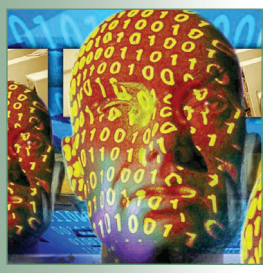

A Web Caching Primer

Now a significant part of the Web's infrastructure, Web resource caching can reduce network latencies and bandwidth demands transparently.

Brian D. Davison
Rutgers, The State University of New Jersey

When the Web was new, a single entity could (and did) list and index all of the Web pages available, and searching was just an application of the Unix egrep command over an index of 110,000 documents.[1] Today, even though the larger search engines index billions of documents, any one engine is likely to see only a fraction of the content available.[2] Moreover, with the widespread commercialization of the Web, exceeding the "eight-second rule" for downloading a Web page can mean a significant loss of revenue as many users will move on to a new site if they are unsatisfied with the performance of the current one.[3] Finally, as increased Web use necessitates larger and more expensive connections to the Internet, concern for efficient use of those connections similarly increases.

This article provides a primer on Web resource caching, one technology used to make the Web scalable. Web caching can reduce bandwidth usage, decrease user-perceived latencies, and reduce Web server loads transparently. As a result, caching has become a significant part of the Web's infrastructure. Caching has even spawned a new industry: content delivery networks, which are also growing at a fantastic rate.

Readers familiar with relatively advanced Web caching topics such as the Internet Cache Protocol (ICP),[4] invalidation, and interception proxies are not likely to learn much here. Instead, this article is designed for the general audience of Web users. Rather than a how-to guide to caching technology deployment, it is a high-level argument for the value of Web caching to content consumers and producers. The article defines caching, explains how it applies to the Web, and describes when and why it is useful. Though I provide several topical references, readers interested in survey papers should look elsewhere (see the sidebar, "Web Caching Resources" on page 43).

Caching in Memory Systems

Memory architectures use caching to improve computer performance.[5] Because central processing units operate at very high speeds while memory systems operate at a slower rate, CPU designers provide one or more levels of cache — a

 1089-7801/01/$10.00 ©2001 IEEE

small amount of memory that operates at, or close to, the speed of the CPU. When the CPU finds the information it needs in the cache, a *hit,* it doesn't have to slow down. When it fails to find the requested object in the cache, a *miss*, it must fetch the object directly and incur the associated performance cost.

Typically, when a cache miss occurs, the CPU places the fetched object in the cache, assuming *temporal locality* — that a recently requested object is more likely than others to be requested in the future. Memory systems also typically retrieve multiple consecutive memory addresses and place them in the cache in a single operation, assuming *spatial locality* — that nearby objects are more likely to be requested during a certain time span.

At some point the cache will become full and the system will use a *replacement algorithm* to make room for new objects, for example, first-in/first-out (FIFO), least recently used (LRU), or least frequently used (LFU). The goal is to optimize cache performance (for example, to maximize the likelihood of a cache hit for typical memory architectures).

Mechanics of a Web Request

In its simplest form, the Web is a set of servers and clients (such as Web browsers, or any other software used to make a request of a Web server). To retrieve a particular Web resource, the client attempts to communicate over the Internet to the *origin Web server*, as depicted in Figure 1. To connect to the server, the client needs the host's numerical identifier. It queries the domain name system (DNS) to translate the hostname (for example, www.web-caching.com) to its Internet Protocol (IP) address (209.182.1.122), with which it can establish a connection to the server and request the content. Once the Web server has received and examined the client's request, it can generate and transmit the response. As Figure 2 shows, each step in this process takes time.

The hypertext transfer protocol (HTTP) specifies the interaction among Web clients, servers, and intermediaries. Requests and responses are encoded as headers that precede optional bodies containing content. Figure 3 (next page) shows one set of request and response headers. The first request header shows the method used (GET), the resource requested ("/"), and the version of HTTP supported (1.1). Another commonly used method is POST, which allows clients to send content with a request (for instance, to carry variables from an

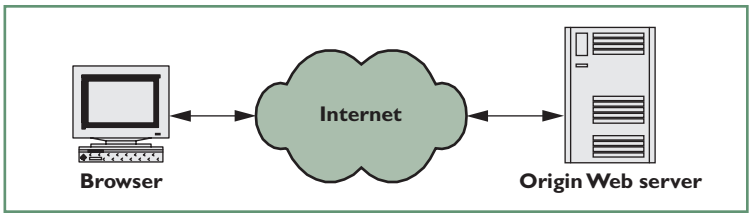

Figure 1. A simplistic view of the Web. At its most basic, the Web consists of a set of servers and clients and the infrastructure that connects them.

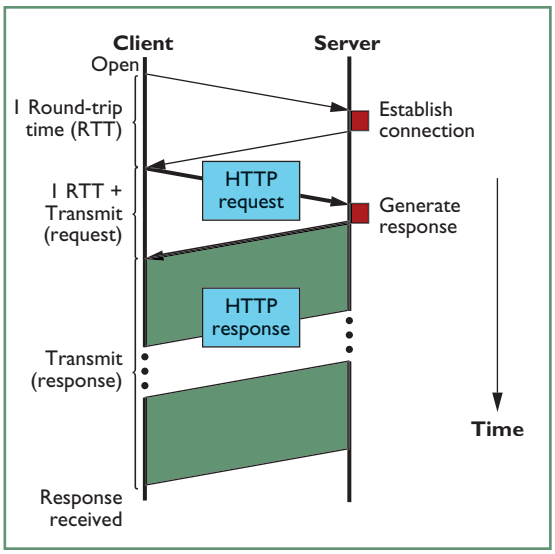

Figure 2. HTTP transfer timing costs with a new connection. The amount of time to retrieve a resource when a new connection is required can be approximated by two round-trip times plus the time to transmit the response (plus DNS resolution delays, if necessary).

HTML form). The first line of the response header shows the HTTP version supported and a response code with standard values.

The headers of an HTTP transaction also specify aspects relevant to an object's cacheability. The relevant headers from the example in Figure 3 include `Date`, `Last-Modified`, `ETag`, `Cache-Control`, and `Expires`. For example, in HTTP GET requests that include an `If-Modified-Since` header, Web servers use the `Last-Modified` date on the current content to return the object only if the object changed after the date of the cached copy. The origin server needs an accurate clock to calculate and present modification and expiration times in the other tags.

An `ETag` (entity tag) represents a signature for the object and allows for a stronger test than `If-Modified-Since:` If the signature of the current object at this URL matches the signature of the

```
Request Header:
    GET / HTTP/1.1
    Host: www.web-caching.com
    Referer: http://vancouver-webpages.com/CacheNow/
    User-Agent: Mozilla/4.04 [en] (X11; I; SunOS
    5.5.1 sun4u)
    Accept: */*
    Connection: close

Response Header:
    HTTP/1.1 200 OK
    Date: Mon, 18 Dec 2000 21:25:23 GMT
    Server: Apache/1.3.12 (Unix) mod_perl/1.18
    PHP/4.0B2
    Cache-Control: max-age=86400
    Expires: Tue, 19 Dec 2000 21:25:23 GMT
    Last-Modified: Mon, 18 Dec 2000 14:54:21 GMT
    ETag: "838b2-4147-3a3e251d"
    Accept-Ranges: bytes
    Content-Length: 16711
    Connection: close
    Content-Type: text/html
```

Figure 3. Sample HTTP request and response headers. Headers identify client and server capabilities as well as describe the response content.

cached one, the objects are considered equivalent. The `Expires` and `Cache-Control: max-age` headers specify how long the object can be considered valid. For slowly or never-changing resources, an explicit expiration date tells caches how long they can keep the object (without requiring the cache to contact the origin server to validate it).

Caching Web Resources

Web caching is similar to memory system caching — a Web cache stores Web resources in anticipation of future requests. However, significant differences between memory system and Web caching result from the nonuniformity of Web object sizes, retrieval costs, and cacheability.

To address object size, cache operators and designers track both the overall *object hit rate* (percentage of requests served from cache) and the overall *byte hit rate* (percentage of bytes served from cache). Traditional replacement algorithms often assume a fixed object size, so variable sizes can affect their performance. Retrieval cost varies with object size, distance traveled, network congestion, and server load. Finally, some Web resources cannot or should not be cached, for example, because the resource is personalized to a particular client or is constantly updated.

Caching is performed in various locations throughout the Web, including at the two endpoints known to a typical user — the Web browser and Web server. Figure 4 shows a possible chain of caches through which a request and response might flow. *Proxy caches,* intermediary caches between the client machine and the origin server, will generate new requests on behalf of users if they cannot satisfy the requests themselves. If a response captured in a browser cache does not satisfy a user, the request might be passed to a department- or organization-wide proxy cache. If a valid response is not present there, a proxy cache operated by the client's ISP might receive the request. If the ISP cache does not contain the requested response, it will likely attempt to contact the origin server. However, *reverse proxy caches* operated by the content provider's ISP or CDN might instead respond to the request. If they do not have the requested information, the request might ultimately arrive at the origin server. Even at the origin server, content, or portions of content, can be stored in a server-side cache to reduce the server load (for instance, by reducing the need for redundant computations or database retrievals).

The response flows through the reverse path back to the client. Each step in Figure 4 has multiple arrows, signifying relationships with multiple entities at each level. For example, the reverse proxy (sometimes called an HTTP accelerator), operated by the content provider's ISP or CDN, can serve as a proxy cache for the content from multiple origin servers, and can receive requests from multiple downstream clients (including forward caches operated by others, as shown in Figure 4).

In general, a cache need not talk only to the clients below it and the server above it. In fact, to scale to large numbers of clients, multiple caches might be necessary. In a hierarchical caching structure, each cache serves many clients, which can be users or other caches.[6] When a local cache cannot serve a request, it passes the request to a higher level in the hierarchy. If the request misses at a root cache (which has no parent), the cache requests the object from the origin server.

Figure 5 (see page 6) shows an alternative cooperative caching architecture in which caches communicate with peers using an intercache protocol such as ICP. In this form, on a miss, a cache asks a predetermined set of peers whether they have the missing object. If they do, the cache routes the request to the first responding peer cache. Otherwise, the cache attempts to retrieve the object directly from the origin server. This approach can prevent storage of multiple copies and reduce origin server retrievals, but exacts a

penalty in the form of increased intercache communication. In another variation, peers periodically receive a summary of the contents of each cache, which can significantly reduce communication overhead.[7]

Variations and combinations of these approaches have been proposed. Current thinking limits cooperative caching to smaller client populations, but both cooperative caching and combinations are used in real-world sites, particularly where network access is expensive.

A client may or may not know about intermediate proxy caches. If a client is configured to use a proxy cache directly, it sends all requests not satisfied by its built-in cache to the proxy. Otherwise, the client has to look up the IP address of the origin host. If the content provider is using a CDN, the DNS servers may be customized to return the IP address of the server (or proxy cache) closest to the client (where "closest" likely reflects network distance and additional information to avoid overloaded servers or networks). In this way, a reverse proxy server can operate as if it were the origin server, immediately answering any cached requests, and forwarding the rest.

Even when the client has the IP address of the origin server it should contact, it might never reach it. Along the network path between the client and the server, there might be a network switch or router that directs all Web requests transparently to an interception proxy cache. This approach can be used at any of the proxy cache locations shown in Figure 4. In this scenario, the client believes it has contacted the origin server, but instead the interception proxy serves the content either from cache, or by first fetching it from the origin server.

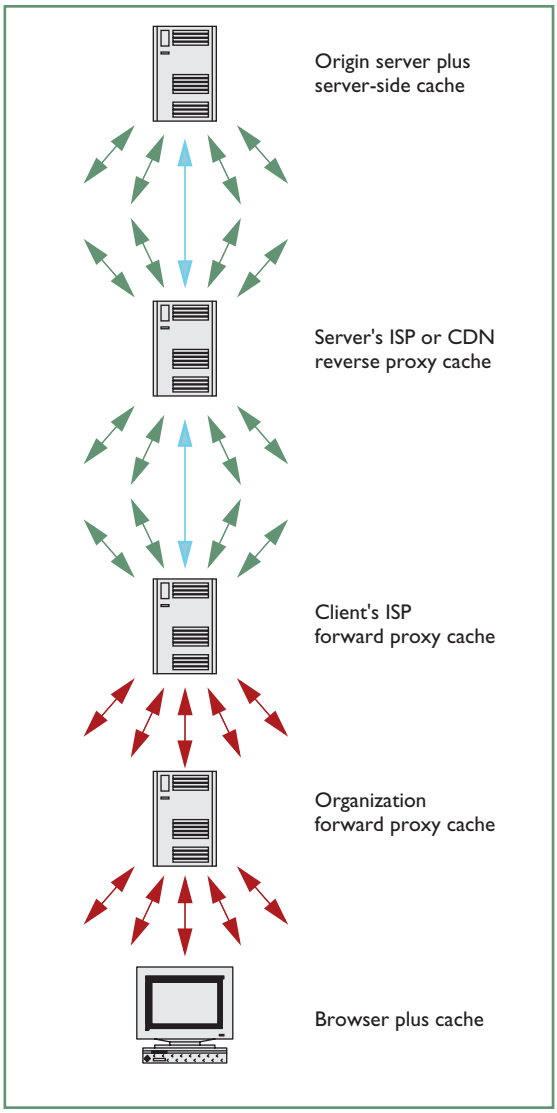

Figure 4. Caches in the World Wide Web. Starting in a browser, a Web request can travel through multiple caching systems on its way to the origin server. At any point in the sequence a response can be served if the request matches a valid response in the cache.

Benefits of Web Caching

Web caching works because of popularity – the more popular a resource is, the more likely it is to be requested in the future. In one study spanning more than a month, out of all the objects requested by individual users, on average close to 60 percent of those objects were requested more than once by the same user.[8] Likewise, much content is of value to more than one user. In fact, of the hits recorded in another caching study, up to 85 percent were the result of multiple users requesting the same object.[9]

Three features of Web caching make it attractive to all Web participants, including end users, network managers, and content creators. Caching

- reduces network bandwidth usage, which can save money for both content consumers and creators;
- lessens user-perceived delays, which increases user-perceived value; and
- lightens loads on the origin servers, saving hardware and support costs for content providers and providing consumers a shorter response time for noncached resources.

When a request is satisfied by a cache, the content no longer has to travel across the Internet from the origin Web server to the cache, saving bandwidth

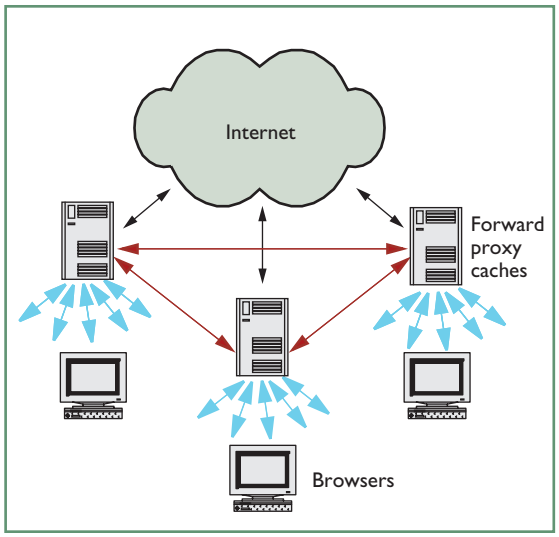

Figure 5. Cooperative caching. Caches communicate with peers before making requests over the Web.

for the cache owner as well as the origin server. TCP, the network protocol used by HTTP, has a fairly high overhead for connection establishment and sends data slowly at first. This, combined with the fact that most requests on the Web are for relatively small resources, means that reducing the number of necessary connections and holding them open (making them persistent) so that future requests can use the improves client performance.

Specifically, a client of a forward proxy can save time because it can retain a persistent connection to the proxy instead of establishing new connections with each origin server a user visits during a session. Persistent connections are particularly beneficial to clients suffering from high-latency network service (for example, clients connected to the Internet via dial-up modems).

Furthermore, busy proxies can use persistent connections to send requests for multiple clients to the same server, reducing connection establishment times to servers as well. Therefore, a proxy cache supporting persistent connections can cache connections on both client and server sides (avoiding the initial round-trip time for a new HTTP connection shown in Figure 2).

Potential Problems

There are a number of potential problems associated with Web caching. Most significant from the perspective of both the content consumer and the content provider is the possibility of the end user seeing *stale* (that is, old or out-of-date) content, compared to *fresh* content available on the origin server. HTTP does not ensure strong consistency and thus there is a real potential for data to be cached too long. The likelihood of this is a trade-off that the content provider can explicitly manage.

Second, caching tends to improve the latency only for cached responses that are subsequently requested (that is, hits). Misses that are processed by a cache generally have decreased speed, as each system through which the transaction passes will increase the latency experienced by a small amount. Thus, a cache only benefits requests for content already stored in it. Caching is also limited by the frequency with which popular Web resources change, and, importantly, the fact that many resources will be requested only once. Finally, some responses cannot or should not be cached.

Content Cacheability

Not every Web resource is cacheable. Of those that are, some can be cached for long periods by any cache, while others have restrictions such as caching for short periods or to certain kinds of caches (for instance, nonproxy caches). This flexibility maximizes the opportunity for caching individual resources. The cacheability of a Web site affects both its user-perceived performance and the scalability of a particular hosting solution. Instead of taking seconds or minutes to load, a cached object can appear almost instantaneously. Regardless of how much the hosting costs, a cache-friendly design will allow a server to serve more pages before it needs to upgrade to a more expensive solution.

Fortunately, the content provider determines its resources' cacheability. The Web server software sets and sends the HTTP headers that determine cacheability, according to the server's caching policy for that data. To maximize a Web site's cacheability, all static content (buttons, graphics, audio and video files, and pages that rarely change) are typically given expiration dates far in the future so that they can be cached for weeks or months at a time. (Note that HTML `meta` tags cannot validly specify caching properties and are ignored by most proxy caching products since proxies do not examine the contents of an object — that is, they do not see the HTML source of a Web page.)

By setting an expiration date far into the future, the content provider trades the potential of caching stale data for reduced bandwidth usage and improved user-perceived response time. A shorter expiration date reduces the chance that the user sees stale content, but increases the number of times that caches will need to validate the

resource. Currently, most caches use a client polling approach that favors revalidation for objects that have recently changed.[10] Because this can generate significant overhead (especially for popular content), a better approach might be for the origin server to invalidate the cached objects,[11] but only recently has this approach been considered for nonproprietary networks. The Web cache invalidation protocol (WCIP),[12] currently in development, lets Web caches subscribe to invalidation channels corresponding to content in which they are interested. This protocol is intended to allow large numbers of frequently changing Web objects to be cached and distributed with freshness guarantees.

Dynamically generated objects are typically considered uncacheable, although they are not necessarily so. Examples of dynamically generated objects include fast-changing content like stock quotes, personalized pages, query results (such as from search engines), and e-commerce shopping carts. Rarely would it be desirable for any of these objects to be cached at an intermediate proxy, although some might be cached in the client browser cache (such as personalized resources that don't change often).

However, dynamically generated objects constitute an increasing fraction of the Web. One way to allow for the caching of dynamic content is to cache programs (such as applets) that generate or modify the content.[13] Another is to enable the server to cache portions of documents to optimize server-side operations (for example, server-side products from SpiderCache, http://www.spidercache.com/; Persistence, http://www.persistence.com/; and XCache, http://www.xcache.com/). Since most dynamic pages include much static content, sending the differences between pages or between versions of a page,[14,15] or breaking documents into separately cacheable pieces and reassembling them at the client[16] are possible solutions. Akamai's EdgeSuite (http://www.akamai.com/) and more generally the new open protocol called Edge Side Includes (http://www.edge-delivery.org/) are essentially examples of this last solution, except that the Web pages are assembled at edge servers rather than at the client.

Cache Latency

Caching can provide only a limited benefit (object hit rates typically reach 40 percent to 50 percent with sufficient traffic), as a cache can only provide objects that have been previously requested. If future requests can be anticipated, objects can be obtained in advance. Once available in a local cache, those objects can be retrieved with minimal delay, enhancing the user experience.

Although *prefetching* shows promise, it is difficult to evaluate and has not been widely implemented in commercial systems. Some browser add-ons and workgroup proxy caches will prefetch the links of the current page, or periodically prefetch the pages in a user's bookmarks. One significant difficulty is accurately predicting which resources will be needed next to minimize mistakes that result in wasted bandwidth and increased server loads. Another concern is determining what content can be prefetched safely, as some Web requests have potentially undesirable side effects, such as adding items to an online shopping cart.[17]

Recently, Cohen and Kaplan proposed techniques that do everything but prefetch.[18] In particular, they demonstrated the usefulness of

Web Caching Resources

Web Sites

IETF Working Group on Web Replication and Caching • http://www.wrec.org/
Information Resource Caching FAQ • http://www.ircache.net/FAQ/
Squid: Open Source Proxy Cache Software • http://www.squid-cache.org/
Standards work on Web Cache Invalidation Protocol (WCIP) • http://www.content-signaling.org/
Web Caching and Content Delivery Resources (news, tutorials, tips, tools, discussions, and links) • http://www.web-caching.com/
W3C HTTP Protocol Page • http://www.w3.org/Protocols/
W3C HTTP Specifications and Drafts • http://www.w3.org/Protocols/Specs.html
W3C's Propagation, Caching, and Replication on the Web • http://www.w3.org/Propagation/

Survey Articles

G. Barish and K. Obraczka, "World Wide Web Caching: Trends and Techniques," *IEEE Comm. Internet Technology Series*, vol. 38, no. 5, May 2000, pp. 178–184.
J.C. Mogul, "Squeezing More Bits out of HTTP Caches," *IEEE Network*, vol. 14, no. 3, May/June 2000, pp. 6-14.
J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Comm. Rev.*, vol. 29, no. 5, Oct. 1999, pp. 36-46.

Books

B. Krishnamurthy and J. Rexford, *Web Protocols and Practice: HTTP 1.1, Networking Protocols, Caching, and Traffic Measurement*, Addison Wesley Longman, Reading, Mass., 2001.
D. Wessels, *Web Caching*, O'Reilly & Associates, Sebastopol, Calif., 2001.

The Need for Web Caching

Caching helps bridge the performance gap between local activity and remote content. In the short term, caching helps to improve Web performance by reducing the cost and end-user latency for Web access. In the long term, even as bandwidth costs continue to drop and higher end-user speeds become available, caching will continue to reap benefits for the following reasons:

- *Bandwidth will always have some cost.* The cost of bandwidth will never reach zero, even though increased competition, a growing market, and economies of scale reduce end-user costs. The cost of bandwidth at the core has stayed relatively stable, requiring ISPs to implement methods such as caching to stay competitive and reduce core bandwidth usage so that edge bandwidth costs can be low.
- *Nonuniform bandwidth and latencies will persist.* Because of physical limitations such as environment and location as well as financial constraints, there will always be variations in bandwidth and latencies. Caching can help to smooth these effects.
- *Network distances are increasing.* Firewalls, other proxies for security and privacy, and virtual private networks for telecommuters increase the number of hops through which content must travel and slow Web response times.
- *Bandwidth demands continue to increase.* Growth in the user base, in popularity of high-bandwidth media, and in user expectations of faster performance guarantee that demand for bandwidth will not end.
- *Hot spots in the Web will continue.* Intelligent load balancing can alleviate problems when high user demand for a site is predictable, but a Web site's popularity can also come as a result of current events, desirable content, or word of mouth. Distributed Web caching can help alleviate these "hot spots" resulting from flash traffic loads.
- *Communication costs exceed computational costs.* Communication is likely to always be more expensive (to some extent) than computation. We use memory caches because CPUs are much faster than main memory. Likewise, we will continue to use caches as computer systems and network connectivity both get faster.

- preresolving (performing DNS lookup in advance),
- preconnecting (opening a TCP connection in advance), and
- prewarming (sending a dummy HTTP request to the origin server).

These techniques can be implemented in both proxies and browsers, and could significantly reduce latencies without prefetching content.

Conclusion

Given the choices of caching products on the market today, how do you select one? The process involves determining the features (manageability, failure tolerance, scalability, and so on), and performance level (such as requests per second, bandwidth saved, and average latency, often measured by cache benchmarking services) you require. Likewise, as caches and content delivery services replace origin servers in serving Web traffic, content developers should consider how to maximize the usefulness of these technologies.

Acknowledgments

Thanks are due to Haym Hirsh, Vincenzo Liberatore, and anonymous reviewers for comments that greatly improved this tutorial. This work has been supported in part by the U.S. National Science Foundation under grant ANI 9903052.

References

1. O.A. McBryan, "GENVL and WWW: Tools for Taming the Web," *Proc. First Int'l World Wide Web Conf.*, Elsevier, New York, 1994, pp. 79-90.
2. S. Lawrence and C.L. Giles, "Accessibility of Information on the Web," *Nature*, vol. 400, July 1999, pp. 107–109; http://www.neci.nj.nec.com/homepages/lawrence/papers.html.
3. "The Economic Impacts of Unacceptable Web Site Download Speeds," white paper, Zona Research, 1999; available at http://www.zonaresearch.com/deliverables/white_papers/wp17/index.htm.
4. D. Wessels and K. Claffy, "Internet Cache Protocol (ICP)," version 2, Internet Eng. Task Force RFC 2186, Sept. 1997, available at http://ftp.isi.edu/in-notes/rfc2186.txt.
5. A.J. Smith, "Cache Memories," *ACM Computing Surveys*, vol. 14, no. 3, Sept. 1982, pp. 473–530.
6. A. Chankhunthod et al., "A Hierarchical Internet Object Cache," *Proc. Usenix 1996 Ann. Technical Conf.*, Usenix Assoc., Berkeley, Calif., 1996, pp. 153-163.
7. A. Rousskov and D. Wessels, "Cache Digests," *Computer Networks and ISDN Systems*, vol. 30, no. 22–23, Nov. 1998, pp. 2155–2168.
8. L. Tauscher and S. Greenberg, "How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems," *Int'l J. Human Computer Studies*, vol. 47, no. 1, 1997, pp. 97–138.
9. B.M. Duska, D. Marwood, and M.J. Feely, "The Measured Access Characteristics of World-Wide-Web Client Proxy Caches," *Proc. Usenix Symp. Internet Technologies and Systems (USITS 97)*, Usenix Assoc., Berkeley, Calif., 1997, pp. 23-36.
10. J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," *Proc. Usenix 1996 Ann. Technical Conf.*, Usenix Assoc., Berkeley, Calif., 1996, pp. 141-151.
11. C. Liu and P. Cao, "Maintaining Strong Cache Consistency for the World-Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, Apr. 1998, pp. 445–457.
12. D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," IETF Internet draft, work in progress, Mar. 2001.
13. P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching

Dynamic Contents on the Web," *Distributed Systems Eng.*, vol. 6, no. 1, 1999, pp. 43–50.
14. B.C. Housel and D.B. Lindquist, "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment," *Proc. Second Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 96)*, ACM Press, New York, Nov. 1996, pp. 108–116.
15. J. Mogul et al., "Potential Benefits of Delta-Encoding and Data Compression for HTTP," *Proc. ACM SIGCOMM*, ACM Press, New York, 1997, pp. 181–194. An extended and corrected version appears as Research Report 97/4a, Digital Equipment Corp. Western Research Laboratory, Dec. 1997.
16. F. Douglis, A. Haro, and M. Rabinovich, "HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching," *Proc. Usenix Symp. Internet Technologies and Systems (USITS 97)*, Usenix Assoc., Berkeley, Calif., Dec. 1997, pp. 83–94.
17. B.D. Davison, "Assertion: Prefetching with GET is Not Good," *Proc. Sixth Int'l Workshop Web Caching and Content Distribution*, Elsevier, Boston, June 2001.
18. E. Cohen and H. Kaplan, "Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency," *Proc. IEEE INFOCOM*, IEEE Press, Piscataway, N.J., Mar. 2000.

Brian D. Davison is currently a PhD candidate in computer science at Rutgers University, but will join Lehigh University in September as an assistant professor of computer science and engineering. He earned a BS in computer engineering from Bucknell University and an MS in computer science from Rutgers. His research interests include Web performance technologies, information retrieval, and machine learning. He is a member of the AAAI, the ACM, and the IEEE Computer and Communications Societies. He also created and maintains www.web-caching.com. For more on Davison's work, see http://www.cs.rutgers.edu/~davison/.

Readers can contact the author at Department of Computer Science, Rutgers, The State University of New Jersey, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019, davison@web-caching.com.