6.033 DESIGN PROJECT #1

Netscape Hypertext Supercharger

Morgan McGuire

morganm@mit.edu

Massachusetts Institute of Technology

March 19th, 1998

Submitted to:

Prof. Saltzer / Patrick Kwon

ABSTRACT. The Netscape Supercharger attempts to reduce latency between a user selecting a link in a Web browser and the display of a new page by speculatively prefetching hyper-linked documents. To avoid overloading the network or Web servers with prefetch data, the Supercharger does not prefetch images, binary executables, or other large files. It selects how many documents to prefetch based on reasonable estimates of server/network latency and bandwidth. To simplify the design, the Supercharger acts almost completely independently of the Web browser. The Interactions between the Web browser and Supercharger module are limited to: 1. The Web browser notifies the Supercharger when a new document has been displayed 2. The Web browser notifies the Supercharger when the user experiences a cache miss 3. The Supercharger may read the Web browser's currently displayed document 4. The Supercharger may write to the Web browser's cache. The shared cache and data are protected by mutexes that prevent one module from reading while another is writing.

1. INTRODUCTION

The *Hypertext Supercharger* attempts to reduce the delay in the Netscape Web browser between a user's selection of a hyperlink and the display of a new document. This is accomplished by prefetching Web pages that are referenced by the current document. The delay may be caused by low bandwidth or high traffic on the network route between the Web server and Web browser, a slow or overloaded Web server, or large startup time to servicing requests on the Web server.

It is desirable for the Hypertext Supercharger to successfully reduce the delay on many different types of network connections without compromising cache transparency, and without reducing overall network performance. The first is an important design consideration because users may be accessing the Web through connections as diverse as dial-up telephone lines and high-speed dedicated lines. Cache transparency requires the user seeing only cached/prefetched documents that are identical to the current copy on the Web server, with reasonable certainty. Overall network performance is a concern when the Web server being contacted is already overloaded, or when network traffic is already high. In these situations, it is important that the Supercharger not overwork already taxed resources. In particular, the design must be such that a large number of Superchargers can be active (on the order of millions of Netscape Web browser users) without a decrease in network performance due to the prefetching.

There is an additional, non-technical design requirement: the Supercharge should be designed as simply as possible without compromising the other design considerations. A simple, easy to implement design will reduce the cost and time of development and testing as well as the potential for bugs, unintended system interactions, and security holes.

To meet these considerations, we have designed a new Web browser that differs from the current version of the Netscape Web browser only by actively and speculatively maintaining the document cache, and by alerting the Supercharger module when the active document has changed. This active/speculative cache service is performed invisibly to the user, "in the background," by the Supercharger module. The Supercharger module is completely integrated into the Web browser from the user's point of view-- the distinction is at the design and source level, not the application level. The Supercharger, which can be disabled by the user through a simple dialog, analyzes the currently displayed document, recent network and server performance, and user habits, then prefetches hyper-linked documents. As with the current version of the Netscape Web browser, documents in the cache are checked for validity by examining the Time-To-Live and date information provided by the Hyper Text Transfer Protocol

(HTTP) when available, and by requesting a new HTTP header description of the document from the Web server. These cache strategies are recommended by the HTTP/1.1 specification.

The Supercharger is implemented as three modules: a *Manager* thread, a collection of *Fetch* threads, and a *Network Analysis* procedure. The Manager communicates with the Network Analysis procedure and the Web browser through synchronized shared memory and broadcast messages, and is responsible for dispatching Fetch threads to prefetch documents. The Fetch threads call the Network Analysis procedure immediately before terminating (*see Section 8, Figure 2*.) Only two kinds of messages need to be sent to the Supercharger by the Web browser: a USER-CACHE-MISS message that terminates all prefetching so the user request may proceed at maximum bandwidth, and a NEW-DOCUMENT message that indicates the Supercharger should begin prefetching documents.

There exist various definitions of key terms involved with networks and multithreading. To eliminate confusion, the 6.033 and HTTP compatible definitions will be used for the following terms throughout this paper: the Netscape Web browser and the Supercharger component are network *clients*; a Web server is a *server*; a *HTTP connection* provides a reliable, ordered channel for communication between a client and a server; a *thread* is a sequence of instructions that can share an address space with another thread; a *document* is any file that can be retrieved from a server and displayed; *latency* is the time delay observed by a client between when it sends a request and when it begins to receive a response; *bandwidth* is the number of bits per second a connection can transmit (limited by the lowest bandwidth link in a connection).

Section 2 of this paper gives a brief overview of relevant features of the HTTP/1.1 protocol and reasons for keeping the Supercharger entirely above this layer. Section 3 describes the Network Analysis algorithm that is used to dynamically tune the Prefetch algorithm presented in Section 4. Section 5 describes the algorithm used to maintain and verify the freshness of the document cache. Sections 6 and 7 give specific implementation notes. Section 8 analyzes the predicted performance of the Supercharger and makes some concluding remarks about optimizations and potential features.

2. HTTP/1.1 LAYER

As defined in RFC 2068, “Hypertext Transfer Protocol -- HTTP/1.1,” HTTP/1.1 is a high-level protocol for use by the World Wide Web. HTTP has important properties that make it safe for the Supercharger to prefetch documents. The GET method, by which a client requests a document from a server, does not side-effect the state of the server (RFC 2068 Section 9.1.1). This means the client can request a document multiple times without affecting the state of the

server. In practice, many servers violate this (server logs, counters), however it is violated with the understanding that the server is responsible for these changes, not the client. The HTTP protocol is stateless, so closing a connection without completing a request/response transaction will not adversely affect the client or server. Similarly, aside from the performance degradation, it is functionally acceptable to open a separate connection for each request/response pair and close it immediately afterwards.

Because HTTP provides a reliable, ordered connection, there is a wide base of installed servers, Netscape already has a HTTP code base, and it is safe to perform multiple fetches using HTTP, the Supercharger will be built entirely on top of the HTTP layer. Only the client side software will be modified; the network and server will not be explicitly aware of the Supercharger, nor will they need any modification. Alternatively, this can be considered an "end-to-end" decision: only at the highest level does the software have the ability to judge the end-to-end (request through response) network conditions, and at no lower level is it reasonable to impose new constraints on other protocols that may not require such constraints.

3. NETWORK ANALYSIS ALGORITHM

Because the Supercharger is above the HTTP layer and invisible to the network and server, they do not explicitly provide analysis information to it. The Supercharger must therefore derive network and server statistics to form a model of the network. Why is this necessary? If documents are returning slowly from a server, the Supercharger will attempt to prefetch documents from that server to hide this delay from the user. But if the server is functioning slowly because it is already overloaded, the Supercharger's prefetching will add to the problem and reduce overall performance. Similarly, if the prefetch requests exceed the client's available bandwidth, then the incoming documents will slow each other down, and the client may see a net decrease in performance when an actual user request for a page has to compete with a backlog of prefetches.

To avoid these situations, a model of the network is constructed and used by the Prefetch algorithm to determine when prefetching is advantageous. The Network Analysis algorithm is used to continually update this model.

As shown in Figure 1, the network model is a very simple picture of the network structure from the client's perspective. It is known that a link with fixed maximum bandwidth, b_c, connects the client to the network. This bandwidth limits all communication with the client. In extreme cases, this bottleneck may be the client's own network card. More frequently, this link is a telephone line to an Internet Service Provider (ISP) or other low-speed route to an Internet

backbone. Some route exists from that fixed bandwidth link to a Web server. For every server, this route may have a different bandwidth, b, and latency, l. At the end of this route is a server that has a delay between receiving a request for a document and actually serving the document. This delay may be variable based on server load and document location/size. For the purposes of the network model used by the Supercharger, a server can be modeled by the maximum bandwidth between the server and the client (possibly limited by b_c), and the total latency between the server and client (network latency + server latency). Given these numbers for a server, the Prefetch algorithm is able to predict how long it will take to retrieve a document from that server.

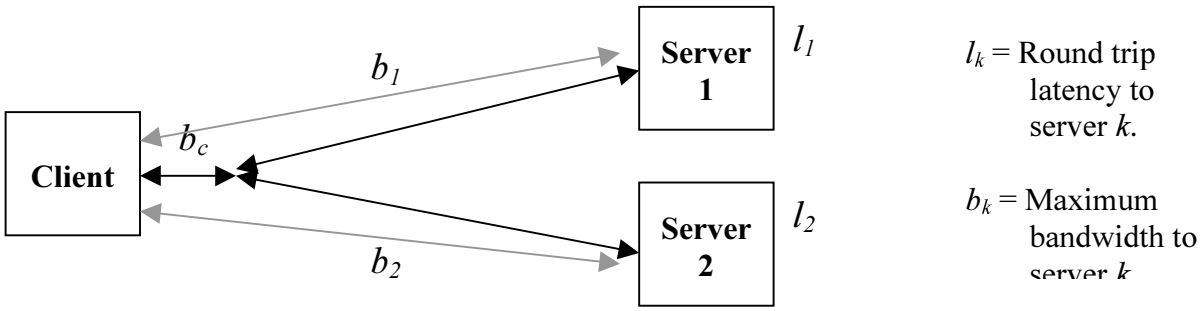

Figure 1: Network Model (from Client's Perspective)

The Network Analysis algorithm takes as input the latency and bandwidth observed by all HTTP transmissions performed by the Supercharger. It could also monitor browser transmissions, but it is simpler to avoid interacting with the browser code. The Network Analysis algorithm is state based, so it also uses its own old model of the network as input and continually refines this model. The model is stored in memory as a single number for b_c and a hash table of round trip latency and bandwidth for each server, indexed by server IP address. Every time the Supercharger is run (i.e. loaded and run as part of the browser starting up), the network model is initialized such that b_c is an arbitrary, small value (e.g. 1 bit/s). The hash table is initialized to be empty.

After each successful request/response transmission, the Network Analysis algorithm is called by a Fetch thread with the server's IP address, the observed round-trip latency, l, and bandwidth, b, (response size / (time elapsed since request − latency)). This information is stored in the hash table. An arbitrary upper limit on the number of entries (e.g. 10000) in the hash table is employed to prevent the table from consuming excessive memory. It is known that the slowest link between the client and server must have bandwidth at least b, so b_c is updated by the rule

$b_c \leftarrow$ maximum(b, b_c). Periodically, b_c should be reset to a low number (1 bit/s) and allowed to grow again to be responsive to changing local network conditions.

The hash table and b_c are made available to the Prefetch procedure via shared memory. Because the Network Analysis procedure is called by the Fetch threads and the Prefetch procedure blocks until the Fetch threads complete their tasks, there is no need to synchronize access to this shared memory. The Network Analysis procedure itself must be synchronized, however, so that no two Fetch threads can call it simultaneously.

4. PREFETCH ALGORITHM

The Prefetch algorithm used by the Supercharger uses the estimated Network performance, expected time until the next user input (mouse click), and a list of the documents referenced by the current document to determine which documents to prefetch from Web servers. These documents are inserted into the local document cache and managed by the Cache algorithm described in the next section. The Prefetch algorithm is implemented as a procedure the Manager thread calls whenever a new document is displayed (whenever the Manager receives a NEW-DOCUMENT message). The Prefetch procedure is responsible for creating Fetch threads to retrieve the documents that are likely to be loaded by the user next.

A good Prefetch algorithm should rarely fail to fetch the document that is actually chosen by the user. To accomplish this without overloading the network, servers, or consuming all of the available bandwidth (which must be shared with other applications on this client), the Prefetch algorithm needs to estimate which documents are too costly to prefetch, how long it will have to prefetch documents, and how long the documents will take to be fetched. Additionally, some documents are not safe to prefetch at all, and must be immediately eliminated from consideration.

To determine which documents can be potentially prefetched, the Manager examines all of the open documents for referenced documents, and gives a list of these referenced documents to the Prefetch procedure. All of the documents that are hyper-linked from a currently displayed document and do not have a valid version already in the cache are candidates for prefetching.

Images, binary executables, archives, PostScript data and other large files are probably not cost effective to prefetch. These files can take a very long time to fetch, and may overload the network and server capabilities very quickly. The Prefetch algorithm implements this by disallowing prefetching of any files that are not encoded using the MIME Text or HTML formats. Although it is often possible to determine the file type from the extension in the URL (e.g. “.jpg”, “.html”), this is not a failsafe method. As a heuristic, all URL’s ending in “.jpg”, “.gif”, “.zip”

and other registered non-text, non-html extensions will be immediately eliminated. Later in the procedure, a test against the actual document encoding will be used for a more exact decision.

Another set of documents can be quickly eliminated from consideration. Although it is safe to prefetch documents with the HTTP "GET" method, it is not safe to prefetch documents with the HTTP "POST" method (used for submitting user data in forms), nor is it safe to prefetch documents from URL's containing the "?" character. Such URL's cause servers to execute scripts and return the results. It is very likely that these scripts side-effect the server they are running on. Hence, all prefetching schemes should avoid URL's with the "?" character. Protocols other than HTTP should not be used for prefetching; for example, documents using the "mailto:" and "ftp:" protocols are not prefetched.

After these first culling steps, the Prefetch algorithm is left with a set of viable documents, some of which may turn out to be non-text, non-html files that can be eliminated from consideration. Each of these documents is on a server. Note that some of the documents may be on the same server. A new Fetch thread is spawned for each server. The Fetch thread retrieves the HTTP headers for the documents located on the server it has been assigned. Using a single thread to fetch multiple documents from a single server reduces overhead significantly (RFC 2068 Section 8.1.2). Note that the Fetch threads employ separate connections from the one the Web browser uses to retrieve documents, even if the documents are on the same server. This is to prevent slowing down the Web browser by closing its connection to the server. The Fetch threads measure the time between the request being sent and the beginning and end of the response returning. The Fetch thread calls the Network Analysis algorithm with this information, as well as the IP address of the server it is connected to and the size of the response it received (requests are of a relatively constant size, and only consume a few bytes). The Network Analysis program uses this information to update the network model.

Once the Fetch threads have returned the HTTP headers for the documents to the Prefetch procedure, it determines whether the document is appropriate to prefetch based on file type. Unlike the file extension, the HTTP header identifies the document type with reasonable reliability. If the header identifies the document as something other than HTML or Text, it is eliminated from consideration by the final step.

In the final phase, the Prefetch algorithm determines how many documents it can afford to prefetch, then assigns the Fetch threads to retrieve these documents. This is done by a simple and crude probability measure. Because the browser notifies the Manager thread whenever the displayed document has changed, the Manager is aware of the approximate time between user actions (mouse clicks). The Manager takes a simple average of the time elapsed between the last

two user interactions and provides this number, ΔT_U, to the Prefetch procedure as a simple estimate of the time until the next user interaction. From the network model, the Prefetch procedure has an estimate of the maximum bandwidth the client has available, b_c, and the bandwidth, b_k, and latency, l_k, between the client and each document's server S_k (where some of the servers are the same). The Fetch threads have reported the sizes of their assigned documents, D_k. Given N documents, the Prefetch algorithm selects documents in the following manner:

$$\Delta T_H = \text{time to fetch headers (already spent)}$$
$$\Delta T_k = \text{time to transmit document } k = D_k / b_k$$
$$D_{sk} = \text{aggregate size of all documents from server } k = \sum_{S_j = S_k} D_j$$

$$B_n = \text{relative bandwidth with first } n \text{ documents incoming} = \frac{\sum_{\substack{S_k \\ k \le n}}^{\text{unique}} D_{Sk} b_k}{\sum_{\substack{S_k \\ k \le n}}^{\text{unique}} D_{Sk} b_c}$$

$$\Delta T_n = \text{time to fetch the first } n \text{ documents} = \Delta T_H + \left(\sum_{k=1}^{n} \Delta T_k \right) B_n + \text{maximum}_{k \le n}(l_k)$$

if $\Delta T_N < \Delta T_U$ then
    Fetch All Documents
else
    Fetch the first n documents, where n is the largest integer satisfying
    $\Delta T_n < \Delta T_U$

Consider ΔT_n, the estimated time cost of fetching the first n documents. ΔT_H is the time that was already spent fetching the headers. ΔT_k is the time to for server S_k to transmit document k to the client, if the client is not receiving any other documents. This quantity is simply the quotient of the document size over bandwidth to that server. Note that the bandwidth to the server includes the bandwidth of the server (e.g. the server's disks and other limitations). $\Sigma \Delta T_k$ is the time required for all servers to transmit their documents, if the documents are transmitted serially. Maximum(l_k) is the worst case latency for communicating with any of the servers. Because the Fetch threads run concurrently, this latency is only paid once (assuming the time to fetch the documents is large enough to hide the latency of serially sending each request).

Documents from a single server are transmitted serially, however documents from multiple servers are transmitted concurrently. Because of this, $\Sigma \Delta T_k$ is an over-conservative estimate of the time for all documents to be transmitted. To obtain a more accurate estimate, it is multiplied by the relative bandwidth factor, B. B is the weighted ratio across all servers of the

bandwidth of each server, b_k, over the maximum shared bandwidth to the client, b_c. This takes into account the fact that multiple transmissions can occur concurrently, however b_c limits the bandwidth of all connections at any moment. For example, say Server 1 and Server 2 are both transmitting 1 MB documents at a bandwidth of .5 MB/min, and the maximum bandwidth into the client has been measured at 1MB/min:

$$D_1 = 1, D_2 = 1, b_1 = .5, b_2 = .5, b_c = 1$$

$$B = \frac{D_1 b_1 + D_2 b_2}{(D_1 + D_2) b_c} = \frac{.5 + .5}{2} = .5$$

The bandwidth adjustment is .5, indicating that it takes half as long for Server 1 and 2 to transmit their documents to the client if they transmit concurrently rather than serially. If Server 1 had been required to transmit both documents and Server 2 had no documents to transmit, then this factor would have been 1.0, indicating that Server 1 had to transmit the documents sequentially. If each server had a bandwidth of 1MB/min and transmitted one document then B would be 1.0 because the servers would be producing data at a higher rate than b_c, so b_c would become a bottleneck and limit overall bandwidth.

The Prefetch algorithm makes a simple decision to fetch the first n documents such that the time to fetch those documents, ΔT_n, is less than the expected time for the user to select a hyper-link. If ΔT_n and ΔT_U are reasonable estimates, then the Supercharger will not slow down the browser (if the Supercharger is prefetching when the user selects a page, the user's request has to compete with the prefetch connections), excessively tax servers by downloading more information than it can use, or overly stress the client's Internet connection. Because b_c is periodically reevaluated by resetting to a low value, the Supercharger is responsive to changes in local bandwidth, such as increased load on a shared line from other users or the user running other network software.

In summary, the Supercharger only prefetches documents that meet the following criterion:

- Referenced by at least one currently displayed document, where:
  - Reference is a hyper-link (`<A HREF="...">`)
  - Protocol of hyper link is HTTP ("http:")
  - No "?" symbols appear in the URL
- No valid version of the file is present in the cache
- Document is MIME encoded as Text or HTML

These documents will be prefetched, subject to the constraint that the Supercharger will only prefetch as many documents as it expects to receive before the user selects a hyper-link. The next section describes how the documents are maintained once received, and Section 6 describes how to stop requests if the user selects a link before the prefetch process is completed.

5. CACHE ALGORITHM

The cache is maintained as it is under the current version of the Netscape Web browser, where validation is performed according to the following process. Whenever a document is requested, the cache is checked and a request is sent to the server for a current header description of the file (using the HTTP "HEAD" method). If a cached version exists and the cached header matches the file size and last modified date from the new header, the cached document is served. The user may specify that cached documents need only be verified once per session or larger time period, rather than every time. Servers can optionally include information about how long a cached document should remain "live." After this time has expired, the cached document should be forced from the cache. If the cache size exceeds a user-specified limit, the oldest fetched (or prefetched) documents can be flushed. The browser interface should allow the user to flush the cache manually, as it currently does.

6. MULTITHREADING IMPLEMENTATION

This section clarifies some of the details of the implementation of the Supercharger and gives implementation notes for starting and stopping HTTP requests. A schematic of the Supercharger module and its interactions with the Web browser and cache is given in Figure 2. The Supercharger module includes the Manager and Fetch threads, the Prefetch and Network Analysis procedures, and the network model. All communication is through procedure calls (denoted by solid lines with arrowheads), messages, or shared data access (denoted by broken lines with arrow heads). Cache data access is protected by a write lock (one per cached file). To write to a cached file, a thread registers its intention to write to the file with the lock. When all currently executing reads are completed, the lock allows the thread to write to the file. To read a cached file, a thread registers its intention to read with the lock. If there are no writes pending, the thread is immediately allowed to begin reading, otherwise the read request is stalled until the write completes. A thread is blocked when it has registered a request to access the cache but has not yet been granted access. All other shared data are protected by simple mutexes. To read or write from shared data, a thread must first obtain a mutex. By convention, threads agree not to access the shared data when they do not have the mutex locked.

When the user launches the Web browser application, a Web browser thread for each open window and a single Supercharger thread for all windows are started. After the network model has been initialized (this write by the Manager not shown in the diagram), the system begins normal operation. Whenever a new document is requested by the user through any Window and successfully retrieved, the Manger is notified by a NEW-DOCUMENT message. As specified in Section 4,the Manager computes the expected time until the next user interaction ΔT_U, and calls the Prefetch procedure with this information and a list of all of the documents the currently displayed documents reference[1]. The Prefetch algorithm then determines which documents should be fetched.

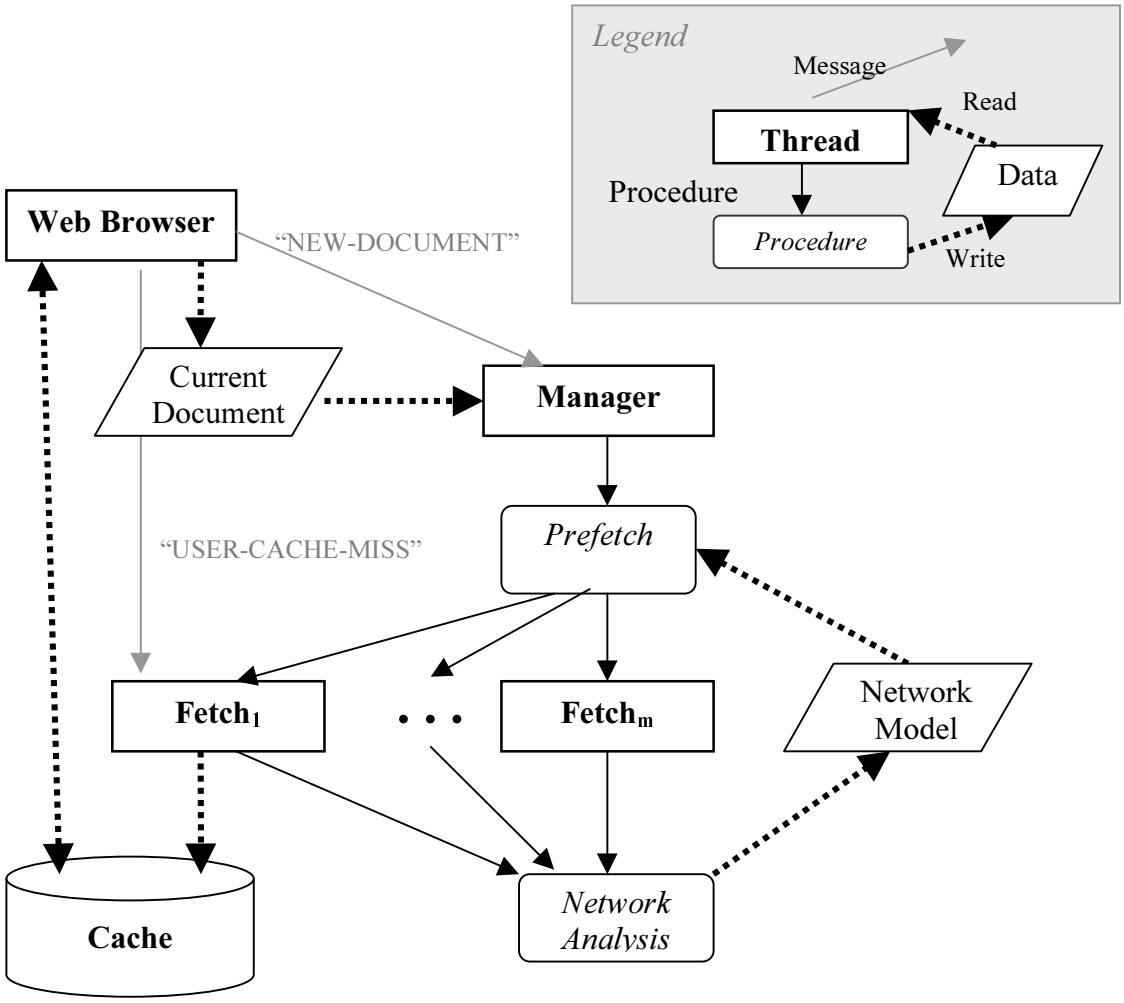

Figure 2: Supercharger Structure

[1] The Manager does not eliminate any documents from consideration, although it would be trivial for it to cull the images and mailto: links, for example. This is to keep all of the decision functionality in the

The Prefetch procedure starts multiple Fetch threads. Each Fetch thread maintains a persistent connection with a single server, for efficiency. First the Fetch threads are used to retrieve HTTP headers, then they are used to obtain the documents themselves. Whenever a Fetch thread completes a transaction (request/response pair), it calls the Network Analysis procedure to update the network model. The Network Analysis procedure is very short (see Section 3), so there should be no performance degradation from the requirement that access to this procedure is synchronized amongst the threads.

If a user requests a document that is not in the cache, a USER-CACHE-MISS message is sent to all of the Fetch threads (broadcast). If Fetch threads are still waiting for responses to prefetch requests, they all close their connections (allowed under HTTP) to provide the maximum bandwidth to the user request. When all of the Fetch threads have returned their responses, the Prefetch procedure terminates and returns control to the Manager thread. Some performance may be gained at the expense of design and scheduling simplicity by making the Fetch threads persistent, and communicating with them via messages. In this way, the overhead of creating a thread may be avoided. Additionally, if the new document and the old document both reference documents from the same server (a reasonable assumption), a persistent Fetch thread could maintain its HTTP connection to that server, and avoid the overhead of opening an HTTP connection.

7. USER INTERFACE CHANGES

A well-designed system provides clear and adequate feedback and information to its user, as well as providing the user with means of controlling the behavior of the system. The Supercharger enabled Netscape Web browser requires some interface changes to meet this requirement. A dialog must be added to enable/disable prefetching, and the user should be given some feedback as to how old a document is. The dialog can be added to the normal configuration menus for the browser, and requires only a simple check box. Displaying some measure of the actual performance gain attributed to using the Supercharger on this system would help the user decide whether to enable the Supercharger or not.

The age of a document can be conveyed graphically by dimming the Netscape icon proportionately to the age of the page, and texturally by appending the fetch date to the title bar of the browser window. As it does currently, the “Refresh” button should force a new version of the document to be retrieved from the server. According to the HTTP specification, any caching

---

Prefetch procedure, and use the Manager only as a gateway to the Web browser. This makes it easier to isolate and modify the Prefetch algorithm.

must report a warning to the user if it is not transparent-- that is, a dialog must appear if the document may be stale. Because documents are validated before removal from the cache, this is only necessary if it is impossible to validate or retrieve a new copy of the document from the server. Documentation of these features should be added to the on-line help for the product.

8. CONCLUSIONS

The best case scenario for the Supercharger is a single computer accessing large text Web pages over a slow link (such as a dial-up telephone connection to an ISP). In this situation, the client's bandwidth is unused while the user is reading a Web page, and it takes a long time to download pages. The Supercharger will take advantage of the low traffic across the slow link while the user is reading to prefetch pages. If the pages are mostly text, then the reader can immediately begin reading a new page after clicking on a hyper-link, although the images present on the page will still have to be downloaded. There are many worst-case scenarios for the Supercharger, however. One particular one motivates a desire for "Supercharger-aware" servers. If the client has a high speed Internet connection and is requesting pages from an overloaded server, then prefetching will likely worsen the load on the server, increasing the time until a request is serviced, not decreasing it. If many Supercharger browsers are accessing that server, the load and delay will be substantially greater than that under regular browsers. Under our design, the Supercharger's Prefetch algorithm would not attempt to download large files, and would reduce the number of pages it prefetches as it observed the bandwidth to the server falling and server latency rising (ΔT would grow while ΔT_U remained constant), however, the Supercharger would still impose more load than a normal browser. Introducing a prefetch notification into the HTTP protocol could alleviate this case. With such notification, clients could specify whether a request was a user- or prefetch-request. Supercharger-aware servers could then treat user-requests as higher priority, and possibly disallow prefetch-requests as they became heavily loaded.

It would also be advantageous if a greater number of servers returned Time-To-Live fields in the HTTP headers for documents. TTL is an optional field in HTTP that allows a server to specify how long a document can be cached. If servers provided reliable TTL numbers (1 min for a stock quote, 10 days for a company homepage, 6 months for a reference document), then local caching could be more efficient. Caches mainly need to be validated because these numbers are unreliable.

These server mechanisms are not part of the Supercharger design, but are examples of modifications that may emerge as a result of widespread use of a Supercharger-like product.