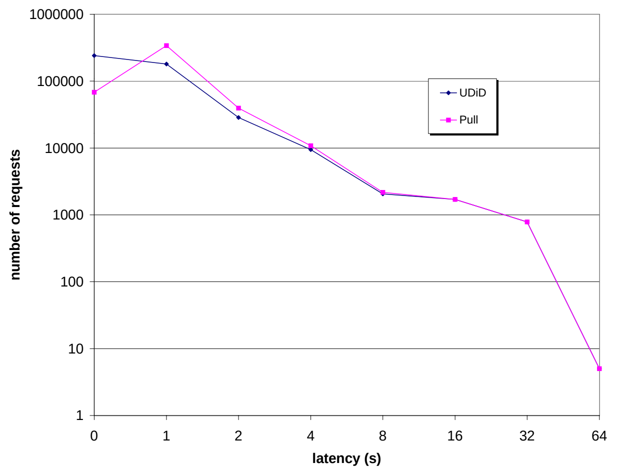
(a) EPA: average latency



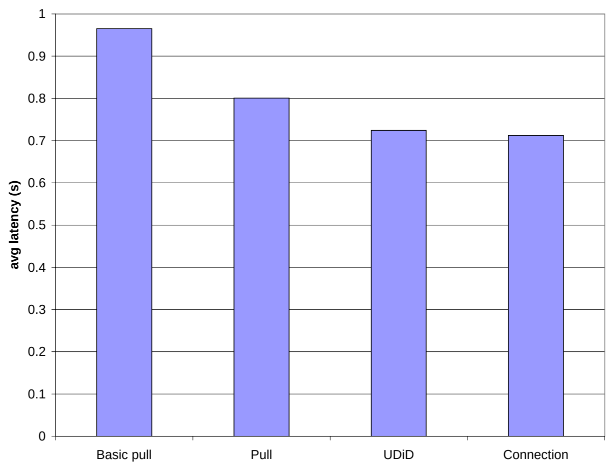
(b) EPA: latency distribution



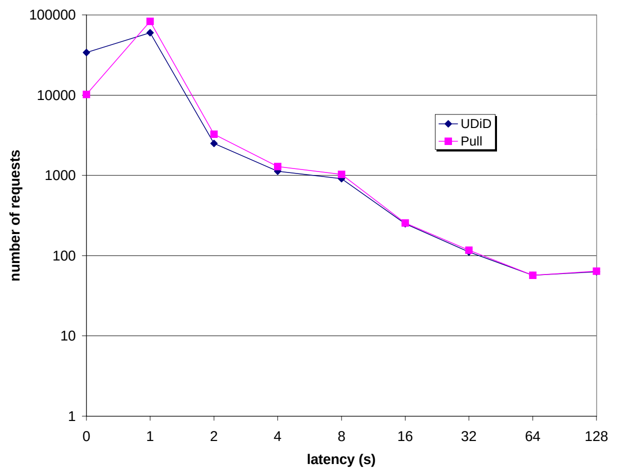
(c) NASA: average latency



(d) NASA: latency distribution

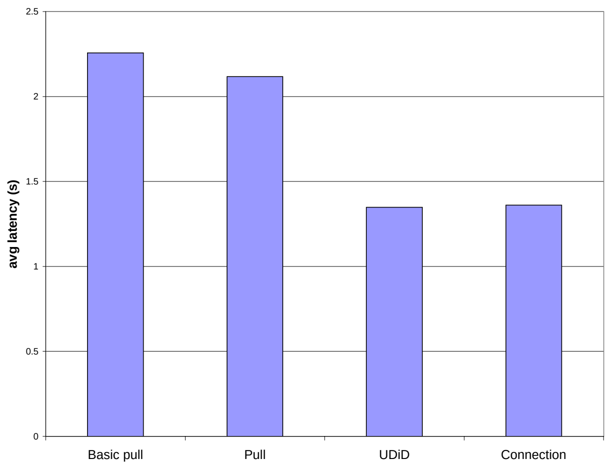


(e) cs.edu: average latency

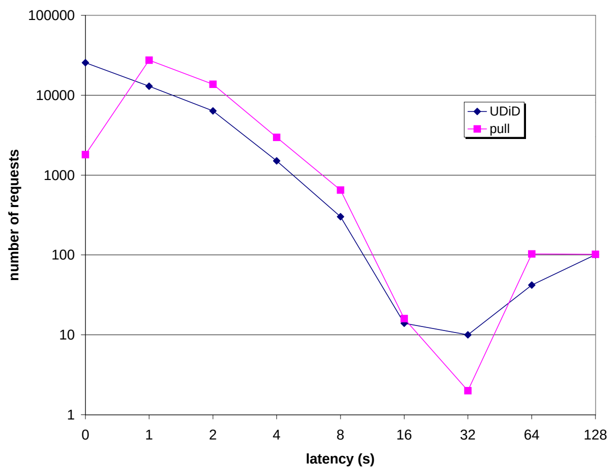


(f) cs.edu: latency distribution

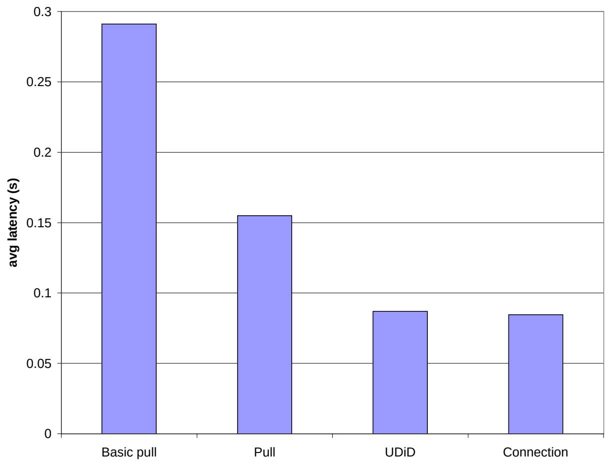
Figure 1: Latency for the `EPA`, `NASA`, and `cs.edu` traces. Left column reports average latencies for four set-ups described in the text. Right column shows the distribution of latencies in a log-log scale.



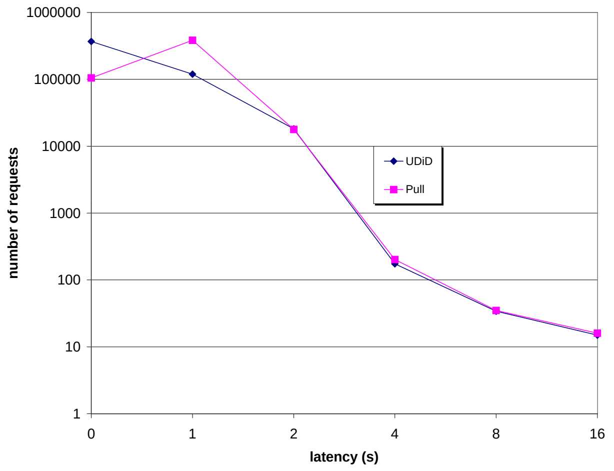
(a) ssdc: average latency



(b) ssdc: latency distribution



(c) DEC: average latency



(d) DEC: latency distribution

Figure 2: 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.

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.

4 Experimental Results

Latency. Figures 1 and 2 give latency in seconds experienced by a client. 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 documents.

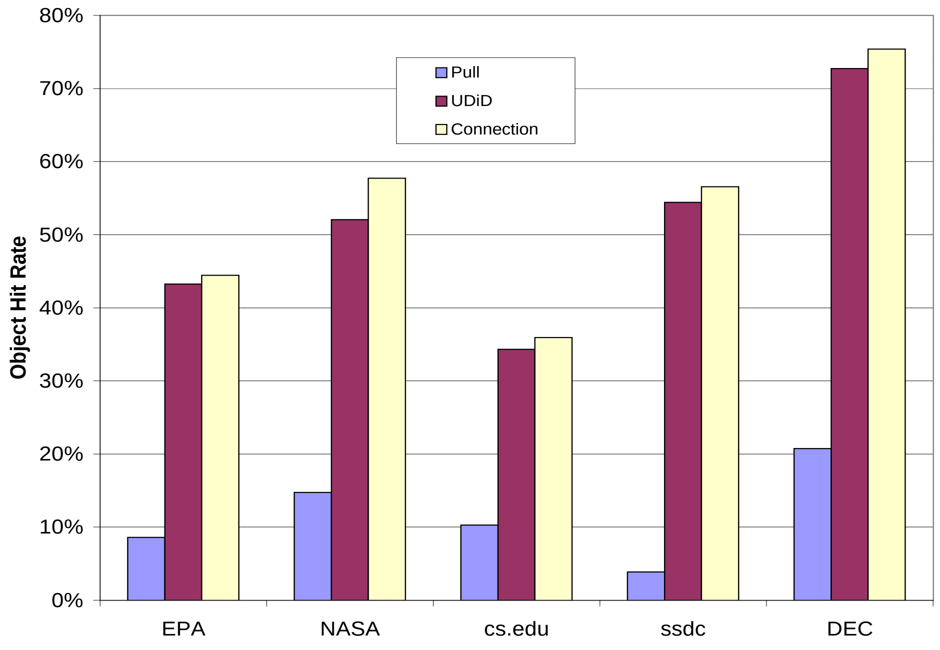


Figure 3: Object hit rates for the three scenarios.

Trace name	*RTT*	*Bandwidth*	*ClientBw*	*Training/Warm-up*
`EPA`	100ms	32KB/s	16KB/s	9 hrs
`NASA`	100ms	128KB/s	32KB/s	1 wk
`cs.edu`	100ms	128KB/s	32KB/s	1 day
`ssdc`	200ms	4KB/s	32KB/s	1 wk
`DEC`	100ms	128KB/s	32KB/s	1 wk

Table 3: Trace specific parameters.

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 the base and in the full pull configuration.

Hit Rates. Although latencies are our fundamental performance metric, we additionally collected hit rate statistics. Figures 3 and 4 report object hit rates and byte hit rates for pull and push.

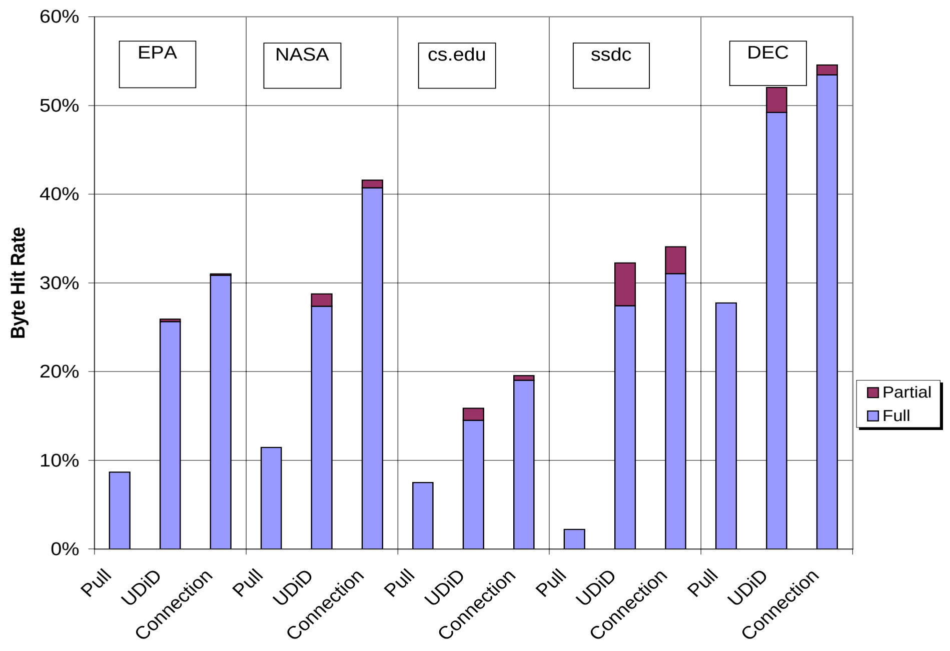


Figure 4: Byte hit rates for the three scenarios. Full byte hit rates are relative to documents fully found in the cache and partial byte hit rates to documents partially found in the cache.

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 5 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. The behavior of these curves can be roughly divided into three regions depending on the size of *AgentMax*.

Medium to large. Experiments did not exhibit significantly different values of latencies for values $AgentMax \geq 8$.

Small. When *AgentMax* is 2 to 4, latency increased for some traces (`NASA` and `EPA`), decreased for others (`DEC`s), remained basically unchanged for the rest. In other words, such *AgentMax*

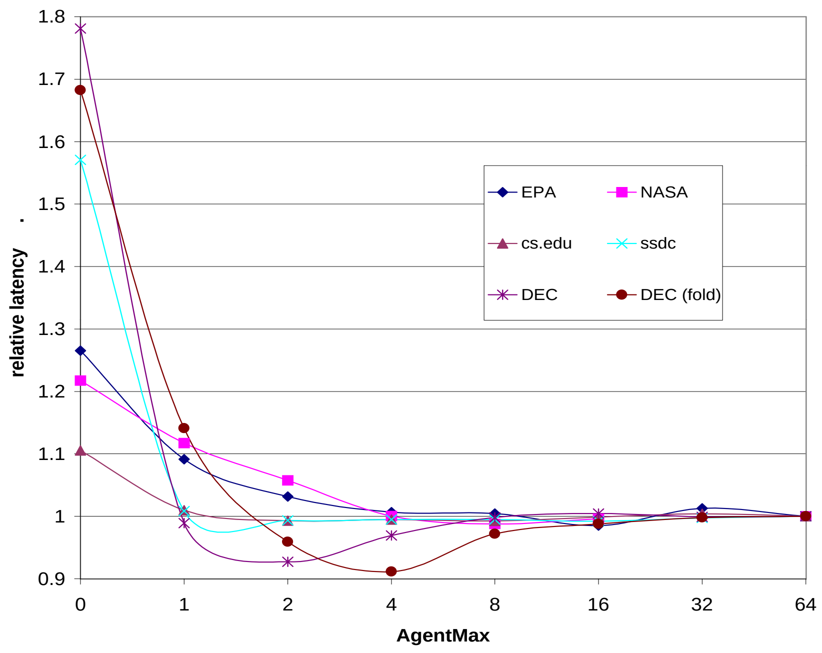


Figure 5: Latency as a function of *AgentMax*. Latencies are normalized to the value for the base case *AgentMax*=64.

values led to increased variance of relative latencies but have small effects on their expected values. We believe that latencies can decrease in this region for two reasons: first, agent contention for server bandwidth diminishes, and, second, more recently active agents are favored over less recently active ones. The folded DEC trace roughly shadows the original trace. The increased number of clients and the increased server load did not require larger server state in order to achieve comparable performance. Thus, our system was perfectly scalable to such increased loads. However, when few agents are allowed, it is critical to identify a policy to retain those agents that have the greatest potential to improve performance. In our experiment, the least recently active strategy was effective in maintaining or improving latencies for small values of *AgentMax*.

Very small. When *AgentMax* is 1 or 0, latencies increased rapidly.

In conclusion, 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.

Imprecise bandwidth estimates. Although available bandwidth can be estimated with reasonable accuracy, servers will not necessarily have correct values. Figure 6 shows relative latencies normalized to the base case as *ClientBwError* varies. Relative latencies worsened by less than 15% even when *ClientBwError*= ±70%.

Bandwidth utilization. Figure 7 gives bandwidth utilization for pull and push. 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, data dissemination uses more bandwidth than on-demand

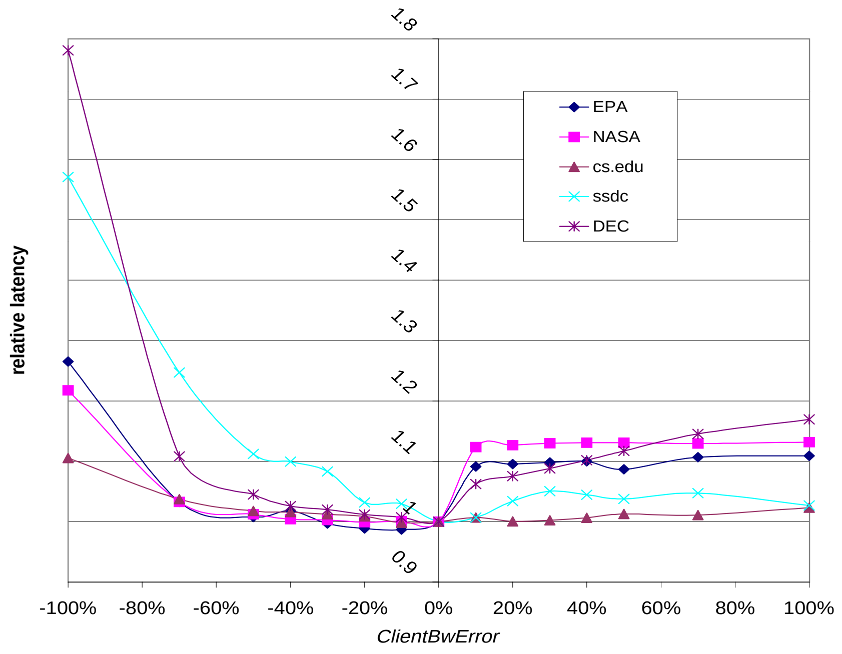


Figure 6: Effects of *ClientBwError* as speed-up of the base configuration ($ClientBwError = 0$) over experiments where *ClientBw* is incorrectly estimated ($ClientBwError \neq 0$).

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. Traffic burstiness can be summarized in one figure by the coefficient of variation [Kle75] of the sum of pull, push, and drop traffic. Figure 8 shows that data dissemination greatly improves the coefficient of variation. Increased *PacketDropProbability* progressively worsens performance, but data dissemination shows significant speed-up over on-demand strategies even for $PacketDropProbability = 20\%$ (figure 9). Figures 10 and 11 shows the speed-up of data dissemination for different values of *Bandwidth* and *ClientBw*. In general, most parameters did not considerably affect push performance. The effects of changes in *CacheTimeOut*, *RTT*, and *PushTimeOut* are small. Bigger client caches translate into smaller latencies but have increasingly smaller returns.

5 Related Work

Much research has been devoted to the problem of caching and replication of web documents, see for example [CI97, TDVK98]. Long-term data dissemination is provided by *mirrors* [Obr94], which differ from UDiD because UDiD operates on a short time-scale, takes advantage of relatively short lulls in the transmission time, and does not necessarily require the presence of an intermediate information repository where data is mirrored. Data dissemination of web documents has been considered in [BC96, Bes95, GS95, MR99], but those techniques differ from UDiD because UDiD

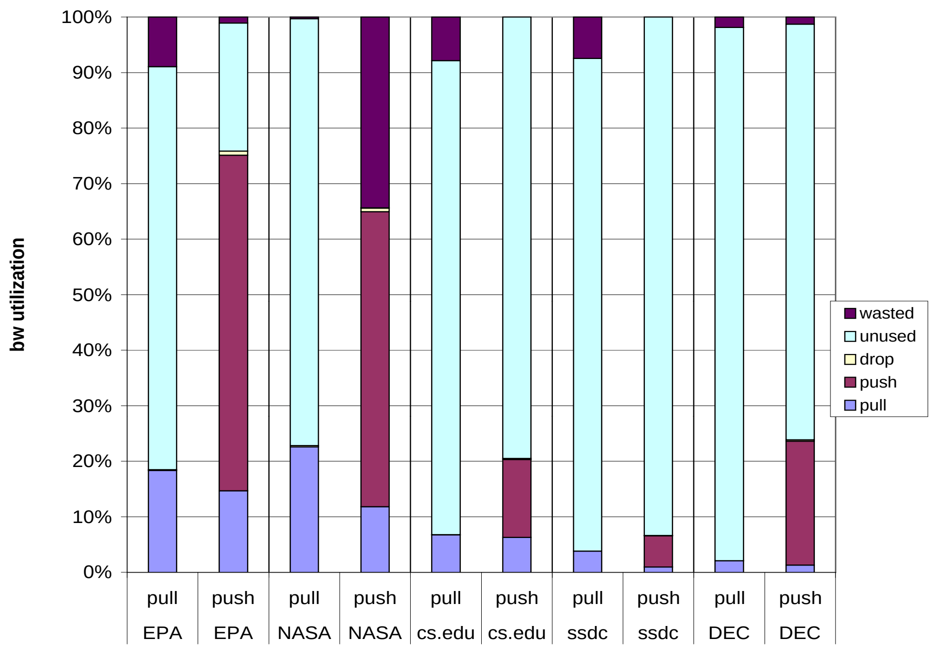


Figure 7: Unused bandwidth is bandwidth that is not utilized while the server queue is empty, and alternately wasted bandwidth while the queue is not empty.

follows peak and lulls in pulled data traffic, and either does not assume the availability of proxies that can mirror data for long periods of time, or can delay push decisions, or does not need topology information. Data dissemination has been considered in the context of broadcast disks [FZ97], hybrid networks [SRB97], and cyclic web multicast [AAF98], where periodic broadcast is used. Recently, a commercial endeavor has received a U.S. patent [RB99] for a mechanism that includes pushing advertisements and other data in a way that does not interrupt demand communications, and has been incorporated in their BackWeb Polite^{TM} Communications product line.[4]

The burstiness of server data traffic is amply documented in the literature [WP98], is substantially different from that in traditional Poisson models, and requires the use of self-similar statistics. Speculative data dissemination is analogous to *prefetching*, except that, in the case of prefetching, data transfers are initiated by clients. Prefetching has been studied in virtually every area of Computer Systems, and a sample recent paper is [CFKL95]. Several web prefetching systems assume that client prefetching is helped by the server, which piggybacks hints on top of regular HTTP replies [CKR98, CB98, JK97, KLM97, PM96]. If prefetching is not aware of network performance, it can aggravate the load on the network and the burstiness of the data traffic [CB98]. Two schemes have been proposed to address the issue. The first scheme allows a client to prefetch a document only if estimated access probabilities are above a certain threshold but, once a prefetch request is issued, it has the same priority as pull requests [CKR98, JK97]. The second scheme is *rate-controlled prefetching* [CB98, EH99], whereby no bound is imposed on probabilities, but the transmission rate of prefetched documents is less than that of requested documents. Finally, server-initiated prefetching has been proposed in [FCJ99] in the context of a proxy servicing dial-up customers. Server initiated prefetching differs from UDiD in several aspects, including reliable modem link,

[4]`http://www.backweb.com/`

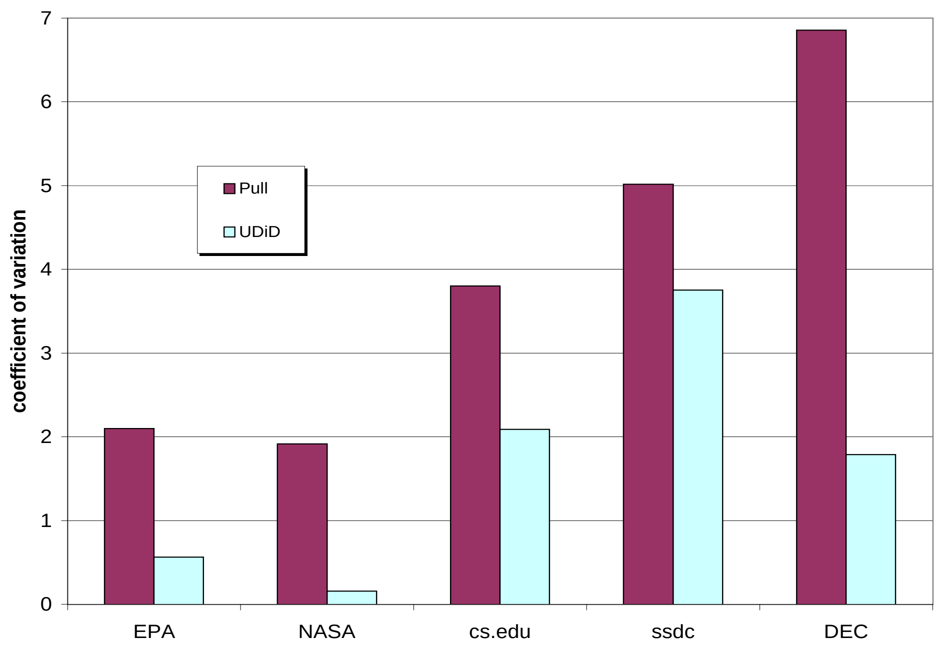


Figure 8: Coefficients of variation for full pull and base configuration.

exact client bandwidth estimates, lack of agent contention, and no use of ranged requests.

While not specifically addressing the task of prefetching, others have noticed the utility of assigning higher drop priority to packets that might otherwise cause congestion (e.g. [PK98] which changes priority within a TCP stream). Open-loop flow control can be done with a number of known techniques, and most notably with the leaky bucket algorithm [Kes97]. UDiD exploits unused bandwidth available in a network. As a related topic, much research has been devoted to exploiting idle CPUs (see [RH98] for a recent paper) and memory [AS99] in distributed systems. Prediction has been often based on a maximum likelihood first-order Markov reference model (MRM) [Bes95, Bes96, CKR98, CB98, JK97, PM96]. Cache consistency and time-out schemes are discussed in [DP96].

6 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.

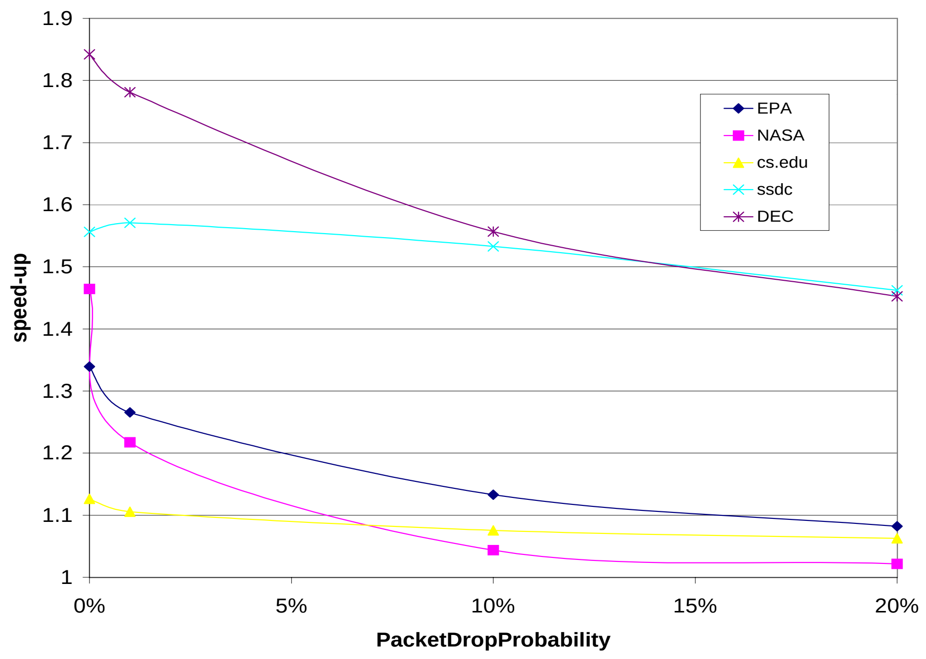


Figure 9: Effects of *PacketDropProbability*. Speed-up is the speed-up of average UDiD latency vs. average latency using pull (not basic pull).

Acknowledgments

We thank Mike Franklin, Samir Khuller, Thu Nguyen, Mark Nottingham, Chau-Wen Tseng, 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

[AAF98] Kevin C. Almeroth, Mostafa H. Ammar, and Zongming Fei. Scalable delivery of Web pages using cyclic best-effort (UDP) multicast. In *Proceedings of IEEE INFOCOM*, 1998.

[AS99] Anurag Acharya and Sanjeev Setia. Availability and utility of idle memory in workstation cluster. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 35–46, 1999.

[BC96] Azer Bestavros and Carlos R. Cunha. Server-initiated document dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19(3), September 1996.

[BC99] Paul Barford and Mark Crovella. A performance evaluation of hypertext transfer protocols. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 188–197, 1999.

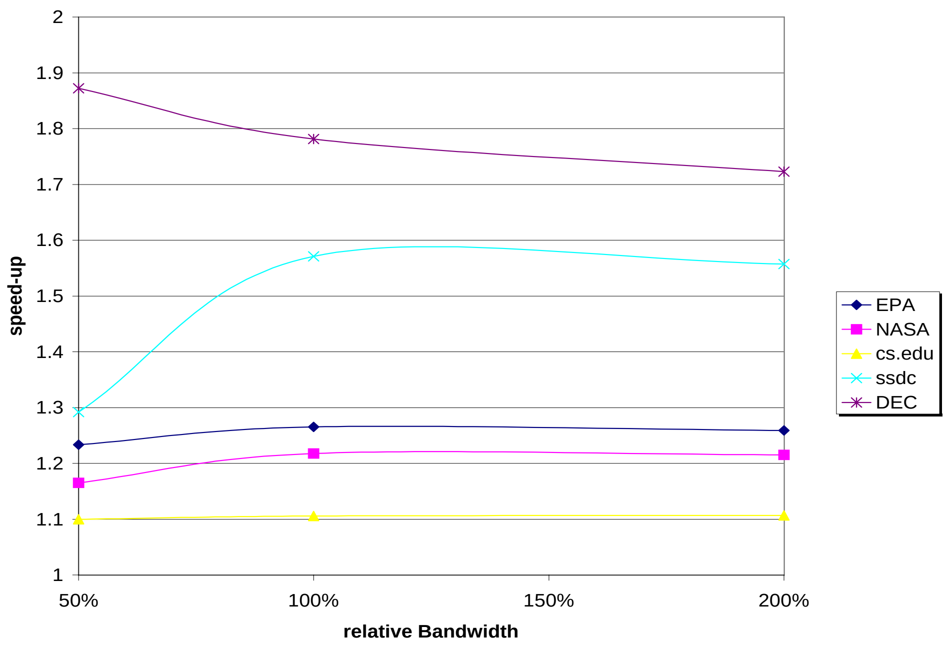


Figure 10: Effects of *Bandwidth*. Relative bandwidth is as percentage of base values. Speed-up is the speed-up of average UDiD latency vs. average latency using pull (not basic pull).

[Bes95] 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.

[Bes96] 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.

[BSSK97] 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.

[CB98] Mark Crovella and Paul Barford. The network effects of prefetching. In *Proceedings of IEEE INFOCOM*, 1998.

[CFHB99] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in Web servers. Technical Report BUCS-TR-99-003, Boston University, April 1999.

[CFKL95] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 188–196, May 1995.

[CI97] Pei Cao and Sandi Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

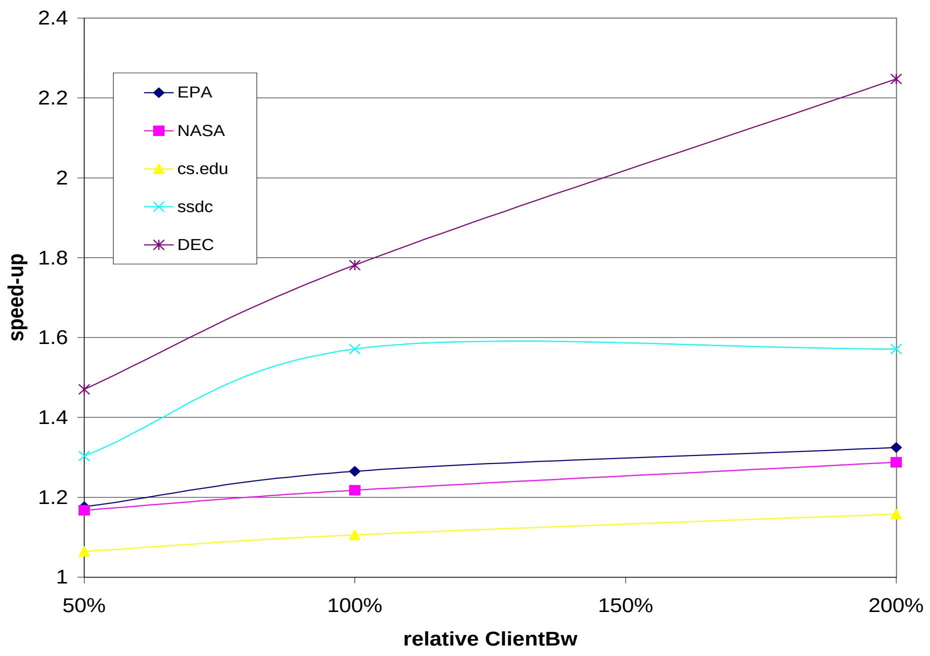


Figure 11: Effects of *ClientBw*. Relative bandwidth is as percentage of base values. Speed-up is the speed-up of average UDiD latency vs. average latency using pull (not basic pull).

[CKR98] 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.

[CKR99] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Efficient algorithms for predicting requests to Web servers. In *Proceedings of IEEE INFOCOM*, 1999.

[Dav99] Brian D. Davison. Web traffic logs: An imperfect resource for evaluation. In *Proceedings of the Ninth Annual Conference of the Internet Society (INET'99)*, June 1999.

[DL00] Brian D. Davison and Vincenzo Liberatore. Pushing politely: Improving web responsiveness one packet at a time (extended abstract). In *Proceedings of the Performance and Architecture of Web Servers (PAWS) Workshop*, June 2000.

[DP96] Adam Dingle and Tomas Partl. Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907–920, May 1996.

[EH99] Lars Eggert and John Heidemann. Application-level differentiated services for Web servers. *World Wide Web*, 3(2):133–142, 1999.

[FCJ99] 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.

[FGM$^+$99] 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.

[FZ97] Michael Franklin and Stanley Zdonik. A framework for scalable dissemination-based systems. In *Proceedings of the International Conference Object Oriented Programming Languages Systems*, pages 94–105, 1997.

[GS95] James Gwertzman and Margo Seltzer. The case for geographical push-caching. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, 1995.

[JK97] Zhimei Jiang and Leonard Kleinrock. Prefetching links on the WWW. In *Proceedings of the IEEE International Conference on Communications*, pages 483–489, 1997.

[Kes97] Srinivasan Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network.* Addison-Wesley, Reading, MA, 1997.

[Kle75] Leonard Kleinrock. *Queueing Systems*, volume 1. Wiley, 1975.

[KLM97] 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.

[KMK99] 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.

[KS98] P. Krishnan and Binay Sugla. Utility of co-operating web proxy caches. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, April 1998.

[MR99] Carlos Maltzahn and Kathy J. Richardson. On bandwidth smoothing. In *Proceedings of the Fourth International Web Caching Workshop*, 1999.

[MRSG99] 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.

[Obr94] Katia Obraczka. *Massively Replicating Services in Wide-Area Internetworks.* PhD thesis, University of Southern California, 1994.

[Pax99] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[PK98] Venkata N. Padmanabhan and Randy H. Katz. TCP fast start: A technique for speeding up web transfers. In *Proceedings of the IEEE Globecom '98 Internet Mini-Conference*, pages 41–46, Sydney, Australia, November 1998.

[PM96] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. *Computer Communications Review*, 26(3):22–36, 1996.

[RB99] Yuval Rakavy and Eli Barkat. Method and apparatus for transmitting and displaying information between a remote network and a local computer. US Patent 5,913,040, BackWeb, June 1999.

[RH98] Kyung Dong Ryu and Jeffrey K. Hollingsworth. Linger longer: Fine-grain cycle stealing for networks of workstations. In *SC'98*, November 1998.

[SRB97] Konstantinos Stathatos, Nick Roussopoulos, and John S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd International Conference on Very Large DataBases*, 1997.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice-hall International, Inc., 1996.

[TDVK98] Renu Tewari, Michael Dahlin, Harrick M. Vin, and Jonathan S. Kay. Beyond hierarchies: Design considerations for distributed caching on the Internet. Technical Report TR98-04, Department of Computer Sciences, University of Texas at Austin, February 1998.

[VH97] Vikram Visweswaraiah and John Heidemann. Improving restart of idle TCP connections. Technical Report 97-661, University of Southern California, November 1997.

[WAS$^+$96] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of ACM SIGCOMM*, pages 293–305, Stanford, CA, 1996. Revised March 1997.

[WP98] Walter Willinger and Vern Paxson. Where Mathematics meets the Internet. *Notices of the AMS*, 45(8):961–970, September 1998.