NCS: Network and Cache Simulator
An Introduction

Brian D. Davison
Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903 USA
davison@cs.rutgers.edu

August 2001

Abstract

NCS (Network and Cache Simulator) is an HTTP trace-driven discrete event simulator of network and caching activity. It is highly parameterized for maximal compatibility with previous caching and network simulations. In granularity, it resides between the high-level caching-only simulators prominant in much Web caching research, and the detailed simulators of networking protocols and traffic. In an effort to capture estimates of user-perceived latency, it simulates simplistic caching and prefetching functionality at various locations in a network comprising of client browsers, an optional intermediate proxy, and Web servers. Caching at the proxy and clients is optional. Additionally, it simulates many aspects of TCP traffic among these entities on a somewhat idealized network. In this report we motivate the development of NCS and describe its features and capabilities. We additionally provide a number of sample experiments showing the simulator's utility in a variety of contexts.

1 Introduction

Simulation is a common approach for algorithmic research, and is widespread in Web caching research (e.g., see [Dav99a]). In general, simulation provides many benefits, including the ability to:

- test scenarios that are dangerous or have not yet occurred in the real world
- predict performance to aid technology design
- predict expected behavior of new protocols and designs without implementation costs or disruption of existing systems
- slow or speed time for effective analysis
- quickly survey a range of potential variations

NCS (Network and Cache Simulator) is an HTTP trace-driven discrete event simulator of network and caching activity. It is highly parameterized for maximal compatibility with previous caching and network simulations. In granularity, it resides between the high-level caching-only simulators prominant in much Web caching research, and the detailed simulators of networking protocols and traffic. In an effort to capture estimates of user-perceived latency, it simulates simplistic caching and prefetching functionality at various locations in a network comprising of client browsers, an optional intermediate proxy, and Web servers. Caching at the proxy and clients is optional. Additionally, it simulates many aspects of TCP traffic among these entities on a somewhat idealized network.

The development goals for NCS included:

- Estimate client-side latencies and bandwidth usages by
  - Capturing intrinsic network effects (e.g., new connection costs, TCP slow start).
  - Modeling real topologies (including browsers, proxies, and servers, with potentially multiple connections and/or persistent connections).
  - Capturing some real-world network effects (including distributions of latencies and bandwidths).
- Provide credibility – be able to compare simulation results to real-world numbers.
- Incorporate optional prefetching techniques for testing and evaluation.

While most Web caching simulators measure hit rates and bandwidth used, few consider the detail needed to believably estimate retrieval latency. In contrast, NCS is specifically designed to estimate the latency experienced by the user, and so includes network simulation in addition to a caching implementation.

This document motivates the design, lists the features, provides an overview of the implementation, and presents some sample experiments to demonstrate the utility of the simulator.

1.1 Motivation

In an effort to improve Web performance from the perspective of both a network administrator and the end user, Web caching has been the focus of much research. (For an introduction and recent surveys see [Dav01b, Wan99, Mog00, BO00]). The evaluation of such research has varied widely among dimensions of measurement metrics, implementation forms (from logic arguments, to event simulations, to full implementation), and workloads used [Dav99a]. Even when restricting one's view to caching oriented simulators (e.g. those used in [WAS$^+$96, CDF$^+$98, FCD$^+$99, FJCL99, ZIRO99, BH00, DL00]), it is apparent that there is a wide range of simulation detail. However, it has also been noted that some details are of particular importance [CDF$^+$98], such as certain TCP slow-start network effects and HTTP connection caching when trying to capture latency estimates.

Caching in the Web is known to reduce network bandwidth, server loads, and user-perceived latency. As overall bandwidth in the net improves and becomes less expensive and more experience is gained in managing loads of popular servers, the final characteristic of latency improvement gains interest among caching researchers. This, in fact, was the initial motivation for the development of NCS. Prefetching is a well-known approach to reduce latency in the memory hierarchy of modern computer architectures, and has been proposed by many as a mechanism for the same in the World Wide Web. However, a significant difficulty in Web prefetching research is the lack of good methods for evaluation. Elsewhere [Dav99b] we have proposed a mechanism for evaluation of fully implemented systems that may employ prefetching, but a simulator is more appropriate for the earlier feedback needed in any research effort.

Therefore, NCS has been designed to estimate the client-perceived latency by simulating the network latencies and the effects of caches present in the network. In addition, it contains optional prediction code to provide various levels of prefetching wherever caching is available.

1.2 Inspiration

Although inspired by the caching simulators described and used in [FCD$^+$99, FJCL99, DL00], the development of NCS has proceeded independently. While this has necessitated significant redevelopment, it has had the side-benefit of being an educational process for the author. An alternative might have been to use the network simulator *ns* [UCB01] and extend it appropriately for prefetching. However, *ns* is also known to have a steep learning curve [BEF$^+$00], and would likely require significantly more computational resources because of the more detailed modeling it performs. By using a less-detailed network model, NCS is able to estimate performance for traces using tens of thousands of hosts much faster than real-time. Additionally, NCS uses trace-based workloads with the associated characteristics of actual data, instead of artificially generated data. In summary, we wanted more detail (and response-time accuracy) than typical caching-only simulators, but faster simulation times for large experiments than the fine-grained network-oriented simulator *ns*, and to be able to incorporate code to optionally estimate the effects of prefetching.

2 Implementation

In general, NCS is designed to be as parametric as possible, so that a large variety of simulated environments can be created. We had particular goals of replicating many of the first order characteristics of existing browsers and proxies (such as caching of data, dns, and connections). The suggested values were often derived empirically (as in [WC98]) or by direct examination of publicly available source code (for Netscape Communicator [Net99] and Squid [Wes01]).

One perceived drawback of a large parameter set is that all parameters must have a value, even when trying to simulate a fairly simple environment. Conversely, this is better viewed as a feature that makes design choices explicit, rather than implicit as in most simulators.

2.1 Features

A quick summary of the features of NCS include its use of a simple form of server and proxy logs as input; its optional support of persistent and/or pipelined connections and more than one connection to the same host. Idle connections can be dropped after timeouts, or when a new connection is needed but the max number of connections have been established. In general it allows an arbitrary number of clients, an optional intermediate proxy, and an optional number of servers. It supports caches at clients and intermediate proxies. It uses parameterized link bandwidths and latencies as input to an emulation of TCP slow-start (with some limitations such as assuming an infinite slowstart threshold). Three versions of TCP slowstart are available: naive slow-start, BSD-style starting with payload of size 2, and a version that uses delayed acks. It can optionally include a simple model for parameterized DNS lookup cost with DNS caching at clients and proxies. Proxies can optionally buffer received files before sending, or can send them as they are being received; in addition, proxies with multiple requests for the same item can either open multiple connections, or wait until the first is received and then serve the second request from cache.

With regard to prefetching, it supports prefetching at clients and/or at proxies. It can model the transmission of hints from server to client at various points in the process. Hints can then be integrated into prediction models. It does not prefetch items that are already being retrieved.

Alternately, NCS does make a number of simplifying assumptions. At the networking level, it always sends (and receives) packets in order, and there is no packet loss. It also ignores TCP byte overheads as well as ACK bandwidth (although ACK timing is used). For some calculations, it also ignores response header bytes and request bandwidth. Finally, it assumes a simple model of a server — in particular that the server (and its subsystems) are not the bottleneck (e.g. through the use of appropriate file and disk management as in [MKPF99]).

2.2 NCS Parameters

The simulator takes just two parameters at the command line. The first is the name of the primary parameter file (described next), and an optional debugging level (increasing values present more detail).

All parameter files have the following form: each parameter is specified, one per line, in the correct order, and is concluded by a special parameter on the last line – the version number of the file format. Since there is just one value used on each line, anything after a whitespace on the same line is ignored.

The simulator reads an entirely numerical trace of requests, one per line, in a log to generate events. The format of the trace file is as follows: the first 6 white-space delimited values are required (client-id, time, request-id, response-code (e.g. 200, 304, etc.), size, and cacheable (binary value)); the server-id is the only item in the first optional group (if not present, all requests go to the same server); the second optional group includes three more values (elapsed-request-time in ms, the last modification time, and the server-request time in ms).

```
#client-id time request-id code size cacheable [server-id [elapsedms lm serverms]]
```

Here is an excerpt of a trace with all parameters:

```
46 846890365.49576 142 200 368 1 51 -1 -1 0.796008
21 846890365.508394 143 200 10980 1 23 -1 833504726 3.505611
47 846890365.519689 144 200 12573 1 52 -1 845000914 19.612891
47 846890365.520312 145 200 6844 1 52 -1 846823803 18.945683
48 846890365.555019 146 200 1340 1 53 -1 845410505 14.995183
```

Thus these requests were made on 1 November 1996, from four different clients to five different servers. All requests were satisfied (response code 200). The first one does not include a last-modified time. None provide an overall elapsed time, but the servers had response times ranging from approximately .8ms to 19.6ms.

2.3 Code Base

While initially prototyped in Perl, NCS is currently implemented in C in approximately 10,000 lines of code (including prediction codes).

Implementation includes the coding of a number of abstract data types, including: 1) doubly-linked circular lists for stack and queue functionality and may optionally be sorted; 2) hash tables; 3) heaps for fast priority queues; and, 4) distributions for random value selection according to various distributions. While not strictly ADTs, separate modules providing caching functionality and other utilities are provided. In the interests of modularity and separate maintenance tracks, the prediction code is likewise separated from the simulation code.

The code for HTTP over TCP, and especially for slow-start effects, is based significantly on the models shown in [THO96, HOT97, KR00]. Heidemann et al [HOT97] claim their model provides guidance for wide-area and low-bandwidth network conditions, but may be somewhat inaccurate when applied to LANs. Therefore, we anticipate similar performance, which has been shown elsewhere [Dav01a]. Fortunately, the area of interest for most caching simulations is not that of LANs, and so we expect that the simulation of network effects over wide-area and low-bandwidth conditions will dominate the occasional inaccuracy in the modelling of LAN conditions.

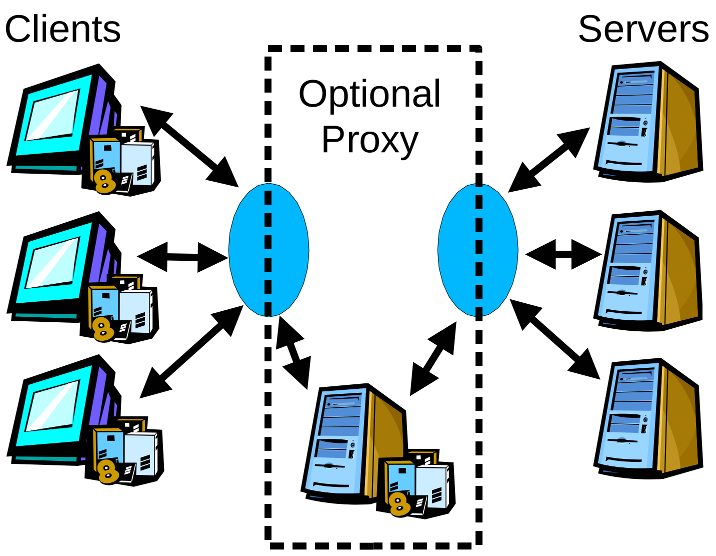

Figure 1: NCS topology. NCS dynamically creates clients and servers from templates as they are found in the trace. It can be configured to use caching at clients and at an optional proxy that can be placed logically near clients or servers.

2.4 Network Topologies and Characteristics

As suggested by the parameters described in 2.2, the simulator is easily configured for a particular set of network topologies. It currently allows for a single set of clients which are connected at a fixed bandwidth and fixed latency. Those clients send their requests either to a single proxy, if used, or to the origin servers. The servers are likewise connected identically with a fixed bandwidth and latency, representing the typical characteristics of all servers. See Figure 1. This particular representation allows for easy management of nodes and parameter files. However, it may be of interest in the future to allow for more complex network topologies and characteristics. For example, it may be useful to have multiple proxies so that different sized sets of clients can be evaluated, or to have some clients use a proxy and others fetch directly. Alternately, it could be desirable to have multiple hierarchically connected proxies to test performance with proxy chains. Each of these scenarios (as well as many others) are supported by the generic simulation code, but have not yet been implemented. Such topological variation would require (among other things) a slightly more complex parameter file format.

3 Sample Experiments

3.1 Demonstration of Proxy Utility

In some situations, proxies can provide performance benefits even without caching any content. Here we will simulate such a situation to demonstrate a sample of what benefits are possible and as an example use of the simulator.

Consider the following scenario: a client is attached to the internet with a bandwidth of 10000B/s, and a network latency of 1s; a server is likewise attached with the same bandwidth and latency. Thus the overall (uncontested) bandwidth between them is still 10000B/s, but the one-way latency is 2s (thus RTT is 4s). We will assume a segment size of 1000, that request and response headers are negligible in size, and that the server takes 0s to calculate a response. If the

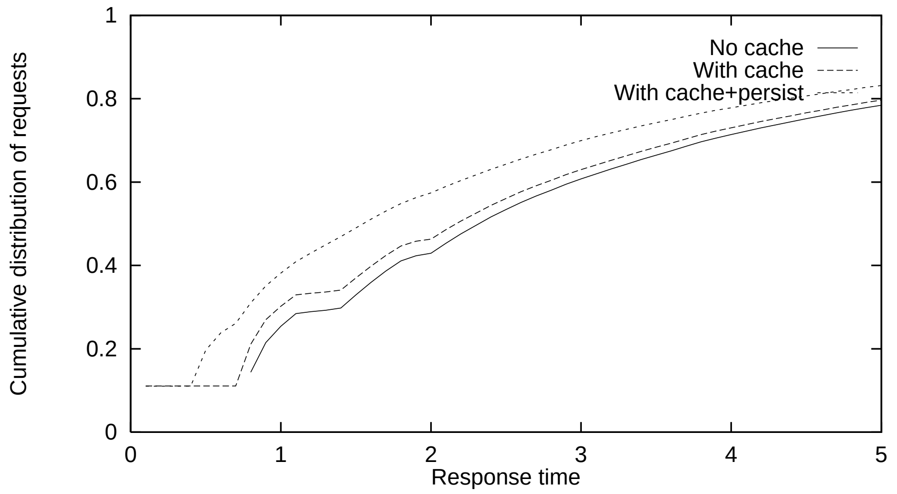

Figure 2: Comparison of client caching on the CDF of response times from the UCB Home-IP request trace.

client makes a new connection to the server, and requests a file that is 10000B long, it will take (according to the simulator) 16.9s to get the entire file.

However, if a proxy is introduced exactly half-way between the two, with infinite bandwidth, no internal latencies and zero-latency connections to the Internet, the response time will improve. In this situation, the bandwidth is still 10000B/s between client and proxy, and between proxy and server, but the delay between client and proxy, and between proxy and server, is now just 1s (RTT of 2s). Even though the RTT from client to server is still conceptually 4s, TCP's slow start is driven by the RTT in the local connection, and effectively forms a pipeline such that activity (or delays) are occuring simultaneously on both sides of the proxy, resulting in a response time of just 13s. For naive slow-start (instead of the more sophisticated one used here) the response times would be 20.6s vs. 14.7s. In either case, the improvement is a significant fraction of the cost of the retrieval.

If we use a more realistic environment in which both client and server are connected via a DSL line (1Mb/s, 10ms latency), we get 229ms response time instead of 196ms with a proxy in the middle. In reality, the introduction of a proxy does introduce some latency for the computation time in the proxy (plus disk, if it is a proxy cache). In certain environments, however, if this latency is small enough, the overall effect can be positive, even without caching. In any case, the simulator provides an environment in which different scenarios can be tested.

3.2 Additional client caching

NCS provides the ability to model Web caching at clients and/or proxies. Here we examine the effect of adding a 1MB cache (in addition to any cache already present) at the client. This new space is called an extension cache [FJCL99]. We further assume that the content of this cache will expire in at most one day from the last request for that object. This simulation uses no persistent connections, nor pipelining, as in the original trace.[1]

To examine the UCB Home-IP trace, we ran variations of the (static) simulator, on the first

[1]Actually, [GB97] says persistent connections were present, but very uncommon.

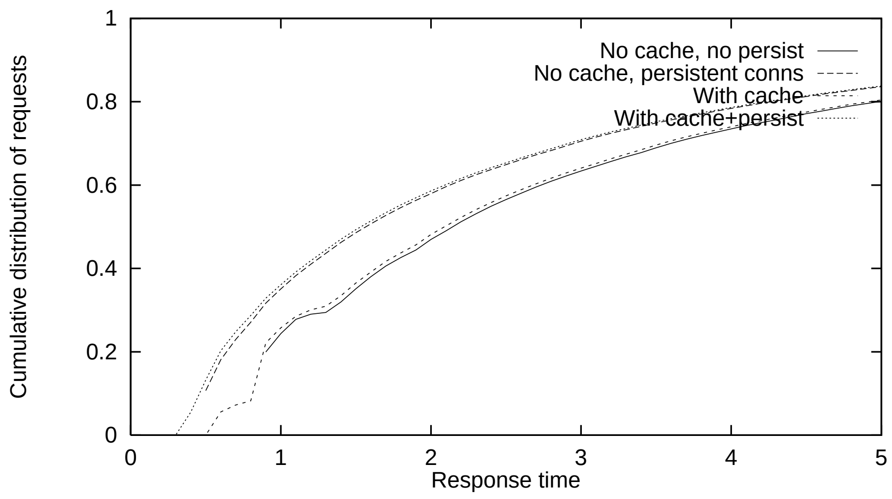

Figure 3: Comparison of proxy caching on the CDF of response times from the UCB Home-IP request trace.

12 days of requests. We used the versions of the traces in which sizes did not include headers, and added a fixed header according to the average response header size from the overall trace (158 bytes). The servers are assumed to be connected with T1 (1.544Mbit) bandwidth, 50ms one-way latencies, and 30ms per-request overhead. Clients are connected at an effective bandwidth of 27.4kbps with 100ms one-way latencies.

From this test, it can be seen that a small amount of additional client caching can improve performance. Just over 11% of requests are served from the client extension caches. The mean response time drops from close to 5.5s to 5.1s, and efficiency improves 17%. Adding persistent connections improves performance even further, dropping the mean to 4.6s (and improving efficiency over uncached by 26%). The median results are visible in Figure 2, in which the uncached median stands at 2.4s, caching improves that to 2.2s, and persistent connections brings the median down to 1.6s.

3.3 The addition of a caching proxy

In addition, or in lieu of client caching, we can examine the effect of caching at the proxy level. In these experiments we assume the use of a single proxy cache that provides all content to the clients in the trace. While the proxy will cache objects for all clients (assuming a relatively small, fixed 512MB cache size), the clients will not use an extension cache.

In this test, the clients and servers are configured identically to those in section 3.2, with the exception of not caching at the client. Here we add instead a proxy, with 45Mbit connectivity upstream and downstream. The modeled proxy also adds 2ms latency in both directions, and a per-request cpu overhead of 10ms.

In this system, a scenario with a non-caching proxy without persistent connections achieves a mean client response time of 5.10s. A caching proxy saves little more than 1.4% in response time, but does provide a reasonable hit rate (19.7%) and reduced network bandwidth usage (a savings of 9%). Likewise, mean per-request throughput improves by 5.3%. These minor improvements are expected in such a scenario, as the predominant bottleneck (the poor client connectivity) has not been addressed in this scenario (either by caching at the client, or by persistent connections

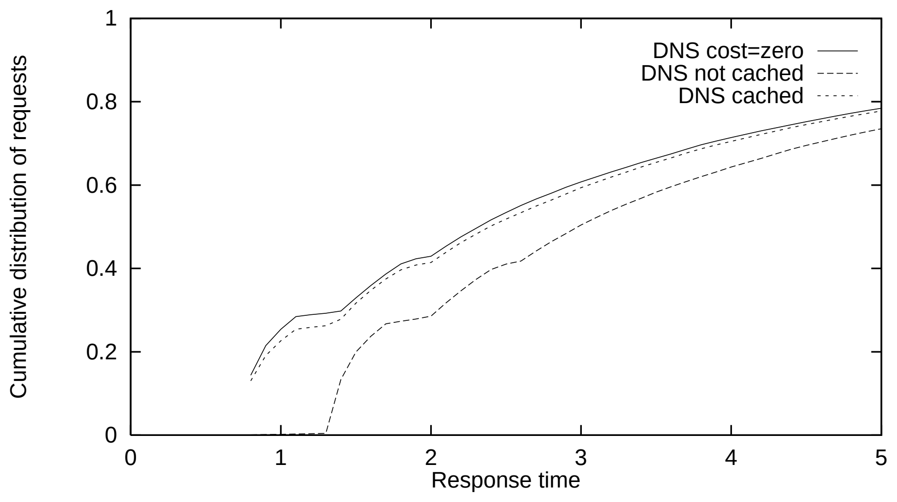

Figure 4: Comparison of DNS caching at the client on the CDF of response times from the UCB Home-IP request trace.

between client and proxy).

Support for persistent connections provides a good boost in response time and throughput. A non-caching proxy with persistent connections shaves 562ms from the average response time, and .6s off of the median (a savings of 10.7% and 27.3%, respectively). Add caching, and the mean response time dropes to 4.65s (11.5% response-time savings over the non-caching, non-persistent scenario). See Figure 3 to compare all four configurations.

3.4 Modelling DNS Effects

NCS offers a simple mechanism to model DNS effects. Each node can provide a fixed cache of DNS lookups for a parameterized amount of time. While the cost of DNS lookups is currently fixed (or possibly selected randomly from some distribution), a more complex simulation (not currently implemented) could be possible, utilizing organization-wide DNS caches to supplement the client cache.

3.4.1 Client caching of DNS entries

Here we model characteristics of the Netscape 4 browser (10 entries in the DNS cache, expiration after 900s), and assume a fixed cost of 600ms to retrieve a new entry for the UCB Home-IP trace. We are not simulating any additional web caching so that only the caching of DNS responses contributes to changes in performance. Otherwise, this simulation is identical to the caching client scenario in section 3.2. We can show improvements in all measures: mean per-request latency decreased by 11% (from 6.25s to 5.54s); median per-request latency decreased by 20% (from 3.0s to 2.4s); and mean per-request throughput also improved by 19%. See Figure 4.

3.4.2 Proxy caching of DNS entries

DNS caching can also be performed by shared proxies, and is in fact a benefit of using a (non-transparent) proxy. When a client is configured to use a parent proxy, the only DNS lookup that is necessary is the IP address of the proxy, since all requests will be sent to it. The proxy will have to

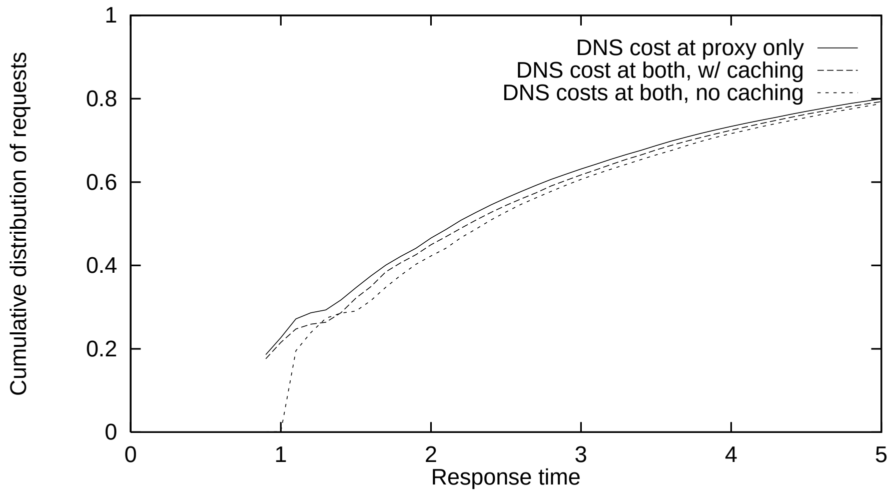

Figure 5: Effect of DNS caching at a proxy on the CDF of response times from the UCB Home-IP request trace.

do the DNS lookup, and can operate a larger and better-managed DNS cache than most browsers. In the case in which requests are 'transparently' intercepted by a proxy, both the client and the proxy have to ask for DNS resolution. In Figure 5, we again assume modem-based clients using the UCB trace with a cost of 600ms for client DNS access, and a cost of 200ms for proxy DNS access, and show a small but measurable improvement in typical access latency.

3.5 Incorporating Prefetching

NCS incorporates the ability to perform prefetching — that is, to speculatively issue retrieval requests in advance of an actual request in the trace. Prefetching can be performed by clients or by proxies. The model to decide what to prefetch can be built by the client, proxy, or server. In the case of a server or proxy model, the predictions can be included as hints that are sent to the downstream prefetcher that can use them or integrate them with its own predictions. However, the model built is reflective of the requests seen by the system building the model — i.e., a server model sending hints will be built from the requests (prefetching or demand) generated by its clients.

To demonstrate this feature, we will use a Web server trace containing months of requests to the website of a small software development company and configure the simulated network topology to consist of non-caching DSL clients (128kbps bandwidth, 80ms round trip delay), a prefetching proxy with 5MB of cache placed at the ISP to which the Web server is connected via 33.6kbps modem. The proxy cache is additionally configured such that it will only retain objects for an hour at most. The prediction module makes only one prediction useing the largest Markov model with sufficient support after each request. The maximum order used was three; the minimum order was one. Ten requests of an object were needed before it could be predicted.

If the proxy is configured to passively cache only, the simulation estimates a mean end-user latency of 3.437s and median of .790s. It had an object hit rate of 51%, and used only 72.2% of the demand bandwidth when communicating with the origin server. Alternately, the prefetching version described above reduced the median client delay to .70s and the mean to 3.280s, at the cost of another 4.7% in demand bandwidth.

Given a suitably augmented trace file, we can simulate a single-step omniscient predictor. Thus, for comparison, if we had an omniscient predictor that would give the next request made by a client, the same prefetching proxy would have achieved a median client delay of .670s and mean of 3.106s.

4 Discussion

4.1 Related Work

Many researchers use simulation to estimate Web caching performance. Often they measure object and byte hit rates and ignore response time latencies. Response time improvement, however, is a common justification for the use of Web caching, and is arguably the initial *raison d'être* for content delivery networks such as Akamai [Aka01]. As mentioned above, however, there are some simulators that estimate response times. We describe a few of them here.

PROXIM [CDF$^+$98, FCD$^+$99] is a caching and network effects simulator developed by researchers at AT&T Labs. Like NCS, PROXIM accounts for TCP slow start and does not consider packet losses and their resulting effects. It does, however, consider cancelled requests and their effect on bandwidth usage, unlike NCS or the other simulators mentioned here. PROXIM does not include prefetching. The traces used by Feldmann et al were captured by a snooping proxy and provide extraordinary timing details (including timestamps of TCP as well as HTTP events), allowing PROXIM to use RTT estimates from measurements of SYN to SYN-ACK and HTTP request and response timestamps on a per-connection basis.

Like NCS, the simulator described in Fan et al [FJCL99] uses the timing information in a portion (12 days) of the UCB trace [Gri97] to estimate latency seen by each modem client. However, it also uses a much simpler model of latency that

- estimates client-seen latency to be the sum of 1) time between seeing the request and the first byte of the response, and 2) the time to transfer the response over the modem link.
- apparently ignores connection setup costs.
- apparently ignores TCP slow start effects.
- groups overlapping responses together.

However, Fan et al claim that by grouping overlapping responses together they are able to measure the time spent by the user waiting for the whole document. This simulator does incorporate some prefetching — it uses one kind of prediction by partial match (PPM) for proxy initiated prefetching (in which the only objects prefetchable are those cached at the proxy, resulting in no new external bw usage).

Kroeger *et al.* [KLM97] examine the limits of latency reduction from caching and prefetching. They use proxy traces from a large company and find that 23% of the latency experienced by a user is internal, 20% is external but cannot be fixed by caching or prefetching, and the remaining 57% correspond to external latencies that can be improved. Prediction is performed by an optimal predictor, with some limitations on when prediction is permitted. The latencies used are not calculated *per se*, but are extracted directly from the logs.

The *ns* simulator [UCB01] is likely the best-known networking simulator, but is not typically used for caching measurements, possibly because of slow simulation speeds. It uses detailed models of networking protocols to calculate performance metrics (see Breslau *et al.* [BEF$^+$00] for an overview).

4.2 Ongoing enhancement efforts

Much of NCS is simplistic, and could be improved (both in fidelity and in simulation efficiency) with additional time and effort. A number of areas of improvement are underway, including:

- Range-based requests and caching. Modern traces include partial content responses, implying that range-requests were used.
- Better models for packet loss. The simulator does include a simple uniform probabilistic model of loss, but one based on congestion, along with recovery mechanisms would be more appropriate.
- A better model for consistency. The current simulator does not sufficiently model changes at the source, and as a result it does not generate simulated GET If-Modified-Since requests.
- Model terminated requests. Some traces provide information that specifies if a response was terminated early (as a result of user cancellation via the stop button, or by clicking ahead before the document fully loaded). Similarly, it may be important that prefetch requests using GET be terminable when demand requests are present.
- Prefetching other than most-likely next request(s). Large objects need extra time to be retrieved, and so prefetching further in advance (i.e. more than one step ahead) may be needed.

Such improvements would increase simulation fidelity. Their impact on simulation results, however, is unknown.

4.3 Conclusion

In this report we have motivated the development of NCS and described its features and capabilities. We additionally provided a number of sample experiments showing the simulator's utility in a variety of contexts.

Acknowledgments

This work was supported in part by the National Science Foundation under grant NSF ANI 9903052.

References

[Aka01] Akamai Technologies, Inc. Akamai home page. `http://www.akamai.com/`, 2001.

[BEF^{+}00] Lee Breslau, Deborah Estron, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[BH00] Adam Belloum and Bob Hertzberger. Maintaining Web cache coherency. *Information Research*, 6(1), October 2000.

[BO00] Greg Barish and Katia Obraczka. World Wide Web Caching: Trends and Techniques. *IEEE Communications Magazine Internet Technology Series*, 38(5):178–184, May 2000.

[CDF$^+$98] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web Proxy Caching: The Devil is in the Details. *Performance Evaluation Review*, 26(3):11–15, December 1998. Proceedings of the Workshop on Internet Server Performance.

[Dav99a] Brian D. Davison. A Survey of Proxy Cache Evaluation Techniques. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, pages 67–77, San Diego, CA, March 1999.

[Dav99b] Brian D. Davison. Simultaneous Proxy Evaluation. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, pages 170–178, San Diego, CA, March 1999.

[Dav01a] Brian D. Davison. HTTP simulator validation using real measurements: A case study. In *Ninth International Symposium on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems (MASCOTS 2001)*, Cincinatti, OH, August 2001.

[Dav01b] Brian D. Davison. A Web caching primer. *IEEE Internet Computing*, 5(4):38–45, July/August 2001.

[DL00] Brian D. Davison and Vincenzo Liberatore. Pushing Politely: Improving Web Responsiveness One Packet at a Time (Extended Abstract). *Performance Evaluation Review*, 28(2):43–49, September 2000. Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, June 2000.

[FCD$^+$99] Anja Feldmann, Ramón Cáceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. In *Proceedings of IEEE INFOCOM*, pages 106–116, New York, March 1999.

[FJCL99] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*, Atlanta, GA, May 1999.

[GB97] Steven D. Gribble and Eric A. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.

[Gri97] Steven D. Gribble. UC Berkely home IP HTTP traces. Online: `http://www.acm.org/sigcomm/ITA/`, July 1997.

[HOT97] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking*, 5(5):616–630, October 1997.

[KLM97] Thomas M. Kroeger, Darrell D. 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 (USITS '97)*, December 1997.

[KR00] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison-Wesley, 2000.

[MKPF99] Evangelos P. Markatos, Manolis G.H. Katevenis, Dionisis Pnevmatikatos, and Michail Flouris. Secondary Storage Management for Web Proxies. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS99)*, pages 93–104, 1999.

[Mog00] Jeffrey C. Mogul. Squeezing more bits out of HTTP caches. *IEEE Network*, 14(3):6–14, May/June 2000.

[Net99] Netscape, Inc. Mozilla source code. Available at `http://lxr.mozilla.org/classic/`, 1999.

[THO96] Joe Touch, John Heidemann, and Katia Obraczka. Analysis of HTTP performance. Available at `http://www.isi.edu/lsam/publications/http-perf/`, August 1996.

[UCB01] UCB/LBNL/VINT. Network simulator ns. `http://www.isi.edu/nsnam/ns/`, 2001.

[Wan99] Jia Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[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.

[WC98] Zhe Wang and Pei Cao. Persistent connection behavior of popular browsers. `http://www.cs.wisc.edu/~cao/papers/persistent-connection.html`, 1998.

[Wes01] Duane Wessels. Squid Web proxy cache. Available at `http://www.squid-cache.org/`, 2001.

[ZIRO99] Junbiao Zhang, Rauf Izmailov, Daniel Reininger, and Maximilian Ott. WebCASE: A simulation environment for web caching study. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, San Diego, CA, March 1999.