

Link Prefetching in Mozilla: A Server-Driven Approach *

Darin Fisher
darin@meer.net

Gagan Saksena
gagzilla@yahoo.com

Abstract

This paper provides a synopsis of a server-driven link prefetching mechanism that we have designed and implemented for the Mozilla web browser, a popular Open Source web browser. The mechanism depends on the origin server or an intermediate proxy server determining the best set of documents for the browser to prefetch. The browser follows prefetch directives provided by the server, either embedded in an HTML document using the <LINK> tag or specified via Link HTTP response headers. The browser determines when best to prefetch the specified URLs based on its own heuristics. In this paper, we describe the mechanism and discuss some of the practical issues that impacted its design and implementation.

1 Introduction

Web caching is a common optimization used by web browsers to reduce latency for the user when they re-visit a web page. For narrow-band Internet connections, the performance gain can be dramatic. It is of interest therefore to consider mechanisms that would enable the web browser to pre-populate its cache of documents, such that when the user initially views a web page, the latency will be nearly reduced to that of loading the web page from the local cache. This technology is commonly referred to as prefetching and has been shown to be effective in reducing latency in the Web [14].

Despite the potential benefit of prefetching, it has not been widely deployed at this time. In addition to the family of Mozilla-based browsers, only a few production web browsers (e.g., WebTV and iCab) and personal web proxies (e.g., NetSonic Pro, Naviscope, and Wcol) support some form of link prefetching. The methods of prefetching vary con-

siderably between products. For example, some of these products analyze the contents of pages or the user's browsing history to predict what the user might visit next. Indeed, a large body of research covering a variety of approaches to prefetching exist on this topic [10, 11, 12, 13, 15, 16]. Other applications rely on some cooperation between browser and server.

In addition, content authors have devised *ad-hoc* JavaScript-based techniques to prefetch content. These generally involve loading content into hidden or <IFRAME> HTML elements so as to pre-populate the browser's document cache. While these JavaScript-based approaches have the benefit of working with most browsers, they often result in complex JavaScript code that is difficult to maintain. With such techniques, the browser is not able to prioritize normal requests above prefetch requests. As a result prefetch requests may end up competing with normal requests for network bandwidth. Additionally, JavaScript-based approaches afford the server little opportunity to control traffic and limit server load resulting from prefetch requests.

In this paper we describe a link prefetching mechanism that involves modifications to both the browser and the server. It is designed to provide a solution to prefetching that is "friendlier" to both client and server, specifically addressing some of the shortcomings of existing *ad-hoc* methods. The browser follows special directives from the web server (or proxy server) that instruct it to prefetch specific documents. The server can inject these directives into an HTML document or into the HTTP response headers. The browser responds to the prefetch directives after it finishes rendering the document (or HTTP response) containing the directives. This mechanism allows servers to control precisely what is prefetched by the browser, and it

*This work was performed at Netscape/AOL during 2002.

allows the browser to determine when best to prefetch documents based on local network inactivity, browser idle time, etc.

We chose a server-driven approach to link prefetching because we believe that servers (including intermediate proxy servers) can better predict what users are likely to visit next. Since pages often contain numerous hyperlinks that the user is unlikely to follow, it is generally difficult for the browser to determine the best subset of hyperlinks to prefetch. For example, an origin server or content author may know that the user is browsing through a series of large images or documents (e.g., an image slideshow or pages in a book) in which case it would be advantageous for the browser to prefetch the next image or document in the series. Or, while the user is confronted with a login prompt, the site might like to have the browser prefetch the portions of the site that do not require user authentication. In such cases there is a static relationship between the previous and the next document, and static prefetch directives can be encoded into the documents to indicate what should be prefetched next. In addition to static prefetch directives, a server might dynamically inject prefetch directives into outbound HTTP responses for the benefit of a link prefetching enabled browser. For example, the server may want to instruct the browser to prefetch the most popular URLs on a given day. Because this technique works with HTTP headers, a proxy can very well generate such dynamically determined directives for popular links.

What follows is a description of the link prefetching mechanism along with a number of practical issues that shaped our design and implementation of link prefetching for the Mozilla browser, a popular Open Source web browser. We added our implementation to version 1.2 and 1.0.2 of the Mozilla browser. It is also included in Netscape 7.01 and other recent Mozilla-based browsers.

2 Prefetching Directives

The HTML 4.01 specification [7] briefly mentions an opportunity for browsers to prefetch URLs indicated by content authors via the `<LINK>` tag. The description of the “next” link relation type (from section 6.12 of [7]) says the following:

[The “next” link relation type] refers to the next document in a linear sequence of documents. User agents may choose to pre-load the “next” document, to reduce the perceived load time.

We have implemented this mechanism in the Mozilla browser along with some variations on this same approach, which are described below. The WebTV browser also follows this recommendation [1]. (Our prior approach involved extending all linked tags with a prefetch marker attribute [2] and using *pathfiles* [3] to prefetch dynamic content. That technique had its limitations and has since evolved into our current implementation.)

Motivated by the fact that the “next” relation is intended to represent the “next” document in a linear sequence of documents, and given that web pages should be able to specify multiple documents to be prefetched, we chose to have Mozilla additionally recognize the link relation type “prefetch” to indicate a document that should be prefetched. This link relation type currently is not standardized; however, the XHTML 2.0 draft specification [9] declares the “prefetch” link relation type.

Examples of these two types of link prefetching HTML directives follows:

```
<LINK rel="next" href="next.html">
<LINK rel="prefetch" href="big.jpg">
```

Suppose `next.html` contains an `` tag, which would cause `big.jpg` to be loaded. These tags might be found in a document in which `next.html` is likely to be the next document visited by the user. As a result of these tags, the browser would silently prefetch the two documents while the user is perhaps busy reading through the current document.

Link prefetching directives specified solely in the body of a document have limited utility in solving problems in which content to be prefetched might be dynamically determined by the origin server or an intermediate proxy server. To support such configurations, Mozilla additionally recognizes link prefetching directives specified via the `Link` HTTP header. The `Link` HTTP header appears in section 19.6.2.4 of RFC 2068 [6]; however, the updated version of the HTTP 1.1 standard, RFC 2616 [8] does not define it. Use of this header, for example

to enable servers to apply style sheets to a group of documents, is described in section 14.6 of the HTML 4.01 specification [7]. The link prefetching example from above could be equivalently written as follows:

Link: <next.html>; rel="next"

Link: <big.jpg>; rel="prefetch"

Note that as with many HTTP response headers, the Link header may also appear in a <META> HTML tag via the `http-equiv` attribute. This is described in section 7.4.4 of the HTML 4.01 specification [7].

We recommend the HTTP-level Link header mechanism above the other mechanisms since it allows proxy servers to more easily intercept prefetch directives and, for example, process them on behalf of a browser that does not support link prefetching. Proxy servers could also remove prefetch directives from the headers before sending them to its clients as a way of limiting the prefetching done by its clients. This might be desirable if the clients connect to the proxy over a high bandwidth connection such that prefetching between client and proxy would have little benefit.

3 Determining When To Prefetch

Given these link prefetching directives, it is up to the browser to determine when best to load the specified documents. This can be based on any number of heuristics chosen by the browser. For documents already in the browser's cache, the browser need not prefetch the document; however, if those documents have expired, the browser may choose to validate the cached documents. For documents the browser chooses to prefetch, an obvious heuristic is to defer prefetching until the browser's network connection is otherwise underutilized. Determining such a state of the user's system is in general a complex problem and one that requires special knowledge of the underlying operating system. APIs to access such information generally vary from platform to platform and are non-existent on some platforms. A simplistic approximation to this, and indeed what Mozilla implements, is to defer prefetching until the browser is idle (i.e., when the browser has loaded all content on the page) and to stop prefetching when the user, or JavaScript on a

page, directs the browser to load something else.

However, this approach ignores the problem of multiple applications on the user's system competing for network bandwidth. To minimize this problem the Mozilla browser prefetches documents sequentially, deferring the next prefetch until after the former has completed. While this is not an ideal solution, it minimizes the problem by reducing the prefetching application's demand on the operating system's networking subsystem.

Given that web pages are commonly composed of multiple HTML frames, the browser may see link prefetching directives from one or more of the frames. Therefore, the Mozilla browser does not begin prefetching until all frames, and any in-line content they contain, have finished loading. This approximately corresponds to the time at which the DOM's `onLoad` event handler for the HTML <BODY> tag would fire for the top-most frame in a frame hierarchy.

As stated above, the intent of link prefetching is to improve the user's perception of how quickly pages load. It is therefore paramount that link prefetching not interfere with normal user activity. Indeed, when the user clicks on a link, or when JavaScript executing on the page causes something new to be loaded, the browser must stop all link prefetching activity and respond to the new network request as soon as possible. For link prefetching this creates a situation in which the browser may be left with partially prefetched documents. Moreover, if the browser was utilizing a persistent connection for prefetching, the connection would have been closed.

Mozilla's support for partial cache entries and byte-range requests under HTTP/1.1 [8] to complete partial cache entries helps to minimize the impact of aborted prefetch attempts. Moreover, the serialization of prefetch requests minimizes the loss of persistent connections to at most one. This is important since browsers commonly keep at most two persistent connections open per origin server, as recommended in section 8.1.4 of RFC 2616 [8], and four per proxy server (which is how most common web browsers are configured).

As a result, if the link clicked by the user happens to be to the same host as the prefetch requests, chances

are good that there will be a persistent connection in the browser's idle connection list available for immediate use. In such cases, the loss of a persistent connection due to an aborted prefetch request is minimized. We assert that the cost is justified by the possibility of correctly pre-populating the browser's cache, even if only with a partial document.

4 Browser Imposed Restrictions

Given that link prefetching is purely a mechanism to pre-populate the browser's cache, the expiration time calculation for prefetched documents must still adhere to the HTTP caching rules as described in section 13.2 of RFC 2616 [8]. This means that the server should only direct the browser to prefetch documents that have a non-zero freshness lifetime. Since the browser must honor the server specified freshness lifetime, it is crucial that the server specify a reasonable interval of time within which the user can view the prefetched document without having to wait for the document to be validated or reloaded by the browser. As a result of this requirement, the Mozilla browser will abort a prefetch if it determines from the HTTP response headers that the resulting document would not be reusable from the cache without a round trip to the server.

The `<LINK rel="next">` prefetching directive creates an interesting problem for the browser. Given that this header is already in use on the Internet for the purposes of site navigation, the authors of the original content may not have considered the possibility of these documents being prefetched. Therefore, it is likely that documents referenced by this tag may not be reusable from the browser's cache. Sites with dynamic content would be excessively burdened by a browser attempting to prefetch references to dynamically generated content.¹ In an attempt to minimize this problem, the browser will not prefetch documents referenced by an URL with a query string when the URL appears in a `<LINK rel="next">` tag. While this is clearly a suboptimal solution, since certainly any URL could be

¹Bugzilla (<http://bugzilla.mozilla.org/>), the Mozilla bug tracking system, serves dynamically generated, non-cacheable bug reports via CGI with `<LINK rel="next">` references to the next bug report. The URLs include a query string.

served up with a zero freshness lifetime, this does catch a large number of instances where prefetching would be of no value. The same restriction is not applied to the "prefetch" relation.

The Mozilla browser further restricts prefetching to only `http://` URLs. It will not prefetch other protocols, with HTTPS being the significant protocol excluded. While HTTPS page load times are typically much higher than those for the same content delivered over raw HTTP, there are a number of compelling reasons not to prefetch HTTPS content. The main reason is that we do not want to load secure content without the user's explicit knowledge. The lock icon found in most browsers is an example of this philosophy. Moreover, for most browser implementations secure content is only stored in the browser's memory cache (for privacy reasons), which is typically very limited in size.

An issue that has been widely debated is whether or not the browser should be permitted to prefetch documents from a different domain. The Mozilla browser allows for this, and the main reason for doing so is for parity with existing *ad-hoc* prefetching mechanisms. Moreover, the ability of one site to cause content from another site to be prefetched is no more or less of a user privacy concern than loading hidden images from another domain or JavaScript's ability to cause content to be loaded without the user's knowledge. However, in light of this issue, the browser offers the user the option to disable prefetching just as it offers the user the option to disable JavaScript. This is of course a heavy-handed solution to the problem of cross-domain prefetching, and other browsers might choose to offer the user more fine-grained control.

Finally, there is the issue of recursion and recursion depth. We took a simplified approach here by limiting the browser to only recognize prefetch directives in documents loaded explicitly by the user. This means that prefetch directives included in the HTTP response headers of a prefetched document will not be followed by the browser until the browser is directed by the user to load the prefetched document. While there is utility for recursive prefetching, we felt that it was prudent to limit the depth of prefetching to one level for simplicity. We hope to investigate recursive prefetching for the browser in the future. This may be a particularly useful ap-

proach for proxies prefetching from origin servers.

5 Identifying Prefetch Requests

It is often valuable for servers to track the origin of requests. The `Referer` HTTP request header (see section 14.36 of RFC 2616 [8]) is commonly used for this purpose. For the server's benefit, the browser may wish to distinguish prefetch requests from normal user initiated requests. The Mozilla browser sends the following additional request header with every prefetch request:

```
X-moz: prefetch
```

Note that the name and format of this header is not finalized. In the future this header (or one serving the same purpose) may carry additional information about the context of the prefetch. For example, it may be valuable to distinguish whether or not the prefetch directive came from a document or from a HTTP header. This would allow an administrator of a website with public or user-provided HTML content to block prefetch directives originating from said content.

6 Practical Limitations

One of the most significant detractors of link prefetching is the potential cost in lost or misused bandwidth. This affects both users and servers. For servers the impact can be very high if enough browsers are hitting servers with prefetch requests for documents that the users never chooses to load. For users who pay for Internet access by the number of bytes transferred (or more commonly by the number of integral megabytes), link prefetching can potentially end up costing the user money. While the majority of Internet users in the United States pay a flat rate or are billed by the number of hours they spend online, this is not necessarily the case in other countries.

Because of these practical problems, our approach to link prefetching has been a decidedly conservative one. While other approaches involve the browser or personal web proxy blindly prefetching embedded anchor tags, we felt that such a technique would not be acceptable in practice. Moreover, by making the server responsible for choosing what the browser prefetches, and by giving servers context enough to recognize a prefetch request, we feel that servers can

properly limit the use of prefetching to instances where it is most beneficial.

7 Applications and Results

As a proof of concept we investigated some simple prefetching scenarios. The first involved a simple slide-show of images with each new page containing two new images. Page size was between 17Kb and 18Kb, and image size was between 10Kb and 20Kb. Each page included prefetch directives for the next page and its images. With a modest connection speed of 128Kb/sec (typical upstream transfer rate of an ADSL connection), each full page took on average 3 seconds to load over the network, and only 300 milliseconds to load out of the browser's cache. This seems to be a case where prefetching is likely to help significantly, given that the user might very well spend 3 seconds looking at each page.

A second usage case involved taking the results page of a typical WWW search engine query and modifying it manually to have the browser prefetch the top-ranked pages when loading the search results. This is something that a WWW search engine could easily implement.

One thing we noticed with this approach is that it is somewhat common for the top-level page of a site to be served up non-cacheable. This seems to be used as a mechanism of tracking page views or rotating advertisements. This problem combined with the fact that we were not recursively prefetching content made prefetching in this scenario much less beneficial. Sites could be better designed (e.g., by making use of HTTP/1.1 cache control mechanisms that would trigger a revalidation on every visit instead of a fresh download) to avoid this problem while still enabling the tracking of individual page hits.

8 Conclusion

The link prefetching mechanism described in this paper is admittedly simplistic in a number of ways. In spite of this, it has proven to be an effective solution for a number of interesting applications and generally works well. In some cases, we've seen dramatic performance gains as a result of prefetching.

When this feature was added to Mozilla (in version 1.2, released November 2002), we received a lot of

feedback, both positive and negative, from the web community. In most cases, the negative feedback came from folks concerned about increased load on their networks. However, the general response from the community was that a server-driven approach was the right approach because it puts the burden on servers to not misuse the network, and server administrators are generally cognizant of their network bandwidth utilization because of the associated monetary impact.

One of our immediate goals is to try to better inform the community about this mechanism and the implementations that exist. To this end, we have published a Mozilla Prefetching FAQ [4] on www.mozilla.org and there is an article on devedge.netscape.com [5] explaining how to make use of this link prefetching mechanism. The article includes a live example.

As for gauging the effectiveness of our link prefetching mechanism, it is a challenging problem, but we feel that there are opportunities for it where the gain is very apparent (e.g., Internet slide shows, login systems, etc.). It is likewise very apparent that it is inappropriate for other types of applications (such as whenever dynamic content is involved). Prefetching top-ranking search results may be an interesting application area; however, utility in the case of general websites is limited since many top-level pages of popular sites serve non-cacheable content.

As a future research direction, we hope to explore recursive prefetching, better browser-side heuristics for determining local network idleness, as well as mechanisms to order and prioritize link prefetching directives. The latter feature would potentially simplify the design of pages with many prefetch directives. As is, the order in which links are prefetched is unspecified, and indeed the order in which Mozilla prefetches documents may vary depending on a number of factors.

Acknowledgments

We would like to thank Prof. Brian D. Davison and Waldemar Horwat for their valuable feedback on this paper.

References

- [1] <http://pages.prodigy.net/guide/optimize/spec30.htm>.
- [2] Gagan Saksena, inventor; Netscape Communications Corp., assignee. User configurable prefetch control system for enabling clients to prefetch documents over a network server. US Patent 6,023,726. Jan. 1998.
- [3] Gagan Saksena, inventor; Netscape Communications Corp., assignee. System and method for creating pathfiles for use to predict patterns of web surfing. US Patent 6,055,572. Jan. 1998.
- [4] http://mozilla.org/projects/netlib/Link_Prefetching_FAQ.html.
- [5] <http://devedge.netscape.com/viewsource/2003/link-prefetching/>.
- [6] *RFC 2068: Hypertext Transfer Protocol, HTTP 1.1*, Jan 1997. <http://www.ietf.org/rfc/rfc2068.txt>.
- [7] *HTML 4.01 Specification*, Dec 1999. <http://www.w3.org/TR/html4/>.
- [8] *RFC 2616: Hypertext Transfer Protocol, HTTP 1.1*, Jun 1999. <http://www.ietf.org/rfc/rfc2616.txt>.
- [9] *XHTML 2.0 Working Draft*, May 2003. <http://www.w3.org/TR/xhtml2/>.
- [10] Jose Borges and Mark Levene. Data mining of user navigation patterns. In *WEBKDD*, pages 92–111, 1999.
- [11] Brian D. Davison. Predicting web actions from html content. In *Proceedings of the The Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02)*, pages 159–168, College Park, MD, June 2002.
- [12] D. Duchamp. Prefetching hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, Oct 1999.
- [13] Z. Jiang and L. Kleinrock. An adaptive network prefetch scheme. In *IEEE Journal on Selected Areas in Communications*, 16(3), pages 358–368, Apr 1998.
- [14] Tom M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *USENIX Symposium on Internet Technologies and Systems*, 1997.
- [15] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.
- [16] James E. Pitkow and Peter Pirolli. Mining longest repeating subsequences to predict world wide web surfing. In *USENIX Symposium on Internet Technologies and Systems*, 1999.