To appear in Proceedings of SDNE '95 (June 5-6 Whistler, Canada)

Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach

Mika Liljeberg, Timo Alanko, Markku Kojo, Heimo Laamanen, and Kimmo Raatikainen
Department of Computer Science, University of Helsinki
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland

Abstract

Modern portable computers and wireless connections over cellular telephone networks have created a new platform for distributed information processing. We have designed a communication architecture that makes it possible to exploit the existing TCP/IP communication protocols but that also takes into account specific features of cellular links. Our communication architecture is based on the concept of indirect interaction. The mediating interceptor is the bridge between the worlds of wireless and wireline communication. It also provides enhanced functionality that improves fault-tolerance and performance. In this paper we demonstrate how the architecture is used to improve the performance of the WWW information browsing. Similar solution methods can be applied to other existing applications and protocols.

1 Introduction

Recent developments in mobile communication and personal computer technology have created a new and challenging platform for mobile information processing. Today the application of cellular telephone networks, such as the Global System for Mobile Communications (GSM) [12, 14], for data transfer between portable personal computers and traditional fixed data communication networks already makes mobile computing possible. However, cellular telephone links have a low and variable throughput. They are also prone to sudden disconnections and are often expensive to use. These characteristics create problems for many existing applications.

New solutions are needed to fully exploit the developing wireless data communication technology. In [9] we presented the Mowgli communication architecture designed to solve many of these issues. In this paper we give a concrete example how the proposed architecture is used to improve the performance of an existing application.

Information retrieval is predicted to be one of the most important applications for future mobile workstations. The World-Wide Web (WWW) [4] is a globally used distributed collaborative information system that has a rapidly increasing user community. Hence, there is real interest for bringing WWW to the mobile workstation. However, the HyperText Transfer Protocol (HTTP) [5] used by the World-Wide Web exhibits a number of performance problems which need to be solved to make HTTP a practical tool on a weakly connected mobile workstation.

In this paper we examine these problems in detail. We show how to address these problems within our architecture. Our goals are: 1) to improve the performance of the WWW browsing over a wireless cellular link without modifying the existing software and 2) to demonstrate the viability of the Mowgli architecture through this example application.

2 Mobile nodes and World-Wide Web

2.1 Hypertext documents

A hypertext document in WWW is a collection of objects containing various information elements such as text, graphical images and links to other (normally hypertext) objects. The representation of a hypertext object is specified using a special language, the HyperText Markup Language (HTML). It defines several kinds of basic markup as well as support for the hypertext links, inline images, and interactive forms. HTML is an application of the ISO Standard Generalized Markup Language (SGML) and its format has evolved spontaneously among the Internet community. Currently it is being prepared for the formal adoption as an Internet standard [3].

Hypertext links are represented in the form of *anchors*. An anchor is the start and/or the destination of a hypertext link. An anchor can either point to a whole document or to another anchor in a hypertext object marking a specific point. In this paper we consider, unless stated otherwise, a hypertext link to point to a whole document. The document pointed to need not be a hypertext document. However, the way other document formats (e.g. plain text or postscript) are treated depends entirely on the WWW client software.

2.2 HyperText Transfer Protocol

The HTTP protocol is a stateless protocol designed to transfer data objects between a WWW client and a WWW server. It is currently being standardized within the Internet community [5]. Each HTTP transaction consists of a single HTTP request and of the associated response. The request begins with a request line specifying a document object and an operation.

The response starts with a status line announcing the success or failure of the request. These are followed, in both the request and the response, by a header part and an optional body part. In practice, the body part of the request is usually empty. The body part encoding is adapted from the Multipurpose Internet Mail Extensions (MIME) [6, 11]. The HTTP headers are in a human readable format that closely resembles the standard for Internet mail messages [7]. Combined with the MIME headers they form the header part of the HTTP requests and responses.

The HTTP protocol defines several operations, called *methods*, that can be performed on objects. From our point of view, the most interesting one is the interrogative method GET, used to retrieve a specified object from the server. A similar method is HEAD, which returns just the header part of the response. The remaining operations are not interrogative, and in practice they are not very common. Typically they pass either information (POST, PUT) or requests (LINK, DELETE, etc.) to the server. It should also be noticed that current implementations do not necessarily support all of these, and some of them may even be disallowed by the administration.

2.3 Problems in using WWW over slow wireless links

Because HTTP requests and responses are in a human readable format, they can contain a relatively large amount of data, often more than one kilobyte. Much of this is not strictly necessary for the HTTP transaction. Document objects can also be very large and they are usually transferred without compression. In addition, HTML documents often contain pictures and other attachments, some of which can be prohibitively large for a retrieval over slow communication links. Hence, the sheer volume of the transferred data can create a severe performance problem when WWW is used over a slow link.

Another problem with WWW lies in the characteristics of the generated traffic. Browsing the World-Wide Web is typically a highly interactive activity: request-response interactions are interleaved with long periods of inactivity when the user reads the latest retrieved document. Such a usage pattern often amounts to a fairly low average link utilization. Since the call charges in most cellular telephone systems are time based, increasing efficiency and reducing the link open time can result in substantial savings. The problem can partly be alleviated by using asynchronous techniques, such as overlapping transfer and display, and by prefetching documents in the background. On the other hand, a mobile user connected through an expensive and slow link might prefer a disconnected mode of operation: the documents are first fetched into the mobile workstation, then the communication link is disconnected, and the user can read the retrieved documents without having to maintain the expensive connection.

A more subtle performance problem follows from the many round-trip delays involved in driving the stateless HTTP protocol on a connection-oriented transport protocol [8, 13, 15, 16]. These round-trip delays in turn increase the response times seen by the user. The slowdown is significant when a wireless link with a long round-trip time (RTT) is involved. For instance, using TCP over a GSM telephone network results in an RTT of about one second [1]. This implies that, from the user's point of view, each unnecessary round-trip can be considered harmful.

There are several sources of unnecessary round-trip delays. Current HTTP implementations are layered on top of TCP. A new TCP connection is created for each HTTP transaction, even subsequent requests to the same server are performed using separate TCP connections. The three-way handshake used in establishing a TCP connection adds a delay of one RTT before any data can be transmitted. Domain Name Server (DNS) lookups, often invoked by the client to find out the server's IP-address, add more round-trips. Yet more round-trip delays are due to the slow-start algorithm used in TCP and the small Maximum Transfer Unit (MTU) of wireless links. Because of the small MTU, a typical HTTP request is too large to be transmitted in one TCP segment. The slow-start algorithm is always used in the beginning of the connection. This means that the client end of TCP has to wait for the server end to acknowledge the first segment of the HTTP request before it can continue. During this RTT the link is idle, although the rest of the request probably could have been transferred in that time. The slow-start algorithm is applied separately for each direction of the TCP connection implying that the same problem also occurs for the HTTP response. Of course, the total slowdown depends on several factors including the RTT, MTU, bandwidth, and the amount of data.

A typical HTML document contains a number of small inline images, e.g. icons, buttons, and bullet marks. Retrieving such a document rapidly amounts to a considerable delay: each inline image is fetched from the server using a separate HTTP request, and each request adds several extra RTTs to the total time required to fetch that document. For example, if the size of an inline image is one kilobyte, the time needed to transfer it over a GSM link is roughly one second. This indicates that each extra RTT is approximately as long as the transfer time of the whole image.

3 The Mowgli communication architecture

The Mowgli communication architecture [9] is designed to increase the usability, reliability, and efficiency of client-server communication between *Mobile Nodes* (MNs) and fixed hosts. The basic idea is to split a channel with end-to-end control into two parts by using a store-and-forward -type interceptor. This interceptor, the *Mobile-Connection Host* (MCH), allows us to implement two separate but flexibly co-operating data communication subsystems: one wireline oriented, the other wireless oriented (see Figure 1). Recently two other systems based on a similar idea have also yielded promising results [2, 17].

In our architecture reliable communication between

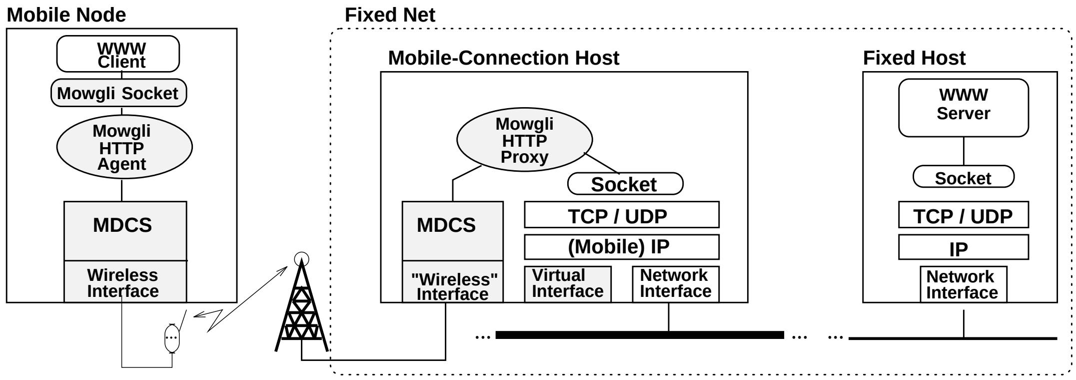

Figure 1: Overview of the Mowgli Communication Architecture

the MCH and the MN is provided by a specialized transport service, the Mowgli Data Channel Service (MDCS). It provides priorized data channels with flow control between MN and MCH. Existing TCP/IP protocols are used between the MCH and a fixed host so that the protocol software in the fixed net remains unmodified. Two mediators, the *Mowgli Agent* and the *Mowgli Proxy*, which reside at each end of the wireless link, are used to provide functionality similar to that of TCP and UDP (see Figure 1).

Applications on the MN connect to the agent through the Mowgli Socket Interface which is downwards compatible with the traditional BSD sockets. In their simplest form, the agent and the proxy serve as relays for the intervening traffic. However, by introducing specialized agents and proxies we enable the implementation of higher-level functionality: through intercepting the appropriate messages the agent and the proxy can optimize application-level protocols for a wireless environment.

4 Performance improvements

In this section we show how to use a specialized Mowgli HTTP Agent and a specialized Mowgli HTTP Proxy: 1) to remove the round trips which are not strictly necessary, 2) to reduce the volume of data transmitted over the wireless link, and 3) to support disconnected-mode operation. With the agent-proxy approach neither WWW clients nor servers need to be modified. We also discuss some possible enhancements to client and server software.

4.1 Decreasing round trips

Removing various extraneous round trips inherent in the HTTP protocol is fairly straightforward with the mediators. The Mowgli HTTP Agent and the Mowgli HTTP Proxy communicate by exchanging messages asynchronously over a long-lived Mowgli-connection. The agent intercepts requests from the local WWW client and forwards them over the wireless link to the proxy. Posing as a client the proxy, in turn, connects to the appropriate server for each request and asks it to perform the request. The proxy then forwards the WWW server's response back to the agent, which passes it on to the client. Since an end-to-end TCP connection is no longer established for each HTTP request, the round trip delays over the wireless link cease to be a performance factor.

The round-trip delays involved in retrieving inline images separately can be eliminated. The proxy can extract references to inline images while transferring an HTML page to the agent. It can begin retrieving the images in parallel without a specific request from the agent. When the transfer of the HTML page is complete, the proxy can immediately start transmitting the inline images.

The name service (DNS) queries needed to find out the IP-addresses of servers are another source of delays. The number of the queries can be reduced by using a special Mowgli DNS Agent. The proxy can co-operate with the DNS agent in order to prefetch addresses that are likely to be needed in the near future. A simple brute-force approach is to let the proxy look up all server domain names found in an HTML page and forward the IP-addresses to the DNS agent in the mobile node. A more sophisticated method capable of actually predicting user access is also possible but such an approach requires intimate knowledge of the user's patterns of behavior.

4.2 Reducing the transfer volume

There are two primary ways to reduce the transmission volume over the wireless link: data compression and caching. These two techniques work well together since documents can be stored in the cache in compressed form which in turn saves precious memory and disk space on the mobile node.

To reduce the HTTP protocol overhead the agent and the proxy can use a compressed version of the protocol headers when forwarding requests and responses between them. Transferred data can be compressed in several ways. Especially HTML, postscript, and plaintext documents are good candidates for compression. Content-type specific compression algorithms can be used for formats which are not susceptible to generic compression algorithms (audio data, for example). Lossy compression is also an option in some cases. For instance, the resolution or the number of colors in a large image might be reduced before sending it over the slow link.

A fairly drastic form of lossy compression is to drop some document parts altogether. For instance, many existing WWW browsers offer the choice of reading an HTML page without inline images. A more flexible solution is to let the Mowgli HTTP Proxy act as an "intelligent" filter. For example, if the size of an image exceeds a user-specified threshold, the proxy replaces it with a suitable default image.

Although data compression can yield significant benefits, it may not be necessary or beneficial to compress everything by default. If the amount of data to be compressed is very small, the compressed result can even be larger than the original. In addition, compression algorithms can consume a lot of CPU and memory resources. If these resources are scarce, the time required for compression may grow to dominate the transfer time. This may happen, if the MCH becomes overloaded or if the mobile node happens to be something like a PDA.

A cache in the mobile node is an obvious solution when transfers over the wireless link are not desired. Hence, the agent stores all incoming objects into its local cache. When the agent gets a request from a WWW client, it first checks whether the desired object already is in the cache. If this is not the case, the agent co-operates with the proxy in order to fetch the object and the possibly associated inline images.

4.3 Asynchronous operation

The cache managed by the agent can also be used to implement asynchronous operations, such as prefetch and background transfer.

For instance, the user can instruct the agent to start fetching a document into the cache as a background task. The user is then free to continue with the currently displayed page and to start additional background transfers. When a document has been completely transferred, any subsequent requests for it are serviced from the cache. A similar technique can be applied to inline images. Instead of waiting for all of the inline images to arrive, the agent responds to the client's requests with a default image (e.g. a picture with the text "Fetching image..."). A later reference to the same document retrieves it from the cache with inline images included.

Document prefetch is another background operation which can potentially yield great benefits in terms of the response times perceived by the user. Technically the procedure is similar to the other background operations. The main difference is that the user does not explicitly have to point out which documents to fetch. The agent and the proxy make these decisions based on their observations about user behaviour. Assuming a sufficiently high hit ratio prefetching is potentially more effective than user-initiated background transfer since the slow communication link can be fully utilized all the time. Of course, the main difficulty is the design of an intelligent prefetch algorithm. It is unlikely that a single algorithm can serve the varying needs of different users. However, assuming that only unused capacity of the wireless link is allocated to prefetch operations, any algorithm with a non-zero hit ratio is better than no prefetch at all. In fact, surprisingly high hit ratios can be achieved with modern AI technology [10]. When the prefetch policy is designed, we should not significantly increase the traffic in the fixed network.

4.4 Disconnected-mode support

Because the cost of maintaining a connection over a cellular telephone link can be high, it is important to support disconnected-mode operation. Disconnected-mode support for WWW can be achieved transparently with the help of the agent-maintained cache. However, current WWW clients typically want to check the validity of each document as it is accessed. It is obvious that cache validation cannot be done in disconnected mode. Therefore, in disconnected mode the agent allows the use of non-validated documents. Since, in connected mode, validating a document over the slow link causes a noticeable delay, we tend to think that automatic cache validation should not occur even then. Cached documents should only be validated if this is explicitly requested by the user. Many documents seldom change, and in many cases the user does not care if he or she has the latest version.

The agent co-operates with the Mowgli DNS Agent to insure that the server IP-addresses of cached documents are also retained. Another important point is that the cache should be made persistent so that the mobile node can be shut down without losing the cache. This allows the user to hoard a sufficient number of documents for extended disconnected periods.

A large hypertext document can consist of a number of subdocuments. Currently there is no satisfactory way to distinguish between hypertext links that refer to an integral part of the document as opposed to links that refer to peripheral and related information. This becomes a problem if the user, in preparation of going into disconnected mode, wants to retrieve all parts of a multipart document into the mobile node. For example, the time tables of the Finnish railroad company are available on the World-Wide Web. A traveler might wish to download them into his or her laptop before embarking on a prolonged trip.

A useful feature would be a user interface for controlling the cache management. The mobile user will need a way to control what will be retained in the cache. For instance, the user may wish to protect some documents from automatic deletion in order to ensure that they will be later available. On the other hand, the user may wish to delete unnecessary documents in order to free up disk space. Other control parameters might include a selection of different cache validation and discard policies and prefetch algorithms.

Special support is also needed for those HTTP methods that transfer information towards the server. Although these methods are not widely used, we anticipate that in the future uploading files as well as sending mail messages and news articles are likely to be supported on top of the HTTP protocol. Since these actions are not usually time critical it makes sense to group several files and messages together to be transferred at a later time. Instead of establishing a communication link for each separate message, it is cheaper to upload messages periodically. While most current WWW browsers handle electronic mail and news using separate protocols (SMTP and NNTP), the Mowgli architecture can be used to optimize these as well. We already have a prototype of enhanced SMTP mail.

5 Other considerations on mobile WWW

In the preceding sections we have paid special attention to optimizing the visible performance of World-Wide Web browsing over a slow wireless link. However, the possibilities of improving performance transparently are limited. A slow communication link remains slow no matter how efficiently it is used. Therefore, it is equally important to consider approaches that are directed towards helping the user to cope with the limitations of the communication media.

Let us, for instance, imagine a user who discovers an interesting hypertext link and follows it only to receive six megabytes of compressed postscript over the slow wireless link. At a speed of one kilobyte per second the transfer would take about one hour and forty minutes. Even if the user has enough patience, the batteries may run out. Of course the user can cancel the transfer at any time. However, it seems clear that the user should have enough information to make the correct decision in the first place. This implies that hypertext links should contain some additional information about the objects they point to.

In a more general sense, mobile and weakly connected workstations should be taken into account when designing WWW pages. While it would be unrealistic to expect all WWW authors to cater to the mobile community, it is likely that the future will bring new WWW services specifically targeted at mobile users. These services will create a need to add new mobility-related features to the HTTP protocol and the HTML format. Meanwhile, the Mowgli architecture provides a convenient way to transparently generate some of the information needed by mobile users and insert it into conventional WWW documents.

Attributes that would be useful with hypertext links include at least: 1) document size, 2) expiration time, 3) a list of alternative presentation formats and their sizes. The document size provides a general estimate how long it takes to transfer and process a document. It allows the user to make decisions about which links to follow. It can also be a useful hint for a prefetch algorithm to skip large documents in favor of smaller ones. Expiration time gives some indication about the frequency at which the document changes. This can be of interest, for instance, when deciding whether the document is worth to be retained for disconnected-mode operation. A list of alternative presentation formats and document sizes in the hypertext link would allow the user or the client program to select the format best suited for the current needs. Presentation format negotiation is currently possible but the server makes the final decision. The client has no way to find out which formats are actually available. The client can only tell the server which formats are suitable. Because the client is better informed than the server, we argue that it should be allowed to make this decision.

Another “nice” attribute of a hypertext link could be the relative “importance” of the link as compared to other links on the same page. By the relative importance we mean an estimate of the likelihood that the user will be more interested in the linked document than in the other documents accessible through the current document. If present, the relative importance of a link is a strong hint for prefetching.

It should be obvious that maintaining the described attributes by hand for each hypertext link would prove an impossible chore for WWW authors. Therefore, automatic methods for generating and maintaining such information are needed. It should also be noted that some of these attribute values can not always be known in advance. For instance, WWW servers can dynamically generate documents. To find out the exact size it is necessary to generate the actual presentation. On the other hand, for the purposes stated in this paper, a reasonable estimate is quite sufficient.

The Mowgli HTTP Proxy can, given sufficient network resources on the fixed net, modify transferred HTML documents to include the size and expiration information to each hypertext link. Unfortunately, the proxy is forced to use the HTTP protocol's HEAD method to fetch the header information for each referred document. This can add up to a lot of network traffic and take longer than the user is willing to wait. The delay could be eliminated by fetching the headers in the background as we did with the inline images. A document that is referred for the first time is displayed unmodified. While the user is reading, the copy in the agent's cache is modified to include the additional information. A subsequent reload of the same document displays it with the modified hypertext links. However, the additional network traffic in the fixed network would still be a problem. It would be better if WWW servers themselves actively maintained the links by exchanging information with other servers periodically. A reasonable compromise between modifying all WWW servers in the world and having the Mowgli HTTP Proxy do all the work might be to install a number of WWW proxy servers with enhanced mobility features within the fixed network. Based on internal statistics such servers could even produce prefetch hints for mediators like the Mowgli HTTP Proxy.

6 Implementation issues

We have implemented a prototype Mowgli HTTP Agent and Proxy to test some of the suggested improvements. We use a Unix (Linux) platform at both ends of the wireless link. The current implementation supports round-trip removal, inline image prefetch, background transfer of documents and images, and persistent caching both in the agent and in the proxy.

So far, we have not implemented data compression or binary encoding of HTTP headers as the effects are easy to predict.

6.1 General considerations

The Mowgli HTTP Agent intercepts the traffic generated by the WWW client on the mobile node. It communicates with the Mowgli HTTP Proxy in the Mobile-Connection Host using reliable data channels provided by the Mowgli Data Channel Service. The MDCS layer takes care of link disconnections and reconnections. Therefore, the agent and the proxy can concentrate on minimizing the intervening traffic.

We have paid special attention to resource consumption and performance. Typically only one WWW client is running on a mobile node. However, if there are several WWW clients running concurrently, the same Mowgli HTTP Agent handles all of their HTTP requests. Similarly, only one Mowgli HTTP Proxy is running on the MCH and it is able to serve multiple agents on different mobile nodes. Having a single agent and proxy saves system resources at both ends of the wireless link and it also simplifies the cache management.

The support for pre-emptive priorization of data channels in the MDCS can be put to good use in our HTTP mediators. We allocate separate data channels for requests and responses. Data channels with low priority are used for the background transfers so that they will not interfere with the normal operation. For the inline images we allocate a data channel with lower priority than for the associated hypertext document. A separate channel is allocated to transfer control information and configuration parameters between the agent and the proxy.

6.2 Fetching and caching documents

Both the agent and the proxy maintain a cache of their own. The cached documents are stored on disk. A memory cache would not make much sense since the differences in access times would be negligible when a wireless link is used. In addition, the file system cache in a typical Unix system does a good job of keeping frequently referred data in memory. The cache index is kept in memory and flushed to disk periodically.

When a WWW browser requests for a particular document, the agent first checks the document's URL against the cache index. This check can produce one of three results: present, missing or pending. If the document is present, the agent simply returns it to the client. If the document is missing, the agent forwards the request to the proxy, creates a new entry in the cache index, marks it as pending and places the client request into a wait queue. The wait queue is located in the newly created cache entry. If the cache entry already exists but is marked pending, the agent merely adds the request into the wait queue.

Eventually, the proxy will respond to the request with either an error response or a document response. As soon as the agent receives the response header, it starts sending the arriving response to the waiting client(s). In addition, a document response is stored

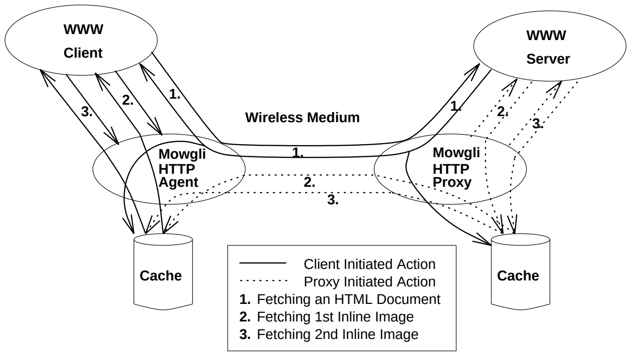

Figure 2: Removing round trip delays from the HTTP protocol

into the agent cache. From this moment on, any new HTTP request for this document object is served from the cache.

The proxy uses a very similar algorithm. It serves the requests from its own cache when possible. Otherwise it contacts the WWW server indicated by the URL in order to fetch the document. When a response from a WWW server becomes available for reading, the proxy checks the associated wait queue in its cache index. It is possible that several agents on different mobile nodes have requested the incoming document, so the proxy extracts from the wait queue those agents that have requested the document. While receiving the response from the WWW server the proxy simultaneously forwards it to these agents. The response is also written into the proxy's cache. If a WWW server is unavailable, the proxy generates the error response.

6.3 Fetching documents with images

While receiving an HTML page from a WWW server the proxy parses it to find references to inline images. The proxy checks the images against its cache index and automatically starts prefetching those images that are not found in the cache. A new cache entry is created for each pending image. The proxy constructs a wait queue for the new cache entry based on the queue in the cache entry of the HTML page. All agents that are interested in this HTML page and have not yet received a copy of the inline image are added into the wait queue. To do this the proxy maintains, for each agent, a list of inline images that it already has delivered to the agent. As a result of this algorithm each agent requesting an HTML page will in addition receive all the missing inline images without requesting for them.

While the agent is receiving the HTML page, it will also extract references to inline images. Knowing that the proxy will send the images automatically, the agent marks them in its cache index as pending. This will prevent the agent from unnecessarily requesting the images from the proxy. Figure 2 outlines how a hypertext document with two inline images is fetched. Request 1 retrieves the HTML page and requests 2 and 3 each retrieve a single inline image. The dotted arrows depict actions independently taken by the proxy. It is easy to see that only a single round trip delay is involved between the agent and the proxy. The figure also illustrates the parallelism in the system. The HTML page is being forwarded and written into both caches simultaneously while the proxy fetches the inline images.

Some WWW browsers can display data as soon as it arrives. If a user having such a browser decides not to wait for the whole document and follows a hypertext link to somewhere else, the unsent pending responses become a problem. In this case the agent informs the proxy that the transfer must be interrupted. The proxy can then either remove the unsent responses from the response queue or move them to the background transfer queue. The action depends on the agent's instructions.

6.4 Background transfers

The user can place the agent into a background transfer mode. In this mode the agent responds all HTTP requests immediately. If the desired document object is not in the cache, the agent releases the client with the HTTP response "204 No Response". Then the agent proceeds in the normal manner to fetch the document object into the cache using the low-priority data channels reserved for background transfers. The HTTP response "204" tells the client that the request was successful but there is no displayable response. A client receiving such a response ought to retain the original display unchanged. As far as inline images are concerned, the agent also has the option to deliver a suitable default image to the client.

Related to background transfer we plan to enable the agent to dynamically modify transferred HTML documents. This allows the agent to give feedback about the current state of the cache to the user. Each hypertext link will be followed by an icon that indicates whether the referred document object is present, pending or missing. Clicking on the "missing" icon will instruct the agent to fetch the document object in the background.

If any non-HTML object is larger than a certain user configurable size, the proxy can automatically decide either to refrain from sending it or to send it on a low-priority channel. In this case the proxy replaces the response in the normal channel with a suitable small default object or HTTP response "204". This prevents very large documents and images from accidentally clogging up the wireless link.

6.5 WWW Proxy Server interface

In addition to intercepting connections to WWW servers, the Mowgli HTTP Agent acts as a local WWW Proxy Server for clients with proxy server support. The advantage in providing the proxy server interface is that the clients do not need to make name server queries to find servers. The responsibility of finding a WWW server is left to the proxy server, which in our case is the HTTP agent-proxy combination.

Table 1: Test results

Test	Normal	Mediated	Improvement
1	26 s	14 s	46 %
2	52 s	31 s	41 %
3	87 s	65 s	25 %

This approach allows us to eliminate the DNS queries without actually having to implement the Mowgli DNS Agent discussed in Section 4.4. Another advantage is that WWW browsers convert the requests they normally make using other protocols (e.g. FTP, WAIS, Gopher, SMTP) to HTTP requests and let the proxy server handle them. Thus, our agent and proxy can also optimize these other requests over the wireless link. Very little additional implementation is needed. The Mowgli HTTP Proxy simply forwards such requests to a preconfigured WWW Proxy Server in the fixed network.

7 Discussion

Information retrieval is an important application for mobile computing. Mobile users have special requirements that should be taken into account when designing applications and communications architecture. We have discussed a number of performance problems with the World-Wide Web over wireless transmission media. We have also shown several ways to transparently improve the performance perceived by the end-user by exploiting the agents and proxies introduced in the Mowgli architecture.

Our pilot trials with the prototype indicate significant performance improvements. We experimented with three WWW pages with different selections of inline images. The first one included 12 small (less than half a kilobyte) images. The second included 4 small and 4 medium (2–6 kB) images. The third test document included 2 small, 4 medium and 2 large (10 kB and 20 kB) images. As a client we used Netscape Mosaic 1.0N with the internal cache disabled and the number of concurrent TCP connections limited to four. Our mediators were configured to automatically prefetch inline images. The average transfer times are given in table 1. In general, the mediators seem to reduce the total transfer time of a document by about 25–50 %.

In this paper our focus was on exploiting mediators for various performance enhancements. However, there are many features that could be, and ultimately should be, part of the client software. In particular, caching, prefetch, background transfer, and disconnected-mode operations are best implemented in the client. In general, all the functions performed by the Mowgli HTTP Agent could be incorporated into a conventional client to produce a WWW browser for the mobile community. The client has the advantage over an agent that it can tune the various optimizations to the display area and resolution. It also has a clearer view of the user's behaviour. Client modifications are also required for true mobility features, such as location dependent services. However, the role of the Mowgli HTTP Proxy remains important, barring widespread acceptance of mobility features into WWW servers all over the world.

Acknowledgements

This work was carried out in the Mowgli research project supervised by professor Martti Tienari and funded by Digital Equipment Corporation, Nokia Cellular Systems, Nokia Mobile Phones, Telecom Finland, and the Finnish Ministry of Education.

References

[1] T. Alanko et al. Measured Performance of Data Transmission Over Cellular Telephone Networks. Report C-1994-53, Univ. of Helsinki, Dept. of Computer Science, Helsinki, Finland, November 1994. To appear in Computer Communications Review.

[2] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. Technical Report DCS-TR-314, Rutgers University, Dept. of Computer Science, October 1994.

[3] T. Berners-Lee and D. Connolly. HyperText Markup Language Specification – 2.0. Internet draft, IETF, HTML Working Group, November 1994. Available at <URL:http://www.ics.uci.edu/pub/ietf/html/html2spec.ps.gz>.

[4] T. Berners-Lee et al. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[5] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen. Hypertext Transfer Protocol – HTTP/1.0. Internet draft, IETF, HTTP Working Group, December 1994. Available at <URL:http://www.ics.uci.edu/pub/ietf/http/draft-fielding-http-spec-01.ps.Z>.

[6] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521, September 1993.

[7] D. H. Crocker. Standard for the Format of ARPA Internet Text Messages. RFC 822, August 1982.

[8] F. J. Hauck. Optimizing Latency for the HTTP Protocol. Technical Report TR-I4-94-23, University of Erlangen-Nürnberg, Germany, October 1994.

[9] M. Kojo, K. Raatikainen, and T. Alanko. Connecting Mobile Workstations to the Internet over a Digital Cellular Telephone Network. In *Proc. Workshop on Mobile and Wireless Information Systems (MOBIDATA)*, Rutgers University, NJ, November 1994.

[10] P. Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[11] K. Moore. MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extension for Non-ASCII Text. RFC 1522, September 1993.

[12] Michel Mouly and Marie-Bernadette Pautet. *The GSM System for Mobile Communications.* Mouly and Pautet, 1992.

[13] V. N. Padmanabham and J. C. Mogul. Improving HTTP Latency. Electronic Proc. Second World Wide Web Conference '94: Mosaic and the Web, Available at <URL:http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html>, 1994.

[14] M. Rahnema. Overview of the GSM System and Protocol Architecture. *IEEE Communication Magazine*, 31(4):92–100, April 1993.

[15] S. Spero. Analysis of HTTP Performance Problems. Univ. of North Carolina, Available on WWW at <URL:http://sunsite.unc.edu/mdma-release/http-prob.html>, 1994.

[16] S. Spero. Progress on HTTP-NG. Univ. of North Carolina, Available on WWW at <URL:http://info.cern.ch/hypertext/WWW/Protocols/HTTP-NG/http-ng-status.html>, 1994.

[17] R. Yavatkar and N. Bhagawat. Improving End-to-End Performance of TCP over Mobile Internetworks. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, Calif., December 1994.