Assertion: Prefetching With GET Is Not Good

Brian D. Davison[*]
Department of Computer Science
Rutgers, The State University of New Jersey
110 Frelinghuysen Road
Piscataway, NJ 08854 USA
`davison@cs.rutgers.edu`

Abstract

The benefits of Web cache prefetching are well understood, and so prefetching has been implemented in a number of commercial products. This paper argues that the current support for prefetching in HTTP/1.1 is insufficient because *prefetching with GET is not good*. Existing prefetching implementations can cause problems with undesirable side-effects and server abuse, and the potential for these problems may thwart additional prefetching development and deployment. We make some initial suggestions of extensions to HTTP that would allow for safe prefetching, reduced server abuse, and differentiated Web server quality of service. It is our hope that this paper will restart a dialog on these issues that will move in time into a standards development process.

Keywords

HTTP, prefetching, Web caching, QOS

1 Introduction

Prefetching into a local cache is a well-known technique for reducing latency in distributed systems. However, prefetching is not widespread in the Web today. There are a number of possible explanations for this, including

- the bandwidth overhead of prefetching data never utilized,
- the potential increase in queue lengths and delays from naive prefetching approaches [15],
- the reported bounds on the potential of prefetching [30], and
- the effect on the Internet (e.g., added traffic and server loads) if all clients were to implement prefetching.

Fortunately, the first two objections can be satisfied by more careful approaches, and the third still allows for a significant reduction in latency from prefetching, and only considers proxy-based prefetching. The fourth is still up for debate, but when prefetching occurs over a private connection for objects stored on an intermediate proxy (as in proxy initiated prefetching [23]), this is no longer an issue.

This paper will first attempt to define prefetchability and then review how Web prefetching has been suggested by many researchers and implemented in a number of commercial systems. In section 3 we will discuss a number of less-well-investigated problems with prefetching, and with the use of GET for prefetching in particular. In effect we argue that 1) GET is unsafe because in practice, GET requests may have (not insignificant) side-effects, and 2) GET can unfairly use server resources. As a result, we assert that *prefetching with GET is not good*. We then propose extending HTTP/1.1 [24] to incorporate methods and headers that will support prefetching and variable Web server quality of service.

2 Background

While some researchers have considered prepushing or presending, this paper is only concerned with client-activated preloading of content. We wish to set aside concerns of how the client chooses what to prefetch (e.g., Markov model of client history, bookmarks on startup, user-specified pages, the use of proxy or server hints), with the understanding that a hints-based model (as suggested in [38, 34, 20] and others) has the potential to avoid many of the concerns discussed here but may also introduce other problems and likewise lacks specification as a standard.

Therefore, we make the not-unreasonable assumption that a prefetching system (at the client or intermediate

[*]New address: Department of Computer Science and Engineering, Packard Lab, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015.

Prefetchability Quiz

```
http://www.cnn.com/
http://www.ibm.com/
http://www.geocities.com/
http://www.microsoft.com/
http://www.napster.com/
http://www.volera.com/
http://www.bn.com/
```

Figure 1: Which of these URLs are prefetchable? (Answers are in Appendix A.)

proxy) has a list of one or more URLs that it desires to prefetch.

2.1 Prefetchability

Can prefetchability be defined? Let us start with prefetching. A relatively early paper [19] defines prefetching as a cache that ”periodically refreshes cached pages so that they will be less stale when requested by the user.” While this definition does have a Web-specific perspective, it does not reflect current usage of the term as it is specific to cache-refreshment. A more general definition is:

> *Prefetching* is the (cache-initiated) speculative retrieval of a resource into a cache in the anticipation that it can be served from cache in the future. (1)

Typically the resource has not yet been requested by the client (whether user or machine), but the definition applies equally to cache refreshes (when the content has changed). Note that with this definition we can see that:

- Prefetching requires the use of a cache.
- Prefetchability requires cacheability.

Note that not all prefetched resources will be used before they expire or are removed from the cache. Some will never be requested by the client. This implies that:

- Prefetching requires at least idempotence, since resources may end up being fetched multiple times.
- Speculative prefetchability requires safety. Unless we can guarantee that the client will actually use this resource (before removal or expiration), a prefetching system should not perform any action that has unwanted (from either the client or server's perspective) side-effects at the server.

Perhaps now we can propose a working definition:

> A Web resource is *prefetchable* if and only if it is cacheable and its retrieval is safe. (2)

Typical concerns for safety (e.g., [25]) are really concerns for idempotency, but idempotency is insufficient for prefetching, as the prefetched response might never be seen or used. Here we are referring to the definition of safe as having no significant side-effects (see section 9.1.1 of the HTTP/1.1 specification included in Appendix B).

2.2 Prefetching research

Prefetching for the Web is not a new concept. Discussions of mechanisms and their merits date back at least to 1994-1995.[1] Many papers have been published on the use of prefetching as a mechanism to improve the latencies provided by caching systems (e.g., [38, 34, 46, 16, 11, 26, 33, 39, 44, 3, 14, 27, 20, 23, 42, 45]).

2.3 Prefetching in the Web today

A number of commercial systems used today implement some form of prefetching. CacheFlow [10] implements prefetching for inline resources like images, and can proactively check for expired objects. There are also a number of browser extensions for Netscape and Microsoft Internet Explorer as well as some personal proxies that perform prefetching of links of the current page (and sometimes bookmarks, popular sites, etc.). Examples include NetSonic [47], PeakJet 2000 [40], and Webcelerator [22].

3 Problems with prefetching

Even with a proliferation of prefetching papers in the research community, and examples of its use in commercial products, there are a number of well-known difficulties with prefetching as described in the Introduction.

Some researchers even suggest pre-executing the steps leading up to retrieval [12] as one way to avoid some of the difficulties of content retrieval and still achieve significant latency savings. However, as suggested above, researchers and developers are still interested in the performance enhancements possible with speculative data transfer. When prefetching (and not prepushing/presending), the temptation for most system builders is to use the HTTP GET mechanism. Advantages of GET and standard HTTP request headers include:

[1]For one interesting discussion still accessible, see the threads “two ideas...” and “prefetching attribute” from the www-talk mailing list in Nov-Dec 1995 at `http://lists.w3.org/Archives/Public/www-talk/1995NovDec/thread.html`.

- GET is well-understood and universally used.
- There are supporting libraries to ease implementation (e.g., for C there is W3C's Libwww [49] and for Perl there is Libwww-perl [1]).
- It requires no changes on the server's side.

However, the use of the current version of GET has some obvious and non-obvious drawbacks, which we discuss in this section. Even from the prefetching client's perspective, there may be some drawbacks. In particular, just like users, a naive prefetcher may not know that the object requested is very large and will take minutes or hours to arrive.

In many cases, however, these drawbacks can be characterized as server abuses, because the inefficiencies and deleterious effects described could be avoided if the content provider were able to decide what and when to prefetch. In the rest of this section we detail the problems with prefetching, with emphasis on the consequences of using the standard HTTP GET mechanism in the current Web.

3.1 Unknown cacheability

One of the differences between Web caching and traditional caching in file and memory systems is that the cacheability of Web objects varies. One cannot look at the URL of an object and *a priori* know that the response generated for that URL will be cacheable or uncacheable (see Figure 1). Some simple heuristics are known, including examining the URL for substrings (e.g., "?" and "cgi-bin") that are commonly associated with fast-changing or individualized content, for which caching does not provide much use. However, while a URL with an embedded "?" has special meaning for HTTP/1.0 caching (see the HTTP/1.1 specification [24], section 13.9, excerpted in Appendix B), a significant fraction of content that might be labeled dynamic (e.g., e-commerce site queries) does not change from day to day [48] and should be treated as cacheable unless marked otherwise (as stated by the HTTP/1.1 specification). Only the content provider will know for certain, which is perhaps why the cacheability of a response is determined by one or more HTTP headers that travel with the response.

Some researchers (e.g., [20]) have proposed prefetching and caching data that is otherwise uncacheable, on the grounds that it will be viewed once and then purged from the cache. However, the HTTP specification [24] states that such non-transparent operations require an explicit warning to the end user, which may be undesirable. Since many objects are marked uncacheable just for hit metering purposes, Mogul and Leach suggest something similar in their "Simple Hit-Metering and Usage-Limiting for HTTP" RFC [35] which states in part (*emphasis added*):

> Note that the limit on "uses" set by the max-uses directive does not include the use of the response to satisfy the end-client request that caused the proxy's request to the server. *This counting rule supports the notion of a cache-initiated prefetch: a cache may issue a prefetch request, receive a max-uses=0 response, store that response, and then return that response (without revalidation) when a client makes an actual request for the resource.* However, each such response may be used at most once in this way, so the origin server maintains precise control over the number of actual uses.

Under this arrangement, the response no longer has to be marked uncacheable for the server to keep track of views and can even direct the number of times that the object may be served.

In summary, though, when a client or proxy prefetches an object that it is not permitted to cache, it has wasted both client and server resources to no avail.

3.2 Server overhead

Even when the object fetched is cacheable, the server may have non-negligible costs in order to provide it. The retrieval will consume some portion of the server's resources including bandwidth and cpu, and logging of unviewed requests. Likewise, the execution of some content retrievals may not be desirable from a content-provider's point of view, such as the instigation of a large database search. In fact, the use of overly aggressive prefetchers has been banned at some sites [41].

As more attention is paid to end-user quality of service for the Web (e.g., [8]), the content provider may want to be able to serve prefetched items at a lower quality of service so that those requests do not adversely impact the demand-fetched requests.

3.3 Side effects of retrieval

According to the HTTP/1.1 specification [24], GET and HEAD requests are not supposed to have a significance other than retrieval and should be considered safe (the relevant portions of the draft standard are included in Appendix B). In fact, the GET and HEAD methods are supposed to be idempotent methods. In reality, many requests do have side-effects (and are thus not idempotent).

Often these side effects are the reason for some objects to be made uncacheable (that is, the side effects are desirable by the content producers, such as for logging usage).

More importantly, some requests, when executed speculatively, may generate undesirable effects for either the content owner, or the intended content recipient. Placing an item into an electronic shopping basket is an obvious example — a prefetching system, when speculatively retrieving links, does not want to end up placing the items corresponding to those links into a shopping cart without the knowledge and consent of the user.

This reality is confirmed in the help pages of some prefetching products. For example, one vendor [21] states:

> On sites where clicking a link does more than just fetch a page (such as on-line shopping), Webcelerator's Prefetching may trigger some unexpected actions. If you are having problems with interactive sites (such as e-mail sites, on-line ordering, chat rooms, and sites with complex forms) you may wish to disable Prefetching.

Idempotency is a significant issue and is also mentioned in other RFCs (e.g., [7, 5, 6]). It is additionally an issue for pipelined connections in HTTP/1.1 (prefetching or otherwise) as non-idempotent requests should not be retried if a pipelined connection were to fail before completion.[2] The experimental RFC 2310 [25] proposes the use of a new response header called Safe which would tell the client that the request may be repeated without harm, but this is only useful after the client already has header information about the resource. The lack of guaranteed safe prefetching also complicates live proxy evaluation [17].

The HTTP/1.1 specification [24] does note that "it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request." A conscientious content developer would not only need to use POST to access all potentially unsafe resources but to prevent GET access to those resources, as linking to content cannot be controlled on the WWW. There may also be some confusion as to the interpretation of the idempotence of GET as a MUST rather than a SHOULD, and what constitutes safety and its significance (as a SHOULD requirement). In any case, enough content providers have built websites (perhaps without thought to prefetching) that do generate side-effects from GET requests that careful developers are unlikely to develop widespread prefetching functionality, for fear of causing problems.

[2]HTTP idempotency continues to be significant. For a relatively recent discussion of HTTP idempotency with emphasis on idempotent sequences, see the discussion thread "On pipelining" from the IETF HTTP working group mailing list archives at `http://www.ics.uci.edu/pub/ietf/http/hypermail /2000/thread.html`.

3.4 Related non-prefetching applications

There are other situations in which server resources are potentially abused. These include price-comparison systems that repeatedly retrieve pages from e-commerce sites to get the latest product pricing and availability, and extensive mechanical crawling of a site at too fast a rate [32]. Both of these can cause undue loads at a Web site and affect performance of the demand-fetched responses. While the need for an improved interface between crawlers and servers has been recognized (e.g., [9]), the need for distinguishing between demand– and non-demand-requests has not been suggested. Interestingly, there has been recent work [43] to build crawlers that attempt to access the "hidden Web" — those pages behind search forms [4], further blurring the semantic distinction between GET and POST methods.

A server may also benefit from special handling of other non-demand requests, such as non-demand-based cache refreshment traffic [13], in which additional validation requests are made to ensure freshness of cached content. From the content provider's perspective, it would be helpful in general to serve non-demand requests of any kind at a different quality of service than those requests that come from a client with a human waiting for the result in real-time.

3.5 User activity conflation

Non-interactive retrieval also has the potential for generating anomalous server statistics, in the sense that a page retrieval is assumed to be represent a page view. Almost certainly some fraction of prefetched requests will not be displayed to the user, and so the logs generated by the server may overestimate and likely skew pageview statistics. This is also the case for other kinds of non-interactive retrievals, such as those generated by Web crawlers. Today, when analyzing server logs for interesting patterns of usage, researchers must first separate (if possible) the retrievals from automated crawlers [29, 18].

Additionally, if the server were maintaining a user model based on history, undistinguished prefetching requests could influence that model, which could generate incorrect hints that would cause additional requests for useless resources which would likewise (incorrectly) influence the server's model of the user, and so on. A server that is able to distinguish such requests from demand traffic would also be able to distinguish them in logs and in user model generation.

4 Proposed extensions to HTTP

The preceding section has discussed the problems with prefetching on the Web using current methods. In this section, we describe the need for an extension (as opposed to just new headers) for HTTP, mention briefly some previous proposals for HTTP prefetching extensions, and then provide a starting point for recommended changes to the protocol.

4.1 Need for an extension

HTTP is a generic protocol that can be extended through new request methods, error codes, and headers. An extension is needed to allow the server to define the conditions under which prefetching (or any other non-demand request activity) is allowed or served.

The alternative to an extension is more appropriate use of the current protocol. One approach, say, to eliminate some wasted effort would be to use range-requests (much like Adobe's Acrobat Reader [2] does) to first request a portion of the document. If the document was small, then there is a chance the whole object will be retrieved. In any case, the headers would be available. A slight variation would be to use HEAD to get meta-information about the resource. That would allow the client to use some characteristics of the response (such as size or cacheability) to determine whether the retrieval is useful. These suggestions do not address the problems of request side-effects and high back-end overhead which would not be visible from headers, and in fact may be exacerbated by HEAD requests (e.g., if the server has to generate a dynamic response to calculate values such as Content-length). Neither do they allow for a variable quality of service at the Web server.

A second alternative would be to classify systems performing prefetching activity as robots, and require compliance with the so-called "robot exclusion standard" [28]. However, conformity with this non-standard is voluntary, and is ignored by many robots. Further, compliance does not ensure a prevention of server abuse, as there is no restriction on request rates, allowing robots to overload a server at whim.

A third approach would be to incorporate a mechanism to support mandatory extensions to HTTP without protocol revisions. Such a mechanism has been proposed in Experimental RFC 2774 [37], which allows for the definition of mandatory headers. It modifies existing methods by prefixing them with "M-" to signify that a response to such a request must fulfill the requirements of the mandatory headers, or report an error. If RFC 2774 were widely supported, this proposal might only need to suggest the definition of new mandatory headers that allow for prefetching and other non-interactive requests to be identified.

4.2 Previous proposals

Other researchers have proposed extensions to HTTP to support prefetching in one form or another.

- In his masters thesis, Lee [31] prepared a draft proposal for additional HTTP/1.1 headers and methods. PREDICT-GET is the same as GET, but allows for separate statistical logging (so that logs are not skewed with prefetching requests). PREDICT-GOT is a method equivalent to HEAD, but is to be used as a way for the client to tell the server that the prefetched item was eventually requested by the client. Lee also lists new headers that would specify what kinds of predictions are desired by the client, and the kinds of predictions returned by the server.
- Padmanabhan and Mogul [38] suggest adding a new response header to carry predictions (termed Prediction) that would provide a list of URLs, their probabilities, and possibly their size.
- Duchamp [20] defines a new HTTP header (termed Prefetch) along with a number of directives. The new header is included in both requests and responses to pass information on resource usage by the client and suggested resources for prefetching by the server. The actual retrieval is performed using GET.

Unlike these proposals, we would like to see changes that are independent of a particular prediction technique.

4.3 Recommending changes

Any realistic set of changes will need to be agreed upon by a large part of the Web community, and so it makes sense to develop a proposal for these changes in consultation with many members of that community. However, there are certain aspects to the changes that appear to be sensible with respect to the type of prefetching described in this document.

We argue that a new method is needed. It should be treated similarly to the optional methods in HTTP/1.1. The capability of the server could be tested by attempting to use the new method, or by using the OPTIONS method to list supported methods. A new method provides safety — if a client uses a new method, older servers that do not yet support prefetching would return the 501 Not Implemented status code, rather than automatically serving the request, which would happen with the use of a new header accompanying a GET request. If the server does support these prefetching extensions, it would have the option to serve the request as if it were a normal GET, or at a lower-priority, or to deny the request with a 405 Method Not Allowed status code. When a client receives some kind of error code using the new method, it may choose to re-try the request with a traditional GET, but with the understanding that such action may have undesirable side-effects.

Therefore, we suggest a second version of GET (arbitrarily called `GET2`). A new header (possibly called `GET2-Constraint`) may also be useful to allow the client to specify constraints (e.g., `safe` to guarantee safety of the processing of this request, `max-size` to prevent fetching even a portion of an object that is too large, `min-fresh` to prevent fetching an object that will expire too quickly, `max-resp-time` to not bother if the response will take too long, etc.). Note that some of the `Cache-control` header values may be relevant here (like `min-fresh`), even though `GET2` requests may be made to servers as well as caches. Another header might be useful to allow client-type identification (e.g., interactive for browsers and other clients needing immediate response, non-interactive for prefetching systems that will eventually send this content to users, robot for non-interactive systems that will parse and use content directly, etc.), but it may also be sufficient to assume that `GET2` requests will only be generated by non-interactive systems.

In summary, we recommend the development of a protocol extension that, when implemented by non-interactive clients that would otherwise use GET, allows for absolutely safe operation (no side-effects). When implemented by servers, gives them the ability to distinguish between interactive and non-interactive clients and thus potentially provide different levels of service.

5 Summary

The benefits of Web cache prefetching are well understood. Prefetching could be a standard feature of today's browsers and proxy caches. (Older versions of Netscape Navigator even have some prefetching code present, but disabled [36].) This paper has argued that the current support for prefetching in HTTP/1.1 is insufficient. Existing implementations can cause problems with undesirable side-effects and server abuse, and the potential for these problems may thwart additional development. We have made some initial suggestions of extensions to HTTP that would allow for safe prefetching, reduced server abuse, and differentiated Web server quality of service.

Overall, the conclusions we reach about the current state of affairs are the following:

- Prefetching with GET can be applied safely and efficiently between client and proxy under HTTP/1.1 using the `only-if-cached` and `min-fresh` options to the `Cache-control` header. Proxy resources can be abused, however, by a client prefetching content that will never be used, and prefetching effectiveness is limited to the contents of the cache.

- The safety nor efficiency of prefetching from an origin server cannot be guaranteed, as the origin server cannot does not recognize non-demand requests and so must serve every request, including requests for content that is uncacheable or has side effects.

- Proxies and origin servers cannot provide different qualities of service to demand and non-demand requests, again because they cannot distinguish between demand requests and non-demand requests (generated by prefetchers and Web robots).

- Much of the uncacheable content of the Web can be prefetched by clients in the sense that the content can be cached if semantic transparency is allowed to be compromised (although user warnings are still required under HTTP/1.1).

An extension to HTTP could rectify the second and third problems and allow for safe, semantically transparent prefetching for those systems which support it, and "gracefully" degrade to the existing situation if necessary when the extension is not supported by either the client or the server. It is our hope that this paper will restart a dialog on these issues that will move in time into a standards development process.

Acknowledgments

Thanks are due to Vincenzo Liberatore, Chandrasekar Krishnan, Haym Hirsh, and to anonymous referees for valuable suggestions. This work is supported in part by the National Science Foundation under grant NSF ANI 9903052.

References

[1] G. Aas. libwww-perl home page. `http://www.linpro.no/lwp/`, 2001.

[2] Adobe Systems Incorporated. Adobe Acrobat Reader. `http://www.adobe.com/products/acrobat/readermain.html`, 2001.

[3] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Pre-sending documents on the WWW: a comparative study. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 2, pages 1274–1279, Stockholm, Sweden, 1999. Morgan Kaufmann.

[4] M. K. Bergman. The deep Web: Surfacing hidden value. White paper, BrightPlanet.com, July 2000. Available from `http://www.completeplanet.com/tutorials/deepweb/index.asp`.

[5] T. Berners-Lee and D. Connolly. Hypertext markup language - 2.0. RFC 1866, `http://ftp.isi.edu/in-notes/rfc1866.txt`, Nov. 1995.

[6] T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. RFC 2396, `http://ftp.isi.edu/in-notes/rfc2396.txt`, Aug. 1998.

[7] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (URL). RFC 1738, `http://ftp.isi.edu/in-notes/rfc1738.txt`, Dec. 1994.

[8] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into Web server design. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, May 2000.

[9] O. Brandman, J. Cho, H. Garcia-Molina, and S. Shivakumar. Crawler-friendly Web servers. *Performance Evaluation Review*, 28(2), Sept. 2000. Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, June 2000.

[10] CacheFlow Inc. Active caching technology. `http://www.cacheflow.com/technology/whitepapers/active.cfm`, 2001.

[11] K.-i. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, June 1997.

[12] E. Cohen and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing Web latency. In *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000.

[13] E. Cohen and H. Kaplan. Refreshment Policies for Web Content Caches. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, Apr. 2001.

[14] E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to Web servers. In *IEEE INFOCOM '99*, New York, Mar. 1999.

[15] M. Crovella and P. Barford. The Network Effects of Prefetching. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, 1998. More detailed version available as Boston University Computer Science Department Technical Report, TR-97-002, February 1997.

[16] C. R. Cunha and C. F. B. Jaccoud. Determining WWW User's Next Access and Its Application to Prefetching. In *Proceedings of Second IEEE Symposium on Computers and Communications (ISCC'97)*, Alexandria, Egypt, July 1997.

[17] B. D. Davison. Simultaneous Proxy Evaluation. In *Proc. of the Fourth International Web Caching Workshop (WCW99)*, pages 170–178, San Diego, CA, Mar. 1999.

[18] B. 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.

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

[20] D. Duchamp. Prefetching hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, Oct. 1999.

[21] eAcceleration Corporation. Webcelerator help page on prefetching. `http://www.webcelerator.com/webcelerator/prefetch.htm`, 2001.

[22] eAcceleration Corporation. Webcelerator home page. `http://www.webcelerator.com/webcelerator/`, 2001.

[23] L. Fan, Q. Jacobson, P. Cao, and W. 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.

[24] R. T. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616, `http://ftp.isi.edu/in-notes/rfc2616.txt`, June 1999.

[25] K. Holtman. The safe response header field. RFC 2310, `http://ftp.isi.edu/in-notes/rfc2310.txt`, Apr. 1998.

[26] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications*, 16(3):358–368, Apr. 1998.

[27] R. P. Klemm. WebCompanion: A friendly client-side Web prefetching agent. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):577–594, July/August 1999.

[28] M. Koster. A standard for robot exclusion. Available from: `http://www.robotstxt.org/wc/norobots.html`, 1994.

[29] B. Krishnamurthy and J. Rexford. Software Issues in Characterizing Web Server Logs. In *World Wide Web Consortium Workshop on Web Characterization*, Cambridge, MA, Nov. 1998. Position paper.

[30] T. M. Kroeger, D. D. E. Long, and J. 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)*, Dec. 1997.

[31] D. C. Lee. Pre-fetch document caching to improve World-Wide Web user response time. Master's thesis, Virginia Tech., Blacksburg, VA, Mar. 1996.

[32] J. C. Luh. No bots allowed! *Interactive Week*, Apr. 2001. Online at `http://www.zdnet.com/intweek/stories/news/0,4164,2707542,00.html`.

[33] E. P. Markatos and C. E. Chronaki. A top–10 approach for prefetching the Web. In *Proceedings of the Eighth Annual Conference of the Internet Society (INET'98)*, Geneva, Switzerland, July 1998.

[34] J. C. Mogul. Hinted caching in the Web. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, Sept. 1996.

[35] J. C. Mogul and P. Leach. Simple hit-metering and usage-limiting for HTTP. RFC 2227, `http://ftp.isi.edu/in-notes/rfc2227.txt`, Oct. 1997.

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

[37] H. Nielsen, P. Leach, and S. Lawrence. An HTTP extension framework. RFC 2774, `http://ftp.isi.edu/in-notes/rfc2774.txt`, Feb. 2000.

[38] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World Wide Web latency. *Computer Communication Review*, 26(3):22–36, July 1996. Proceedings of SIGCOMM '96.

[39] T. Palpanas. Web prefetching using partial match prediction. Master's thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, CA, Mar. 1998. Available as Technical Report CSRG-376.

[40] PeakSoft Corporation. PeakJet 2000 web page. `http://www.peaksoft.com/peakjet2.h tml`, 2001.

[41] J. Pelline. Accelerators cause headaches. *c|net News.com*, Dec. 1996. `http://news.cnet.com/news/0,10000,0-1003-200-314937,00.html`.

[42] J. E. Pitkow and P. L. Pirolli. Mining longest repeated subsequences to predict World Wide Web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, Oct. 1999.

[43] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. Technical Report 2000-36, Computer Science Dept., Stanford University, Dec. 2000.

[44] S. Schechter, M. Krishnan, and M. D. Smith. Using path profiles to predict HTTP requests. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, Apr. 1998.

[45] N. Swaminathan and S. V. Raghavan. Intelligent prefetching in WWW using client behavior characterization. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000.

[46] Z. Wang and J. Crowcroft. Prefetching in World-Wide Web. In *Proceedings of IEEE Globecom*, London, Dec. 1996.

[47] Web 3000 Inc. NetSonic Internet Accelerator web page. `http://www.web3000.com/`, 2001.

[48] C. E. Wills and M. Mikhailov. Studying the impact of more complete server information on Web caching. In *Proc. of the Fifth International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[49] World Wide Web Consortium. Libwww — the w3c sample code library. `http://www.w3.org/Library/`, 2001.

A Answers to prefetchability quiz in Figure 1

As defined in section 2.1, prefetchability depends on the cacheability and safety of the resource. However, only the content provider can determine each of those. Thus, a quiz on prefetchability is a bit of a trick question. Since each of these are home pages, we might get away with the assumption that retrieval of their contents is safe. Thus, we need only concern ourselves with cacheability. Fortunately, cacheability is well-defined, so we can test it (albeit after fetching at least the headers of the content).[3]

- `http://www.cnn.com/` is cacheable for private (e.g., single-user) caches for the next 60 seconds. So, this page could be prefetched by a browser, but not by a generic proxy.
- `http://www.ibm.com/` does not provide a validator (e.g., `Last-Modified` or `ETag` header) so it cannot be validated (and thus is imprudent to cache).

  The same can be said for the home pages of `www.cacheflow.com`, `www.netscape.com`, `www.apple.com`, `www.inktomi.com`, `www.spidercache.com`, `www.google.com`, `www.eurosport.com`, `www.cisco.com`, `www.aol.com`, `www.att.com`, `www.f5.com`, `www.yahoo.com`, `www.altavista.com`, `www.redhat.com`, and `www.news.com`.
- `http://www.geocities.com/` has been explicitly marked stale, and uncacheable at all by intermediate proxies.
- `http://www.microsoft.com/` does not have any explicit caching headers, but does include a validator.

  Likewise, `www.akamai.com`, `www.netapp.com`, `www.infolibria.com`, `www.imimic.com`, and `www.fireclick.com` have validators but no explicit expiration.
- `http://www.napster.com/` has an explicit expiration of one day.
- `http://www.volera.com/` has an explicit expiration of 15 minutes.
- `http://www.bn.com/` is set to expire far in the past, and uncacheable by a proxy.

The moral of course is that one cannot tell the cacheability (let alone prefetchability) of a resource from its URL.

[3]These cacheability tests were performed in early June 2001 using the tools available at `www.web-caching.com`.

B Excerpts from Draft Standard RFC 2616: "Hypertext Transfer Protocol – HTTP/1.1" [24]

9.1 Safe and Idempotent Methods

9.1.1 Safe Methods

Implementors should be aware that the software represents the user in their interactions over the Internet, and should be careful to allow the user to be aware of any actions they might take which may have an unexpected significance to themselves or others.

In particular, the convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. These methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

9.1.2 Idempotent Methods

Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of $N > 0$ identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.

However, it is possible that a sequence of several requests is non- idempotent, even if all of the methods executed in that sequence are idempotent. (A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a reexecution of all, or part, of that sequence.) For example, a sequence is non-idempotent if its result depends on a value that is later modified in the same sequence.

A sequence that never has side effects is idempotent, by definition (provided that no concurrent operations are being executed on the same set of resources).

13.9 Side Effects of GET and HEAD

Unless the origin server explicitly prohibits the caching of their responses, the application of GET and HEAD methods to any resources SHOULD NOT have side effects that would lead to erroneous behavior if these responses are taken from a cache. They MAY still have side effects, but a cache is not required to consider such side effects in its caching decisions. Caches are always expected to observe an origin server's explicit restrictions on caching.

We note one exception to this rule: since some applications have traditionally used GETs and HEADs with query URLs (those containing a "?" in the rel_path part) to perform operations with significant side effects, caches MUST NOT treat responses to such URIs as fresh unless the server provides an explicit expiration time. This specifically means that responses from HTTP/1.0 servers for such URIs SHOULD NOT be taken from a cache. See section 9.1.1 for related information.