Rover Mosaic: E-mail Communication for a Full-Function Web Browser

by

Alan F. deLespinasse

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science
in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Author . . . .
Department of Electrical Engineering and Computer Science
May 26, 1995

Certified by . . . .
David K. Gifford
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Rover Mosaic: E-mail Communication for a Full-Function Web Browser

by

Alan F. deLespinasse

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1995, in partial fulfillment of the
requirements for the degrees of
Master of Engineering
and
Bachelor of Science
in Electrical Engineering and Computer Science

Abstract

The increasing availability and power of mobile computing devices are creating a demand for network applications which can accommodate slow or intermittently available network connections. A prototype system for browsing the World Wide Web on such a network was implemented, using caching, prefetching, and queued communications to hide communication latencies. Electronic mail was used as the underlying transport mechanism because of its ubiquitousness and fundamentally queued operation. A modified user interface based on NCSA Mosaic was designed to accommodate queued communications. The experimental system was found to be well-suited to the task, as well as to have significant advantages in the areas of reliability and user interface functionality.

Thesis Supervisor: David K. Gifford
Title: Professor of Electrical Engineering and Computer Science

Contents

Chapter 1

Introduction

The explosive growth of computer network connectivity in the past few years has led to a corresponding increase in the number of ways to use it. New applications allow users to find, retrieve and exchange information more quickly, easily and casually than was previously imagined possible. The most striking examples of such applications are those that provide access to the World Wide Web [4].

While the speed and spread of traditional hard-wired networks improve, a curiously opposite trend has also started: many users are trading their hardware's ability to communicate rapidly at any time for other features, such as physical mobility or decreased cost. These variations are a natural result of the ever increasing and diversifying user population, but many of the popular new communication standards are not flexible enough to accommodate all the different hardware configurations. For example, the World Wide Web's Hypertext Transfer Protocol [5] assumes that the hardware can support a high-speed TCP connection [13] to any Web server on the Internet at any time. A user of a laptop computer which is only occasionally plugged into its docking bay for data exchange cannot provide Web pages for access from other sites, and can only access the pages stored at other sites when docked.

The goal of this thesis is to explore the practicality of expanding the Web protocols to allow access from mobile clients, and to show how this might be done. A prototype system is described which uses caching, prefetching, and queued communications in an attempt to hide latencies and periods of network unavailability. All communications to and from the client machine are done through electronic mail because of its universal availability and it fundamentally queued delivery operation. Results of performance experiments and user evaluations are reported.

1.1 The Rover project

The problem of mobile connectivity is addressed in a general sense by MIT's Rover project [14]. Current research focuses on ways to more efficiently use communications channels which have very low bandwidth, and/or which may not be available (or have the same bandwidth) at all times. "Dockable" notebook computers and those with infrared, microwave or cellular phone connections fit this description. Rover attempts to optimize these channels by supplementing standard protocols with Queued Remote Procedure Calls (QRPC) and Dynamic Relocatable Objects.

Dynamic Relocatable Objects are data objects which can be copied from one host machine to another, bringing along procedures to perform actions on the data, in a manner similar to standard object-oriented design. This allows flexible distribution of work, and can often save a large amount of communication time. For example, a small CGI script [21] might easily produce a very large document to be downloaded via HTTP; it would be much more efficient, if possible, to move the script to the client machine and run it there. Another useful example would be to send highly compressed data along with a specialized decompression algorithm. In some cases, an object can be modified by these procedures and sent back to its origin, in which case there is a possibility of update conflicts; these can be resolved by application-specific conflict resolution procedures, which are also attached to the objects.

When objects are sent around, they may have to wait for a slow or temporarily unavailable connection. It is therefore not practical to wait for a reply to an object message after sending it; all communications must be asynchronous. This calls for a queue of outgoing messages on each host. Many messages can be added to a queue and forgotten about while other work is done; if and when a reply comes back, the current work may have to be suspended while the reply is dealt with. These messages have been named Queued Remote Procedure Calls (QRPC). In the case of applications which have until now called for communicating at interactive speeds, such as browsers for the World Wide Web, this may require some modification to the user interface (see Chapter 2).

This thesis can be thought of as a small part of the Rover project, addressing the specific problem of providing World Wide Web access to users of mobile or other computers which are only occasionally connected, or which have relatively slow connections. It provides the first test of the QRPC strategy, but does not make use of Dynamic Relocatable Objects.

1.2 Other related work

Traditionally[1], users of computers without direct connections to the Internet have had one crude, last-ditch method for accessing the World Wide Web. They can send the URL (Uniform Resource Locator) [6] of the document they want in an e-mail message to an automated gateway server in Switzerland, at address **agora@mail.w3.org**. This server will then download the document,

[1] The word "traditionally" is used in a very relative sense here, since the Web has only been in existence for about five years.

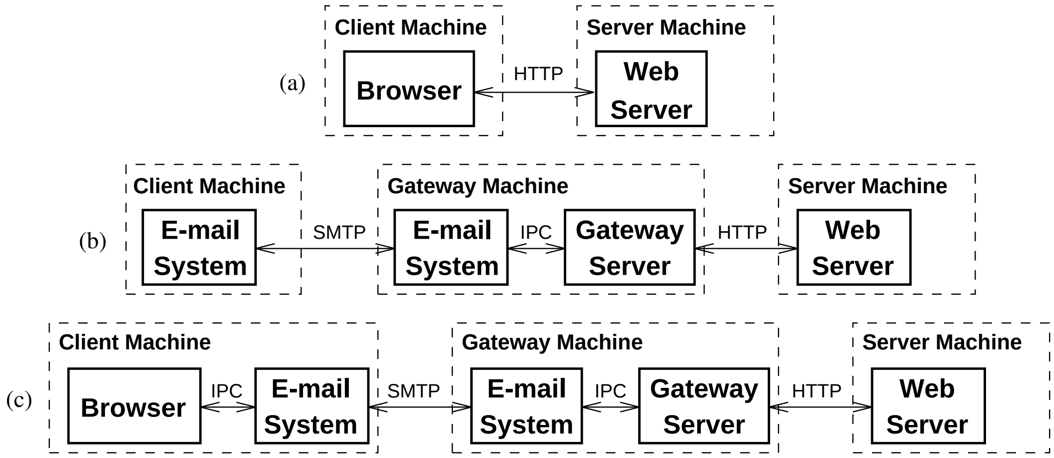

Figure 1-1: Different methods of accessing the World Wide Web. (a) Conventional method. (b) The Agora mail-server method. (c) New method. "IPC" is used as a generic name for all forms of Unix interprocess communication, including signals, pipes and program executions.

format it as clearly as possible in plain ASCII text (with hypertext links numbered), and send it as a reply message to the user. The user can follow links by replying in turn, with the number of the link to follow. This strategy, called Agora [31], is contrasted with the standard Web protocols (and the new strategy described in Chapter 2) in Figure 1-1.

The advantage of this method is that *anyone* can use it— e-mail is always the first application implemented on any network, and gateways are always provided, if possible, to the Internet, and hence to the above address. The disadvantages are obvious and severe. To begin with, this method is slow, especially for people who are not in Switzerland. This problem can be partially relieved by setting up many servers in strategic locations. Of course, portables without wireless links will still have to be docked before the messages can get through (and wireless links come with their own limitations on speed and geographical movement). Another problem is that the user must manually send e-mail each time a new document is desired, instead of simply following links by clicking on them, as in most popular Web browser applications. In addition, Web documents can contain text formatting information, pictures, video, sound, forms to be filled out, and many other kinds of data. All of these are filtered out by the Agora server, leaving only plain ASCII text that standard mail readers can cope with.

Recently, several researchers have devoted attention to the specific problem of accessing the Web from mobile clients. The Infopad project [18] treats a wireless machine as basically a dumb terminal, with all the work happening on a stationary workstation. The W4 application [1] adds very simple caching and prefetching of one screenfull of text at a time to its wireless palmtop interface. The Wit project, with its W* browser application [33], also uses a palmtop with wireless communication, with a much more sophisticated caching and prefetching method. It also uses lazy evaluation and futures techniques to hide latency; these give some of the same advantages as queued requests.

In a precursor to the Rover project, MIT researchers experimented with *dynamic documents* [15], a form of the Dynamic Relocatable Objects mentioned in the previous section. By combining these with simple caching and prefetching algorithms, they were able to make great improvements in the efficiency of network usage when accessing specially modified servers. The Mobisaic system [32] uses *active documents* to provide special services that are particularly useful on mobile clients, including pages that automatically change themselves based on the user's location.

The Coda file system [17][30] represented pioneering work in the use of prefetched caching and network scheduling to overcome slow or intermittently available network connections when accessing a file system. Ebling et al. [11] give a very concise overview of the issues involved in creating the Coda file system and other mobile network applications. The Little Work Project [12] and the Ficus [29] file system continued work in this direction.

The usefulness of electronic mail as a general-purpose transport mechanism was pointed out by researchers who were implementing massively parallel algorithms for factoring very large integers [19]. They used e-mail instead of more specialized protocols because of its ubiquitousness; they were not interested in its automatic queueing features.

A new experimental Web browser, DeckScape [8], was only brought to the author's attention after the work described in this thesis was completed. DeckScape's user interface offers background fetching and document list organization features that are strikingly similar, and in some ways superior, to those found in the browser created for this thesis. However, DeckScape is not designed to work in a mobile environment, and does not support a queued communications model. It also does not implement inlined images or HTML forms.

Chapter 2

Design

Like Agora, the experimental browsing system built for the Rover project uses e-mail to communicate with an automated gateway. However, as shown in part (c) of Figure 1-1, it adds a browser-style user interface to the front end of the mail system. This browser handles the sending and receiving of the e-mail communications and presents all types of data to the user in the same way as standard Web browsers. It thus solves all of the problems mentioned in reference to Agora except for speed. The speed issue is circumvented by a combination of queueing requests and caching pages.

2.1 Internals

All documents retrieved are stored in a cache in the client machine's local file system. If the user wishes to see a document that has fairly recently been saved in the cache, it can be displayed extremely quickly. To make requested documents more likely to be in the cache, certain ones can be prefetched, based on links from the ones explicitly requested already. The problem is how to determine when a cached copy of a document is old enough that it should be either reloaded or deleted to make room. A simple least-recently-used elimination algorithm would seem to be the obvious choice;

but in the case of HTTP-retrieved documents, additional information may be available which could lead to more efficient algorithms [5]. In some cases, a document actually comes with an explicit expiration date, before which it is (weakly) guaranteed not to change significantly. In most other cases, a "last-modified" date is specified; a relatively old one can be considered an indication of stability. There is one major problem, however: the results of HTML form submissions [9] have no natural "lifetime". For example, two documents retrieved by identical requests in rapid succession may be completely different, and it may be desirable to keep both for comparison, or there may be no reason to keep either at all. For this reason, combined with user interface considerations described in the next section, it was decided to let the cache be manually controlled by the user, rather than automatically filled and flushed. (One other option, recommended and used by CERN [20], is not to cache form results at all, but this would require a major departure from our highly successful policy of only displaying cached documents.)

When the document requested by the user is *not* in the cache, a request for it must be sent out via e-mail. These requests are queued (by the e-mail system) and delivered asynchronously to the gateway server, where replies are generated and sent, again asynchronously, back to the browser. They therefore represent a very simple form of Queued Remote Procedure Calls (QRPC), as mentioned in Chapter 1. The fact that any document (including prefetched ones not yet explicitly requested— see Chapter 3) may arrive at any time presents an interesting user interface design problem.

2.2 User interface

One possible design for the system's user interface would be to make it exactly like a standard Web browser, such as Mosaic [22] (see Figure 2-1): the user clicks on a hyperlink within a displayed document, and then

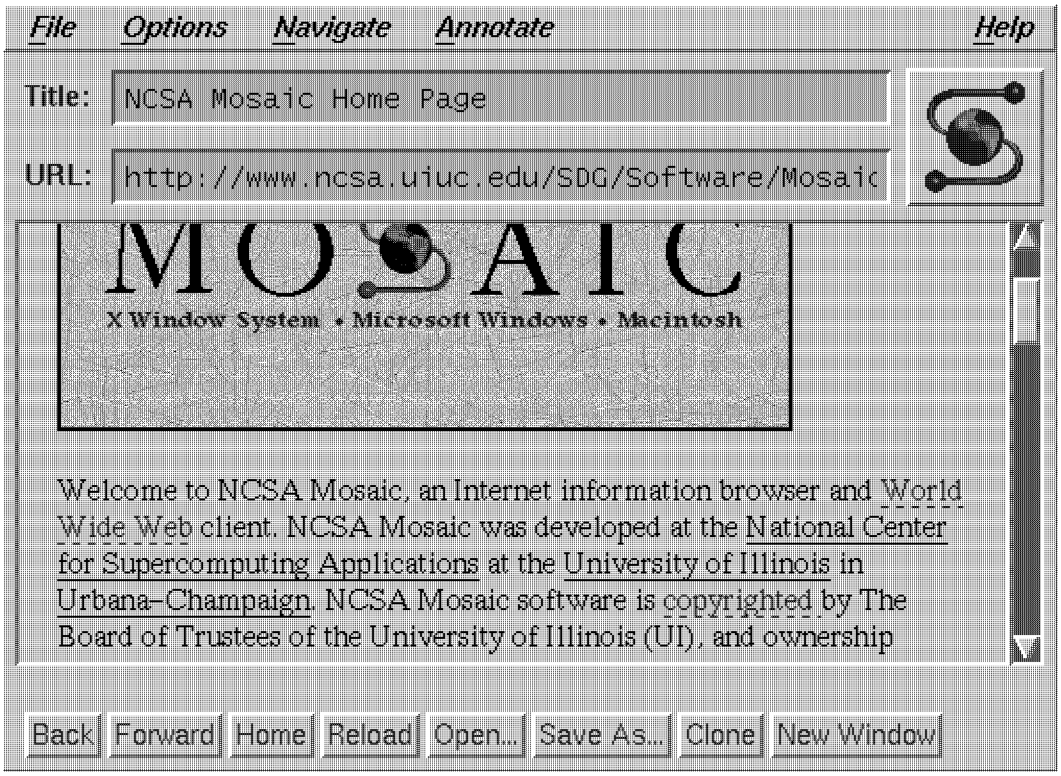

Figure 2-1: Main window of NCSA's Mosaic. Hyperlinks are underlined. The links with solid underlines have not been visited, while the ones with dashed underlines have.

waits while the document at the other "end" of that link is loaded from the server. This interface is fine for locally cached documents as well, but it is obviously unacceptable when a fetch request may take several hours to be serviced. Newer browsers, notably the commercial Netscape Navigator^{TM} package (Figure 2-2) [25], offer an improvement: the user can continue to scroll around in the old document while the new one is on its way. This is still not ideal. It would be much more useful if the user could browse around in all of the documents stored in the cache, plus request multiple additional non-cached documents— that is, after all, the whole point of queueing the requests. Alternatively, if documents are very slow in arriving, it would be nice to allow the user to shut down the application altogether, to allow the efficient use of other large applications or the termination of power to the

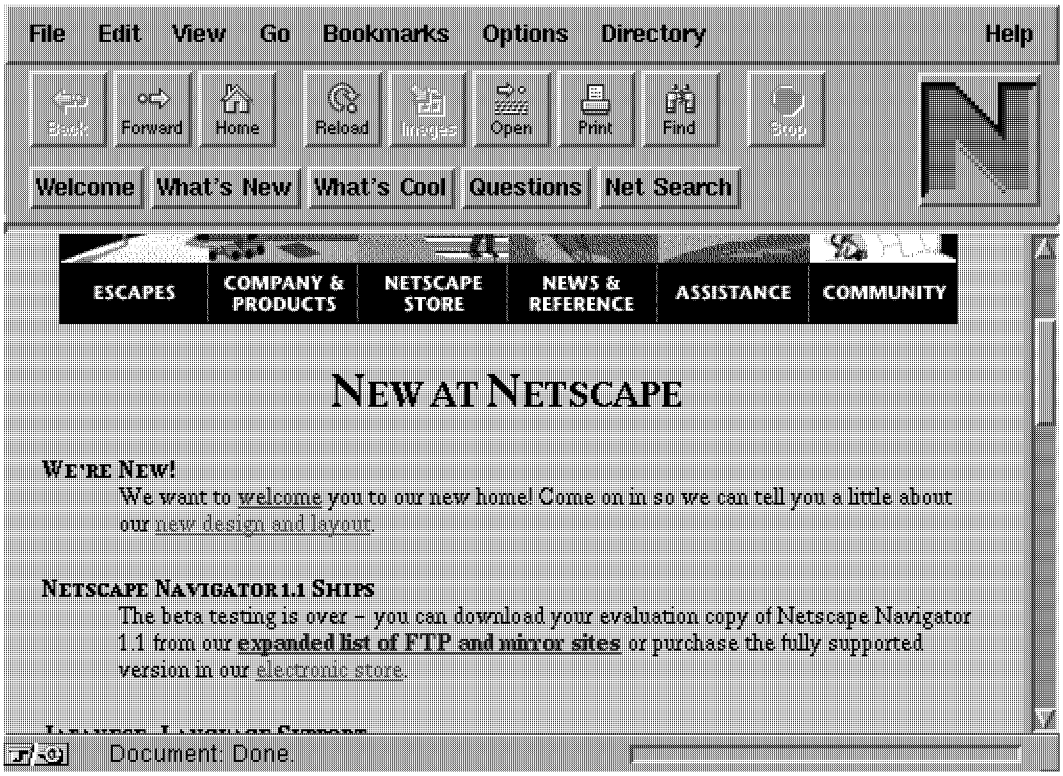

Figure 2-2: The Netscape Navigator^{TM} browser from Netscape Communications Corporation.

computer (an important consideration with portables).

The problem, then, is how to modify the user interface such that the user is made aware of newly arrived documents and can view them— or others previously cached— at any time. An initial thought was to automatically open a new window (perhaps iconified) for each arriving document. This could result in a sudden proliferation of windows, though, which might quickly overwhelm the screen and/or the user. It also does not provide for arrivals when the application is not running.

An even simpler design, which was partially implemented before it was rejected, would be to indicate a document's arrival by changing the color of the hyperlink leading to it, rather than Mosaic's usual practice of changing the color of links to documents which have already been viewed (see Figure 2-

Figure 2-3: Rover Mosaic's cache list window. The user interface is simplified using the **-kiosk** option added in Mosaic 2.5.

1). This method failed to adequately notify the user if the link in question was not visible when the page arrived, i.e., if the user was reading an unrelated page at the time. It also did not take into account documents which were requested by direct specification of the URL, rather than via a hyperlink[1].

It eventually became clear that the user would have to be shown a list of all requests made, along with the status of each request and a way to view the results of fulfilled ones. Given this, it was a small leap to add buttons to "delete" and "reload" each cached document, giving the user complete control over cache operations. The list was implemented as a dynamically updated HTML document, which could be shown in a browser window just like any

[1]The implementation of this design also had the more practical drawback of requiring modification of Mosaic's innards, which are an awful mess and ought to be left alone.

other, as shown in Figure 2-3. At first, to conserve screen space and memory, the list was considered the user's "home page", and it could be viewed in the browser's main window alternately with other pages. Eventually it became clear that even this was too restrictive, and a second window was added to display only the list.

Chapter 3

Implementation

The experimental system is shown in greater detail in Figure 3-1. It runs on a local network of Sun SPARCStations running SunOS and Intel Pentium PCs running the BSD/OS operating system [2], and should be easily portable to other variants of UNIX. It has been dubbed Rover Mosaic: Rover after the name of the larger project it is a part of, and Mosaic after the popular Web browser [22] used as its user interface. Two copies of Mosaic are actually run simultaneously; one displays the pages requested by the user, just as standard Mosaic does (Figure 2-1), while the other shows the cache list (Figure 2-3). (Mosaic is a rather large application to be running multiple copies of, but this is only an experimental system. A production system would hopefully be much more efficient.) Both copies are controlled by a

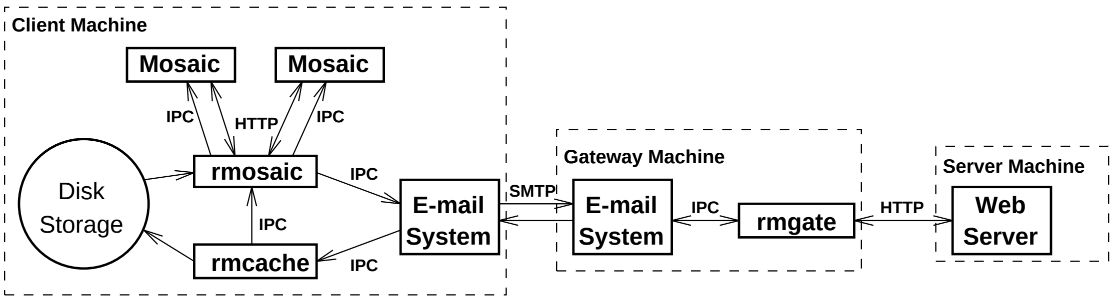

Figure 3-1: Detailed diagram of Rover Mosaic.

separate program, called **rmosaic**, which is really in control of the whole operation. It makes requests via e-mail to the gateway server program, **rm-gate**, which fetches documents and returns them as more e-mail messages. The **rmcache** program reads these messages, places the documents in the cache, and notifies **rmosaic** of the changes.

3.1 Rover Mosaic— the control program

Rover Mosaic is started by running the **rmosaic** program, which begins by spawning two instances of Mosaic version 2.5 for the X Window System, as provided by the NCSA [22]. Both are configured to obtain all documents through a proxy HTTP server on the same machine, and **rmosaic** serves as that server. It therefore has complete control over what is shown in both windows in response to any request. It can also force them to request specific documents at any time by sending "remote control" commands, which are implemented with Unix signals and files [23]. By these means **rmosaic** exercises a large degree of control over both copies of Mosaic, and can basically use them as an interface to its own functionality. A few of Mosaic's functions are completely beyond the control of external programs; these are generally handled by really ugly hacks, which are documented in the code's comments (see Appendix B for instructions for downloading code).

The first copy of Mosaic run is meant to behave just like standard Mosaic whenever possible, i.e., whenever the document the user asks to see is already in the cache. It looks exactly like any other instance of Mosaic, as shown in Figure 2-1. But when a requested document is not in the cache, **rmosaic** sets in motion a complex sequence of events which may overlap with other sequences like it, or with other events caused by the user. The first thing that happens is that an empty response (HTTP response 204 [5]) to the request is sent, telling Mosaic to continue on as it was in the previous document. (In fact, Mosaic does not quite correctly react to this part of the HTTP specification: it unobtrusively displays a rather enigmatic error message, “And silence filled the night,” before continuing.) The second thing that happens has to do with the cache list displayed in the other window.

The list stored internally by **rmosaic** contains an entry for each document requested from and/or returned by the gateway server. However, only the documents explicitly requested by the user are marked as "visible"; these are the entries displayed by the other copy of Mosaic (see Figure 2-3). Each entry contains the name of the document requested, plus the name of the document it was requested from, to aid the user’s memory. The names shown are the HTML-specified titles of the documents, if known; otherwise, the URL is used instead. Also shown are the time and date of the last operation on the entry (either the request or the arrival) and whether or not the request contained HTML form information (those that did are marked “query”). If the response to the request has not yet arrived from the gateway server, the entry is marked “pending”; otherwise, the entry is marked “viewed” or “not viewed,” depending on whether the user has looked at the document yet, and three hyperlinks are shown: “View,” “Reload” and “Delete.” The URLs attached to these links are actually special codes which tell **rmosaic** to take the appropriate action.

As soon as the empty response to the original request is sent, **rmosaic** adds a “pending” entry to the cache list and invokes the **sendmail** program (or whatever program is appropriate to send an e-mail message) to send the request on to the gateway server. The message contains the entire HTTP request, along with several pieces of auxiliary information (including the user’s name, the return address and instructions for inlined images and prefetching— see Section 3.2 and Appendices A and C). Then the second Mosaic is told to reload the updated cache list, and browsing continues as before.

3.2 The gateway server

On the gateway server machine, the **rmgate** program is configured to run every time a request message arrives, with the message sent to the program's standard input (see Appendix B). If the document is to be fetched by any protocol other than HTTP (FTP [27] or WAIS [16], for example), **rmgate** just runs CERN's line-mode browser, **www** [26], sends the results back to the client, and exits. HTTP- fetched documents are handled directly by **rmgate** for a variety of reasons.

First of all, **www** cannot handle HTML forms at all, so HTTP must be implemented by **rmgate**. Also, **www** is designed to be interactive, which means it only tries to make an HTTP connection for a short while before giving up. Since **rmgate** is part of a queued, background process, it can afford to be much more persistent, actually making it more reliable than conventional Web browsers.

If an HTTP-fetched document is in the HTML format, **rmgate** parses it for hyperlinks and inlined images as it creates the response message. Inlined images are immediately fetched and mailed back to the client; they will usually arrive before the HTML document, so that they can be displayed along with it. Linked documents may or may not be added to a list for prefetching after the parsing is done. This allows processing of prefetched documents to take place after the explicitly requested ones, so that the documents which are definitely wanted arrive first. Prefetching depends on two parameters which are sent from **rmosaic** as part of the request (Appendix C). The first is the number of levels of recursive prefetching to do: 0 means no prefetching at all; 1 means fetch all the documents directly linked to from the requested one; 2 means also fetch documents linked from those; and so on. The second parameter specifies a number of seconds. If the request message took longer than this to arrive at **rmgate** (according to a time-stamp field included with the message), then communications are assumed to be slow, so that prefetching is an important thing to do. Otherwise, no prefetching is done, to conserve cache space and communication bandwidth. The defaults are 1 level of prefetching if the message takes more than 3600 seconds (1 hour).

Of course, since the topography of the World Wide Web is practically unlimited, recursive prefetching means that a document may be referred to more than once during an invocation of **rmgate**. Also, it is common for one inlined image to be used many times within a single document, as an item list "bullet," for instance. For these reasons, a hash table is kept of all the documents fetched during a particular invocation of **rmgate**, and multiple references are ignored.

The response messages are sent by invoking **sendmail** or some such low-level e-mail program. There is a separate message for each document fetched. The document is encoded in MIME's base 64 format [7] to allow non-text files, such as pictures and audio. Several other pieces of information, such as the name of the user who made the request, are also included (see Appendix C for a full list), to help the cache filler program, **rmcache**, store the document correctly in the cache.

It should be recognized that more than one instance of **rmgate** may be active at once. They will have no interaction with each other; two simultaneous instances may both fetch the same document and never even notice. Such is not the case with the cache filler.

3.3 The cache filler

The cache filler program, **rmcache**, runs on the client machine alongside **rmosaic**; like **rmgate**, it is run every time a message arrives. But since multiple instances of **rmcache** must interact with a single copy of **rmosaic**, they are not as independent as instances of **rmgate**.

The mail messages are sent to a special e-mail address on the client machine, which is configured to pipe each message to **rmcache**. Data must be written to files in the user's home directory, and a signal must be sent to the user's **rmosaic** process, so **rmcache** must be run with superuser priviledges (see Appendix B).

The **rmcache** program begins by writing the document to a file. It then appends all other information necessary for the cache list (titles, time stamps, and so on) to a special file which serves as an input queue for **rmosaic**. Since this file may be simultaneously opened by several copies of **rmcache** plus **rmosaic**, it uses the advisory locking mechanism provided by BSD Unix [28, p. FLOCK(2)] to prevent race conditions. Once the file is written, a Unix signal is sent to **rmosaic** to notify it (unless **rmosaic** is no longer running, in which case **rmcache** can simply terminate; the file will be read automatically by **rmosaic** next time it is run). **Rmosaic** reacts by loading in the new information, deleting the file (taking care not to do so when it's locked), and updating its display of the cache list. The document can then be loaded from the first file created above to be viewed at any time. Documents which were never explicitly requested by the user (prefetched pages and inlined images) are marked as not "visible" in the cached list (see Section 3.1). This means they are not displayed in the cache list window. If an invisible document is ever explicitly requested by the user, its cache entry becomes visible when the document is displayed. (The cache entries of inlined documents never become visible.)

The presence of "invisible" cache entries creates a bit of a problem, in that the cache is supposed to be manually controlled by the user. If the user never happens to decide to view a particular prefetched document, it will never become visible, and the user will not have a chance to delete it. To prevent the unbounded growth of the cache, a simple garbage collection scheme is used, which removes any invisible cache entry which is not referred to via hyperlinks, directly or indirectly, by a visible one. This algorithm is run once every time **rmosaic** exits.

Chapter 4

Evaluation

This system can never reasonably hope to perform as quickly as a standard Web browser on similar hardware. To see why this is so, consider that the same process which is normally used to fetch a document (HTTP, FTP, or whatever) is here used in exactly the same manner, by the gateway server instead of the client. The gateway machine could, of course, be closer in terms of network topography to the server machine that the client machine is, making the fetching process faster; but since the additional e-mail transit uses essentially the same underlying transport mechanisms (usually TCP [13]) as the Web protocols, the total time taken will always be at least as long. Added to this are the inefficiencies caused by binary file encoding, parsing at various stages, and additional information to pass around.

The results of some idealized timing experiments are shown in Figure 4-1, comparing a mail-served gateway architecture (marked HTTP-SMTP) with direct HTTP, as normally used for Web access. The experiments were done on a local 10-megabit Ethernet network carrying no other significant traffic, with the client on a SPARCstation IPX, and the gateway and the server sharing a single SPARCstation 5. Highly simplified versions of the client and gateway programs were used, in order to eliminate any delays caused by inefficient implementation. The server was NCSA's HTTPD 1.3

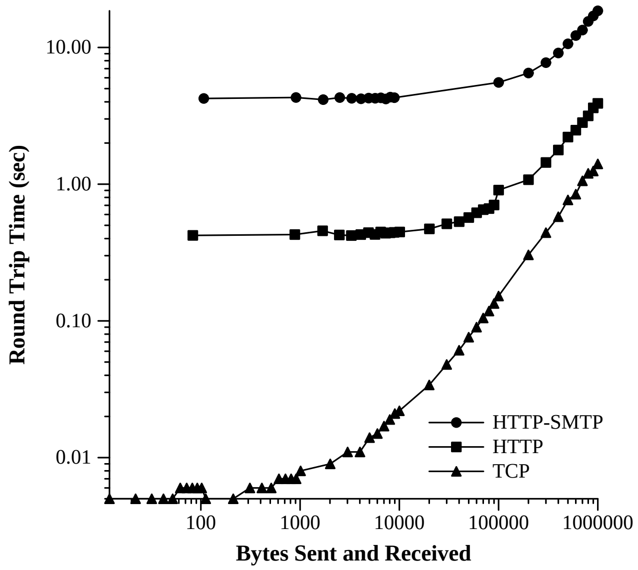

Figure 4-1: Idealized timing experiments.

[24]. To compute each data point, a request was made for a special URL containing the number of bytes to return. The server generated a document of the appropriate size by calling a CGI script [21]. The time was measured from the start of the initial request transmission to the end of the document's final receipt at the client. The simplified gateway program did not encode the document into MIME's base 64 format as the full implementation does (see Appendix C), so the diagram should accurately reflect a lower bound on the added overhead of a mail-served gateway. It is quite evident from the diagram that this overhead is substantial. (The third curve, marked "TCP", shows the times needed to transfer data over a simple raw TCP [13] connection. It is shown to indicate the limits on possible future optimizations of the protocols, assuming they do not use additional compression techniques.)

Of course, this system was not designed to be used when the standard methods are available. But, in spite of the loss of absolute performance, this architecture may have certain advantages, even for a traditional stationary, hardwired workstation environment. All time-consuming operations are performed by queued background operations, allowing the user to keep working. The act of loading a document into a viewer (when it is possible) is actually *faster* than with a traditional browser, since the document is cached locally. This could be particularly advantageous when dealing with very large amounts of data, such as video files: the communications could even be given very low priority (although such capability is not yet implemented in this system) and allowed to happen overnight.

This is also the only known fully-functional Web browser which allows the user to request several more documents while ones already requested are still being loaded. (DeckScape [8] cannot yet be considered fully-functional.) This can be thought of as "clicking ahead" of the incoming data stream, in the same way that keyboard buffers allow the user to "type ahead" of a slow user interface. The cache list makes it very easy to remember which documents have been requested, and of those which have been viewed. It can be used as a memory aid, in case the user gets distracted by intriguing links which are unrelated to the topic being researched (a very common occurrence).

As noted in Section 3.2, Rover Mosaic is much more persistent than standard browsers when trying to contact Web servers, and is therefore more reliable about returning requested pages. Regular users of Mosaic are all to familiar with the "unable to contact server" message which necessitates trying again; Rover Mosaic simply does all the repeated attempts automatically. Unfortunately, the improvement is reduced by certain bugs in Mosaic which seem to be only brought out by **rmosaic**, causing Mosaic to occasionally crash or freeze at apparently random times.

The system has been used and evaluated by several people, who have generally agreed that it is a reasonably good solution to the problem of mobile Web access. Their comments and suggestions have contributed heavily to the evolution of the interface (see Section 2.2), which now seems to be approaching the ideal goal of being almost as easy to use as standard Mosaic while offering increased functionality and mobility, although there still are (and always will be) plenty of opportunities for improvement[1].

[1]Admittedly, the people consulted for evaluations were not exactly a broad cross section of the population at large: they were all either students or professors of computer science at MIT.

Chapter 5

Conclusion

The Rover Mosaic system can be considered a success. It has shown itself to be a working solution to the problem of mobile World Wide Web access, and also provides certain other features not available in standard Web browsers.

The implemented system still contains a few bugs, mostly because of the clumsy adaptation of Mosaic to do things for which it was not originally designed. To be used in the real world, a much better-integrated system would have to be developed, preferably from scratch. The creation of such a system would be an excellent opportunity to incorporate functional improvements, as well. Here are some suggestions:

- Priority queueing
- Dynamic Relocatable Objects
- User interface features
- Bypassing the gateway

Priorities would be a very useful addition to the queued RPC model. For instance, as mentioned in Chapter 4, some very large documents could be given low priority and allowed to arrive overnight, while collections of small, tightly linked hypertext documents would benefit from fast delivery and should be given high priority. Research in this direction is already taking place for applications other than Web browsing as part of the Rover project.

Another Rover concept that might improve a Web browsing system is Dynamic Relocatable Objects. Scenarios in which these could be put to good use were described in the Introduction. Unfortunately, their effective use would require modification of existing Web servers, and would thus constitute a change in the architecture of the Web itself. Nevertheless, research is continuing on the topic, as the benefits could potentially be enormous.

Beyond that, many small improvements could be made to Rover Mosaic's user interface. Some users have stated that they would like to be able to choose which information was shown for each entry in the cache list: some would prefer to always see the URL of a document instead of its HTML-specified title; and various extra information, such as the dates of expiration and last modification, might be found useful. Also, user notification of document arrival is subtle at best; perhaps a small icon in the corner of the screen could change color and ring a bell, similarly to the familiar **xbiff** program, especially when **rmosaic** is not running.

Finally, since some mobile computers may at times be attached to standard full-speed networks, it would be nice to provide an option to bypass the e-mail and gateway server stages entirely, and simply use the standard Web protocols as they were originally designed. This would allow the user to enjoy the (usually) fast fetch latencies of standard Web clients, while also benefitting from Rover Mosaic's other advantages.

These advantages include background fetching, which minimizes the time spent waiting for documents to load; "clicking ahead" and explicit document-list control, which help the user to navigate through nonlinear hypertext geometries; and increased reliability in certain situations, as noted in Section 3.2 and Chapter 4. The only immediate disadvantage (aside from bugs and inefficiencies, which could be solved by rewriting the system) is the added complexity of the user interface, which may also lead to a shortage of screen real estate. However, there is one other possible problem which is harder to see.

One of the great features of the World Wide Web's hypertext structure, as seen by many advocates [3], is the style of publishing it encourages. Since links are very easy to create and quick to follow, any single page can be very small, to cover a specific subject at a particular level of detail. Further details and closely related topics can be in separate pages, connected by hyperlinks. This leads to a natural structure which mimics the structure of a particular field of knowledge, rather than being limited to sequential text in the manner of conventional printed articles. When designed well, hypertext documents can be much more effective tools for the transfer of knowledge than are printed documents.

Rover Mosaic has the potential to work against this benefit. Since its users cannot always follow links quickly, hypertext authors will now have an incentive to organize their writings into larger, more linear segments, rather than small documents with many links. This could represent a major step backward in the evolution of world connectivity.

It is the author's sincere hope that this will not happen— that authors will, in fact, completely ignore the needs of mobile users and continue to compose hypertext documents as if access to them could always be instantaneous. In the future, perhaps better ways around the delays can be found (for instance, a group of closely interconnected documents could be explicitly marked as a group, so that they could all be fetched at once if communication latency is high). For now, though, Rover Mosaic should be thought of, not as a regression in the progress of publishing technology, but as a small new addition to the growing number of options available to its users.

Appendix A

Rover Mosaic user's guide

This appendix is intended to help users who are already familiar with Mosaic and the basic concepts of the World Wide Web to get started using Rover Mosaic. If you have never used normal Mosaic, try it out before reading this. (On most systems, this means typing "Mosaic" or clicking on the NCSA Mosaic icon. Talk to your system administrator if you can't find it.)

Before running Rover Mosaic, you must create a subdirectory named ".rmosaic-cache" within your home directory. (Note: future versions of Rover Mosaic ought to do this automatically when first run.) Do this by typing "mkdir ~/.rmosaic-cache". Rover Mosaic stores cached documents in this directory.

To start Rover Mosaic, type "rmosaic". This will open two windows on your screen: one which looks pretty much like regular Mosaic (we'll call it the main window; see Figure A-1), and one which looks like Mosaic with a simplified user interface (which we'll call the cache window; see Figure A-2). The cache window will be empty the first time you run Rover Mosaic.

When you click on a hyperlink in the main window, one of two things will happen. If the page you're requesting happens to be cached on your machine, then the main window will display it, just like normal Mosaic. If it's not cached, then it won't be displayed; rather, a new *entry* will be added to the cache window, saying that the page has been requested. (Figure A-2 shows the cache window with several entries already in it.) This entry will at first be marked (pending), meaning that the page is on its way, but is still not in the cache. When the page arrives, the entry will change to display three hyperlinks, to View, Reload, and Delete the document, respectively. (This may take anywhere from a few seconds to several hours or even longer, depending on the size of the document and the speed of your network connection.) You may then view the page by clicking on the View link, or by clicking on the original hyperlink again.

The Delete link is used to remove the entry from the cache window and the page from the cache. You will want to do this frequently to keep the cache from getting too large. The Reload button is intended to request a fresh copy

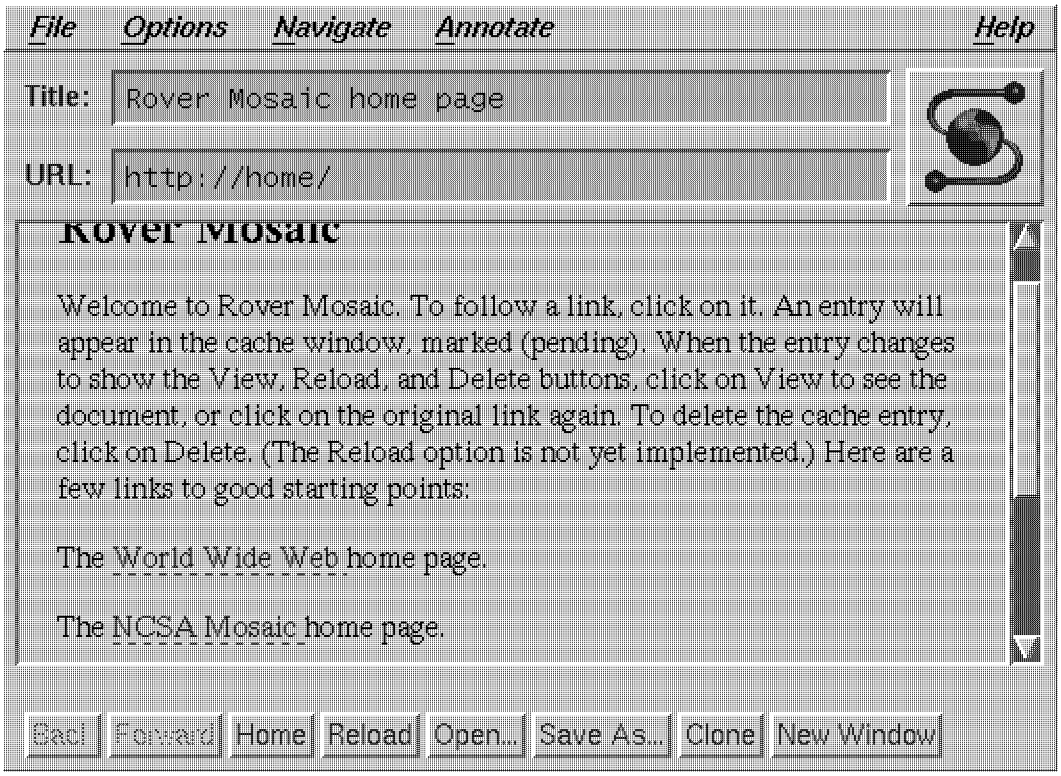

Figure A-1: Rover Mosaic's home page.

of the page, in case you suspect that it may have changed. This feature is not yet implemented, however.

A.1 Options

Rover Mosaic may be started with any of several command-line options to **rmosaic**. These are:

-clear Empties the cache before starting.

-d Causes **rmosaic** to spew large amounts of information about what it is doing. Meant for debugging purposes.

-force Forces **rmosaic** to run, even if it thinks another copy is already running. This may be necessary after a crash caused by one of Rover Mosaic's many bugs. If another copy of **rmosaic** *is* already running, this option may really mess things up.

-p *port* Makes Rover Mosaic use *port* as the port number for its local socket communications. Default is 8080.

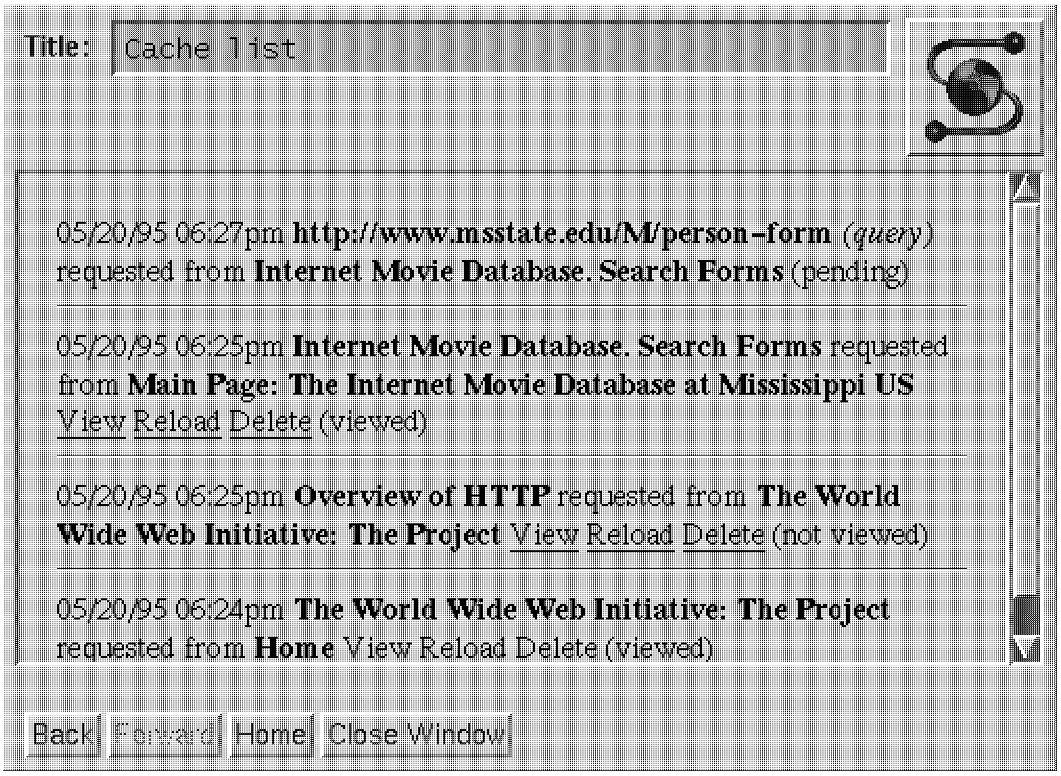

Figure A-2: The cache window.

-pf *level* Causes Rover Mosaic to perform prefetching of hyperlinks *level* levels deep. Default is 1.

-pft *time* Sets the minimum time limit for request transmission time, below which no prefetching will be done. Specified in seconds. Default is 3600 (1 hour).

The last two deserve some explanation. If the network is slow enough, Rover Mosaic may download into the cache certain pages which you haven't explicitly requested. Which pages to get are determined by following hyperlinks from the pages you *have* requested. If the **-pf** option is set to 0, this is not done at all; if it is 1, each page directly linked to from a page you request it retrieved; if it is 2, all the pages linked to from those pages are retrieved; and so on. These pages do not show up in the cache window unless you later decide to view them.

If the network connection is fast enough, the **-pf** option is ignored, and no prefetching is done at all. The **-pft** option sets the criteria for how fast the network must be to do this. The time in question is the delay between when an e-mail message is sent from your machine and when it arrives at the Rover Mosaic gateway server machine (e-mail is used for fetching all documents under Rover Mosaic). If you want prefetching to always occur, use **-pft 0**.

A.2 Environment variables

Two environment variables can be set to customize the behavior of Rover Mosaic (in addition to all the variables that standard Mosaic already recognizes— see NCSA's documentation [22]). Usually users need not worry about these, as system administrators should set the defaults to appropriate values.

WEBMAILCMD The command line for sending e-mail to the Rover Mosaic gateway server. This should be the command line for a program which, when run, will take its standard input to be a standard e-mail message, as formatted for SMTP [10], *including the headers*. This program should then send the message to an appropriate Rover Mosaic gateway server site. A typical value would be "/usr/lib/sendmail webserv@vienna.lcs.mit.edu".

WEBREPLY The e-mail address at your site for the gateway server to send fetched documents back to. Under normal circumstances, it should be completely unnecessary to set this variable.

Appendix B

Rover Mosaic administrator's guide

Administrating a Rover Mosaic installation might mean one or both of two things: running a client site for users, or running a gateway server for client sites to connect to. The necessary software for both comes together in a single package; after obtaining it, you can choose to install whichever parts you like. Compiling requires that the Berkeley Sockets library [28, p. SOCKET(2)] is supported. So far, we've mainly just run the whole system on an Intel Pentium processor running BSD/OS from Berkeley Software Design, Inc. [2]. It should port fairly easily to other systems. If there are any major problems, feel free to report them to **aldel@media.mit.edu**.

B.1 Downloading and compiling the source

The source package is available on the Web at **http://www.psrg.lcs.mit.edu/~aldel/thesis/rmosaic.tar.Z** (sorry, no FTP access is available). Put this by itself in a directory somewhere and unpack it. Edit the Makefile to set certain options; then run **make**. The default is to make everything; to make only the client side or gateway side executables, type "make rmosaic rmcache" or "make rmgate", respectively.

There are only three options to set in the Makefile. The first is simply to choose a C compiler; **gcc** works quite well for us. The second option sets the default e-mail command line (including the default gateway server to use) for **rmosaic**. You will want to change this if **sendmail** is not in **/usr/lib** on your system; if some program other than **sendmail** is to be used; or if you want to use a gateway server other than **vienna.lcs.mit.edu**. *Please* change the default gateway server if at all possible; **vienna** is one of our user workstations, and we would like to avoid loading it down. We recommend setting up your own gateway server, in which case you will want to use that instead of ours. The third option is the e-mail address for all incoming documents from the gateway to be sent to; it should be set to *NULL*, unless there is a user on the system named **webcache**, which strikes us a very unlikely. If there is, the WEBREPLY variable should be set to the full address you decide to use, including your client site's hostname.

If there are any problems compiling, you might want to try switching compilers; if that doesn't work, you may have to edit the source code. The top of **rmgate.c**, in particular, contains some definitions that are supposed to be provided by standard header files, but aren't on all systems; you might need to comment out or decomment certain lines to compile correctly.

B.2 Setting up a client site

A client site is an installation of Rover Mosaic which allows user to browse the World Wide Web via e-mail. In many cases, it will be installed on a portable machine, which means the administrator may be the only user. The following instructions describe installation for a more general multi-user scenario; it should work for a single user as well.

The executable **rmosaic** is the one that users actually run to use the system, so it should be placed somewhere in the standard path. The **rmcache** program need not be placed in the standard path, as it is never called by the user. Instead, it should be run automatically whenever e-mail arrives at the address for incoming documents (usually **webcache**— see above). This usually means putting a line in the **/etc/aliases** file, along the lines of "webcache: | /usr/local/rmosaic/webcache". The program referenced in this line should be an executable script that changes to an appropriate directory and runs **rmcache**, which may dump error reports to a file called **rmcache.errlog** in the directory. (You may want to periodically check this file to make sure your installation is working correctly, and erase it when it gets too large.)

Since **rmcache** must write to files in users' home directories, it is important that it is run with superuser privileges. Make sure its file permission flags include **setuid**, with the file owned by **root**. It is true that this program writes to a filename specified by the incoming mail message, but since the filename is not allowed to contain any path specification (see Appendix C), this should not create a security hole.

Since the **-kiosk** option, which simplifies Mosaic's interface, was not offered in earlier versions, you should make sure that Mosaic 2.5 or later is installed on your system. (If this is impossible for some reason, replace the string "-kiosk" in **rmosaic.c** with empty quotes.)

B.3 Setting up a gateway server

A gateway server should be set up on a machine which is permanently connected to the Internet. It responds automatically to requests contained in e-mail messages by downloading World Wide Web pages and sending them in reply e-mail messages.

Set up the **rmgate** program to be automatically run whenever e-mail arrives at a particular address on the system (typically **webserv@hostname**). The e-mail should be piped to **rmgate** as its standard input, and it may be necessary to include a few arguments on **rmgate**'s command line (see below). The current directory when **rmgate** is run is likely to end up containing a file named **rmgate.errlog**, which contains any error messages generated by the program. For this reason, it is generally a good idea to have the **/etc/aliases** file entry for **webserv** run a shell script which changes to an appropriate directory, then runs **rmgate** with the correct arguments. It is not necessary for **rmgate** to have root privileges.

By default, **rmgate** tries to send e-mail by running **sendmail**, with no path specified. You can either make sure that **sendmail** is in the execution search path when **rmgate** is run, or specify a different command (with path optionally included) as the first command-line argument to **rmgate**. The e-mail address to be sent to will be appended to this command as its final argument.

To fetch Web pages via any protocol other than HTTP, **rmgate** runs the CERN line mode browser, **www**, with command line "www -n -source" plus the URL of the page. If **www** is not in the search path, you can specify an alternative command as **rmgate**'s second argument. Make sure to use double quotes to make the entire command one argument if it contains any spaces.

Appendix C

Message formats

The e-mail messages passed from **rmosaic** to **rmgate** and back to **rmcache** are MIME-compliant messages [7] with special extended header fields. Most of these are common to messages sent in both directions, so they are all described first. Most of the fields specify either strings, which extend to the end of the line; integers, which are in decimal notation; or booleans, which are specified as either **yes** or **no**. Fields are optional and valid in both directions of communication unless otherwise stated.

X-Page The unique name that the page is known by. This is always in the form of a URL, and is *usually* the actual URL of the page, but may not be, especially when forms are involved. Must always be included in both directions.

X-Pagetitle The HTML-specified title of the page.

X-Source The unique name of the page that was shown when the request was made; generally assumed to be a page that contains a link to the current one in question.

X-Sourcetitle The HTML-specified title of the above source page.

X-URL The real URL of the page. Ignored by **rmgate**, since it is also specified in the message body. Defaults to be same as X-Page within **rmcache**.

X-Reply E-mail address to send fetched document to. Mandatory for request message; invalid in reply (although the reply should contain the same string in its **To:** field).

X-User The user name (on client system; must not contain the hostname) of the person requesting the page. Mandatory in both directions.

X-Path The name of the file to save the page in at the client. May not contain any path information (slashes) for security reasons. Optional, since **rmcache** will make up a new name if not specified.

X-Form Boolean. Tells whether or not the request contains query data (either as part of the URL after a question mark, or as an HTTP message body with the POST method). Default is **no**.

X-Time1 The time at which the original request was made. Specified in the typical Unix format of the number of seconds since midnight, January 1, 1970 GMT.

X-Time2 The time at which the page was fetched at the gateway. Same format as X-Time1. Valid only in reply.

X-Prefetch The number of levels of recursive prefetching to do (see Section 3.2). Default is 1. Valid only in request.

X-Preftime The minimum time limit, in seconds, on message latency before prefetching is turned on (see Section 3.2). Default is 3600 (1 hour). Valid only in request.

X-Inline Boolean. Whether or not to automatically fetch inlined images. Default is **yes**.

C.1 The request message

Exactly one e-mail message is sent for every page requested. This usually contains all of the header fields mentioned above except for Pagetitle and Time2. The message body contains the exact text, in untranslated ASCII, of the HTTP request message [5], including all of its headers and body.

C.2 The reply message

Exactly one e-mail message is sent for every page fetched. Since prefetching and inlined images result in fetching of pages which were not requested, this means that more reply messages than request messages may be sent. The headers should usually contain all of the above fields except Reply, Prefetch, Preftime, and Inline. (The current implementation also omits the Pagetitle field, which is filled in by **rmcache** later on. This is inefficient.) The omission of the Path field indicates that the document was not explicitly requested, and should therefore not be marked "visible" in the cache.

The body of the message contains the entire response from the fetch operation (including the HTTP headers, if HTTP was used). In the current implementation, the body is always encoded in MIME's base 64 format, to allow binary files. A more efficient implementation might allow text documents to be in a normal ASCII format.

Both the request and the reply could theoretically be broken up into smaller messages, using MIME's partial message facility; or conversely there could be several requests or replies within a single message, using the multipart message type. Neither of these features is currently implemented.

Bibliography

[1] J. Bartlett. Experience with a wireless World Wide Web client. In *Compcon '95*, 1995.

[2] Berkeley Software Design, Inc. Berkeley software design: Home page. http://www.bsdi.com/.

[3] T. Berners-Lee. Style guide for online hypertext. http://www.w3.org/hypertext/WWW/Provider/Style/Overview.html.

[4] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk, and A. Secret. The World Wide Web. *Communications of the ACM*, August 1994.

[5] T. Berners-Lee, R. T. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. *IETF HTTP Working Group Draft*, March 1995.

[6] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). *Internet RFC 1738*, December 1994.

[7] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. *Internet RFC 1521*, September 1993.

[8] M. Brown and R. Shillner. DeckScape: An experimental web browser. *DEC Systems Research Center Research Report 135a*, March 1995.

[9] D. W. Connolly. HyperText Markup Language (HTML): working and background materials. http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[10] David H. Crocker. Standard for the format of ARPA internet text messages. *Internet RFC 822*, August 1982.

[11] M. Ebling, L. Mummert, and D. Steere. Overcoming the network bottleneck in mobile computing. In *Workshop on Mobile Computing Systems and Applications*, 1994.

[12] L. B. Huston and P. Honeyman. Disconnected operation for AFS. In *USENIX Symposium on Mobile and Location-Independent Computing*, August 1993.

[13] Information Sciences Institute. Transmission Control Protocol: DARPA internet program protocol specification. *Internet RFC 793*, September 1981.

[14] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford, and F. Kaashoek. Rover: A toolkit for mobile information access. Unpublished draft, 1995.

[15] F. Kaashoek, T. Pinckney, and J Tauber. Dynamic documents: Mobile wireless access to the WWW. In *Workshop on Mobile Computing Systems and Applications*, 1994.

[16] B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. *Technical Report TMC-199, Thinking Machines, Inc.*, April 1991.

[17] J. J. Kistler and M. Satyanarayanan. Disconnected operation on the Coda file system. *ACM Transactions on Computer Systems*, October 1992.

[18] M. T. Le, F. Burghardt, S. Seshan, and J. Rabaey. Infonet: the networking infrastructure of Infopad. In *Compcon '95*, 1995.

[19] A. K. Lenstra and M. S. Manasse. Factoring by electronic mail. In *Advances in Cryptology: Proceedings of EUROCRYPT*, 1989.

[20] A. Luotonen, H. Frystyk, and T. Berners-Lee. CERN httpd. http://www.w3.org/hypertext/WWW/Daemon/Status.html.

[21] Rob McCool. Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/intro.html.

[22] National Center for Supercomputing Applications. NCSA Mosaic Home Page. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/help-about.html.

[23] National Center for Supercomputing Applications. Using Mosaic by remote control. http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/remote-control.html.

[24] NCSA httpd Development Team. NCSA httpd Overview. http://hoohoo.ncsa.uiuc.edu/docs/Overview.html.

[25] Netscape Communications Corporation. Welcome to Netscape. http://home.mcom.com/home/welcome.html.

[26] N. Pellow, T. Berners-Lee, and H. Frystyk. WWW line mode browser. http://www.w3.org/hypertext/WWW/LineMode/Status.html.

[27] J. Postel and J. Reynolds. File Transfer Protocol (FTP). *Internet RFC 959*, 1985.

[28] Regents of the University of California. *Unix Programmer's Reference Manual*. USENIX Association, April 1986.

[29] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the Ficus file system. In *USENIX Summer 1994 Technical Conference*, 1994.

[30] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operations in a mobile environment. In *USENIX Symposium on Mobile and Location-Independent Computing*, August 1993.

[31] A. Secret. Agora: Retrieving WWW documents through mail. http://www.w3.org/hypertext/WWW/Agora/Overview.html.

[32] G. M. Voelker and B. N. Bershad. Mobisaic: an information system for a mobile wireless computing environment. In *Workshop on Mobile Computing Systems and Applications*, 1994.

[33] T. Watson. Application design for wireless computing. In *Workshop on Mobile Computing Systems and Applications*, 1994.