Efficient Web Browsing for Mobile Clients using HTTP Compression

Ronny Krashinsky
Distributed Operating Systems (6.894) Term Project
MIT Laboratory for Computer Science, Cambridge, MA 02139
ronny@mit.edu

12-11-2000

Abstract

Efficient web browsing on mobile computers presents a unique challenge. These machines are different from other classes of client computers since they have relatively low-bandwidth connections and they are battery-powered and therefore limited by their energy consumption. However, they tend to interact with the same servers for the delivery of web content. This project investigates optimizing the final critical link between a mobile client and a stationary base station by compressing HTTP request and response messages. Using a split proxy design, compression of individual request messages reduces bandwidth by 26% to 34% across a variety of benchmark traces, and applying compression to response messages yields savings of 59% to 82% of the compressible data. Higher compression rates are achieved by using streaming compression algorithms to compress the streams of request and response messages. In this case, the bandwidth for requests sees an order of magnitude improvement, and the response stream obtains additional savings of 7% to 25% on top of the savings achieved with per-response compression.

1 Introduction

There is a widening gap between two classes of client computers. At one end are stationary desktop machines with persistent high-bandwidth connections and relatively unconstrained energy usage. At the other end are battery-powered mobile devices that have low-bandwidth connections, and have functionality limited by energy constraints. Figure 1 illustrates this situation. These two client classes clearly have different requirements; however, they tend to interact with the same servers for access to internet content. The servers are for the most part unaware of the needs of the clients they interact with; this unified view is arguably a good thing in terms of limiting system complexity. Additionally, the networks which connect these machines together look the same in terms of bandwidth and energy constraints up until the last hop from a base station to a mobile client. This project seeks to optimize the performance of this final low-bandwidth energy-constrained connection.

For a mobile client, communication is very expensive. In terms of delay, the communication bottleneck makes computation essentially free in comparison. Additionally, the energy cost of sending or receiving a single bit is several orders of magnitude greater than the cost of executing an instruction. For example, [11] models an aggressive radio design as dissipating 50 nJ/bit for sending or receiving data, plus an additional 100 pJ/bit/m^2 for transmitting a signal over a distance. In comparison, current low-power processor designs can operate in the range of 0.27 nJ/cycle to 1.125 nJ/cycle (derived from [5]). The gap between communication and computation will continue to grow as processors on mobile devices become faster and more energy-efficient at the levels of software, architecture, and hardware technology. This imbalance opens up an opportunity, or perhaps necessity, for data compression.

Much internet content is in fact compressed, for example, gif and jpeg images, streaming audio and video, and vector graphics. However, a large portion of internet content is text-based, and this data is usually not compressed. Interestingly, most current web browsers do support compressed text formats, but most content providers do not take advantage of this feature. This may be due to any one of a myriad of reasons, but from the point of view of this project, content servers are uncooperative and do not provide compressed data in general. However, even though servers don't provide compressed content, a base station can compress data before it is sent over the final critical hop to a mobile device. An advantage here is that cooperation is only required between the base station and the mobile device, which in general can be tightly coupled.

This project focuses on using compression to minimize the bandwidth on the communication link between a mobile client and a stationary base station. To implement this, a split proxy design is described in which proxies on the client machine and base station cooperate to compress the data communicated between them. In order to obtain higher compression rates, streaming compression is implemented. In this case, compression is applied to the stream of data communicated across the client to base link, rather than to individual messages alone.

2 Basic Web Browsing

The Hypertext Transfer Protocol (HTTP) defines the interface which enables web browsing. This study is based on version 1.0 of this protocol [3]. The HTTP protocol allows for simple transactions between clients and servers using a request/response message pair. HTTP transactions are conducted over Transmission Control Protocol (TCP) connections; a connection is established for each transaction, and closed once the transaction completes. The Hypertext Markup Language (HTML) is the most common format for web pages. HTML pages may contain embedded links to other objects such as images. A web browser fetches a web page by establishing an HTTP connection with the appropriate server, and requesting the desired HTML page. It then parses the HTML content, and fetches any embedded images using additional HTTP transactions. These fetches are typically done in parallel. This structure is diagramed in Figure 2.

A web browser can be configured to use a web proxy, as shown in Figure 3. All of the browser's transactions go through the proxy which can provide services such as caching and access control. The interface provided to the browser is mostly transparent; except, for example, it must communicate to the proxy the name of the server that an HTTP request should go to.

3 Split Proxy with Compression

In order to dynamically compress HTTP content, I implemented a split web proxy as shown in Figure 4. In this split proxy design, the client proxy acts as a tunnel between the browser and the base proxy. For every new TCP connection that the browser initiates, the client proxy establishes a corresponding TCP connection with the base. The base proxy does the necessary HTTP processing to implement the standard web proxy interface. This design places no limitations on the existence of parallel HTTP connections. The goal of the split proxy is to compress the data on the link between the client proxy and the base proxy.

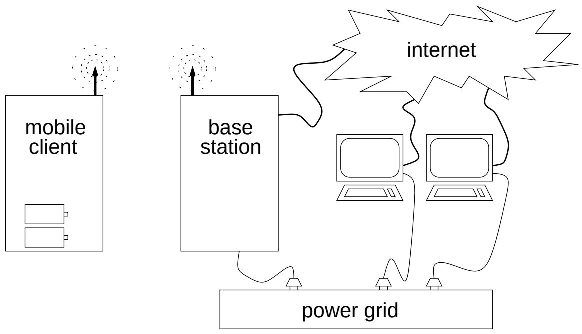

Figure 1: The constraints of a mobile client. Communication is over a low-bandwidth wireless link, and energy is supplied by batteries.

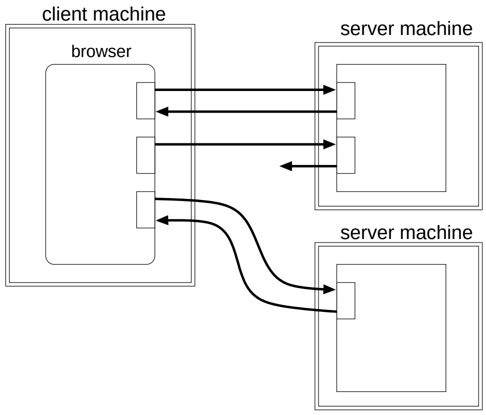

Figure 2: Basic client/server web browsing.

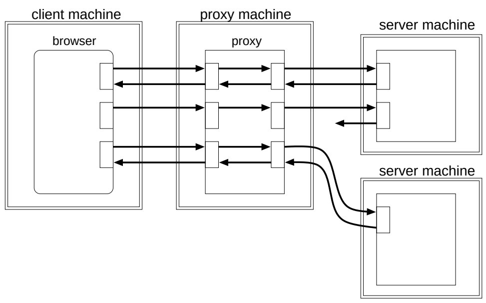

Figure 3: Web browsing using a proxy.

3.1 HTTP Request Compression

HTTP requests are text messages, and therefore compressible to some degree; I choose to compress all incoming requests. One way to implement this compression would be for the client proxy to get the entire request from the browser, compress it, and then send it along to the base proxy. A disadvantage with this scheme is that it requires the client proxy to parse the HTTP in order to determine when an entire request has been received. Instead, the approach taken is to compress and transmit the incoming parcels of data as they arrive. This could be done by compressing each parcel of data individually, but in this case extra header information would have to be communicated to the base proxy so that it could separate the incoming data correctly before uncompressing each parcel. Therefore, a streaming compression algorithm is used; instead of compressing a block of data all at once, such an algorithm continuously accepts a stream of bytes as input and produced a compressed stream as output.

The base proxy uncompresses the request stream as it is received from the client. It does the necessary HTTP parsing, and forwards the request on to the appropriate server once it is complete. If the server never responds, the connection will time out, and all associated state in the base and client will be freed.

3.2 HTTP Response Compression

Some web data is already compressed, images for example, so compression should not be applied to all HTTP responses. In order to determine whether a response should be compressed, the base proxy examines the HTTP header. If the `Content-Type` field is `text` of any sort and the `Content-Encoding` field is not present, the response is chosen for compression. In addition, compression is applied to any response for which the length of the HTTP header is more than that of the content (as determined by the `Content-Length` header if present). This is because compression is applied to the header as well as the data, so applying compression to responses that contain a small amount of encoded data may be useful.

Once the base proxy has processed the HTTP header and determined whether or not to compress the response, it could begin streaming the data to the client in the same manner as with the requests. However, the current implementation actually receives the entire response before processing it and forwarding it to the client. This design artifact has a negative impact on the delay of getting the response to the browser, and should be corrected in a more optimized implementation.

In order to determine whether or not the data it receives is compressed, the client proxy checks if the initial characters in the response are "HTTP"; this string begins every HTTP response, and therefore indicates uncompressed data if present. At this point, the client proxy begins streaming the data to the browser, uncompressing the stream as the data arrives if necessary. The various connections are destroyed when the browser or server disconnect, or a timeout occurs.

4 Split Proxy with Streaming Compression

Higher compression rates can be possible if the compression is applied to a larger collection of data. To achieve better compression rates, I considered compressing the HTTP requests and responses as a continuous stream rather than as individual objects. The difficulty in doing this is that it is contrary to the browser's abstraction that each HTTP transaction is orchestrated using its own connection.

The setup for streaming compression is shown in Figure 5; it is somewhat similar to the virtual sockets described in [12]. When the browser initiates a new connection with the client proxy, the client proxy makes a corresponding connection with the base proxy as before. In addition, it sends a connection id number to the base proxy over this

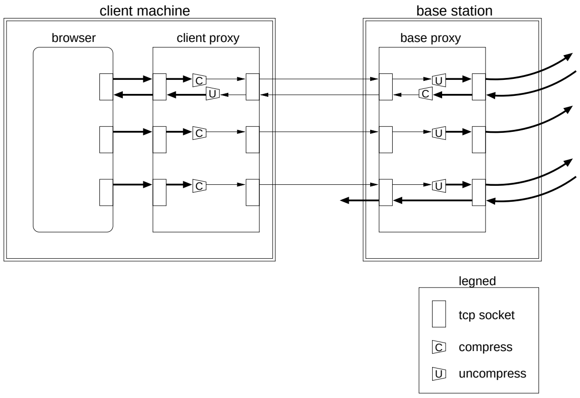

Figure 4: Split proxy compression scheme.

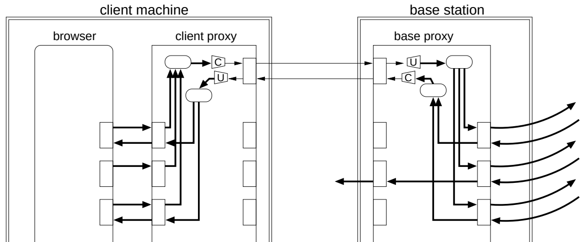

Figure 5: Split proxy with streaming compression scheme.

connection in order to maintain a namespace for the connections shared by the two proxies (this transaction is not shown in Figure 5). In a more optimized implementation, this could probably be done at the TCP level since the two machines must already have a shared namespace for these connections in order to communicate.

The client and base proxies maintain a persistent connection over which to send the stream compressed data. In order for the other side to be able to interpret the stream of data, a header message is sent before each parcel of compressed data. This header consists of the compressed size and uncompressed size of the data parcel, and the connection id number which it is associated with. Many connections with the browser and outside web servers may be operating in parallel, but all data sent over the stream compression channel must be serialized. To efficiently support this, when a connection is ready to send a data parcel over this channel, it is stream compressed and the resulting data and header information is placed on a queue; this queue is processed whenever the connection can accept more data. On the receiving side, the header information is peeled off, and then the stated amount of data is uncompressed as it arrives. This data parcel is then passed off to the state associated with the connection specified in the header.

As the client proxy receives blocks of HTTP request data from the browser, it enqueues these on the stream compression queue. It is possible that blocks of data from various requests will be interleaved in this queue. The base proxy uncompresses this data as it comes in, and passes it off to the state associated with the given connection. The HTTP request is then parsed, and forwarded to the appropriate server as before.

When a response arrives at the base proxy, it determines whether or not it should be compressed as described in Section 3.2. If the data will not be compressed, it is sent to the client proxy over the associated connection, and everything proceeds as before. If the response is to be compressed, it is added to the stream compression queue. When the client eventually receives and uncompresses the response, it is passed to the state associated with the connection, and then forwarded on to the browser.

An inefficiency in this design is that in many cases a TCP connection is created between the client proxy and base proxy, but never used because both the HTTP request and response use the stream compression connection. A more optimized design could only establish this connection for the cases in which a response is not stream compressed. Or, these connections could be eliminated all together, and all data could be multiplexed over a single connection between the base and client; this is more similar to the approach taken in [12].

5 Implementation

The split proxy compression schemes were implemented using object oriented C++ code. The compression was performed using zlib version 1.1.3 [10], which uses an adaptive compression strategy based on LZ77 and Huffman coding, and supports a streaming interface; level 9 compression (the highest) was used. The design made use of the C++ asynchronous programming library and the HTTP 1.0 parser provided by the MIT 6.894 lab [1]. The static buffers used for receiving data were 4 kB.

The implementation was tested using both Netscape Communicator version 4.74 and Microsoft Internet Explorer version 5.00.2013.1312. The client and server proxies were run on Sun Ultra 5 workstations on different machines within a LAN. Multiple simultaneous HTTP requests from different browser instances were verified to work correctly, as well as connections prematurely terminated by either the browser or the server. There are no know bugs.

6 Results and Analysis

6.1 Test Traces

Six traces were constructed to test the performance of the compression schemes, as shown in Table 1. The first trace was made by following a series of links and queries to investigate two professors at MIT. The second trace explores the CNN website by retrieving the headline page for a sequence of news sections. The third trace visits the same pages as the second, except images were disabled in the web browser. The fourth trace uses the CNET site to retrieve stock quotes and news headlines for the first ten stocks in the Dow Jones Industrial Average. The fifth trace visits the ten most popular websites as determined by recent statistics. Finally, the sixth trace sequentially fetches the nine pages in the online gzip documentation.

Netscape Communicator version 4.74 was used as the browser for collecting data for all the traces. The client and base proxies ran on the same machine, and the browser was configured to use the client proxy. Netscape's cache was cleared before running each test, but caching was enabled during the tests. For each test, the number of HTTP message bytes communicated across the link between the client and base proxy was counted. It should be noted that the actual number of bytes which are communicated between the base and client will depend on the details of the underlying TCP implementation.

The results are shown in Figure 6.1. The graphs show the number of bytes transmitted in the base case, and using the two compression schemes. For responses, the totals are

Trace 1
www.mit.edu
web.mit.edu/research.html
www.lcs.mit.edu
www.lcs.mit.edu/people
www.lcs.mit.edu/search/people_results?name=kaashoek
www.lcs.mit.edu/research/groups/group?name=pdos
www.pdos.lcs.mit.edu/
www.pdos.lcs.mit.edu/people.html
www.pdos.lcs.mit.edu/ kaashoek/
www.pdos.lcs.mit.edu/ rtm/
www.pdos.lcs.mit.edu/click/
www.nms.lcs.mit.edu/projects/ron/
www.pdos.lcs.mit.edu/grid/
www.pdos.lcs.mit.edu/ rtm/Projects.html

Trace 2, 3 (no images)
www.cnn.com
www.cnn.com/WORLD/
www.cnn.com/US/
www.cnn.com/WEATHER/
www.cnn.com/TECH/
www.cnn.com/HEALTH/
www.cnn.com/SHOWBIZ/
www.cnn.com/ALLPOLITICS/
www.cnn.com/LAW/
www.cnn.com/FOOD/
www.cnn.com/books/

Trace 4
www.cnetinvestor.com/quote-fast.asp?symbol=AA
www.cnetinvestor.com/quote-fast.asp?symbol=AXP
www.cnetinvestor.com/quote-fast.asp?symbol=T
www.cnetinvestor.com/quote-fast.asp?symbol=BA
www.cnetinvestor.com/quote-fast.asp?symbol=CAT
www.cnetinvestor.com/quote-fast.asp?symbol=C
www.cnetinvestor.com/quote-fast.asp?symbol=KO
www.cnetinvestor.com/quote-fast.asp?symbol=DD
www.cnetinvestor.com/quote-fast.asp?symbol=EK
www.cnetinvestor.com/quote-fast.asp?symbol=XOM

Trace 5
www.yahoo.com
www.microsoft.com
www.lycos.com
www.aol.com
www.altavista.com
www.egroups.com
www.cnn.com
www.excite.com
www.google.com
www.cnet.com

Trace 6
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_1.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_2.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_3.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_4.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_5.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_6.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_7.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_8.html
www.gnu.org/manual/gzip-1.2.4/html_chapter/gzip_9.html

Table 1: Test traces.

	Requests		Responses	
Trace	comp	stream-comp	comp	stream-comp
1	30	89	62	71
2	28	90	75	79
3	26	87	78	79
4	33	90	80	82
5	27	87	74	78
6	34	86	59	69

Table 2: Total savings for HTTP request and response messages using per-message compression (comp) and streaming compression (stream-comp). Savings are given as the percentage of total bytes eliminated using compression. For responses, the savings correspond only to data which was determined to be compressible.

divided between the responses which were determined to be incompressible, and those for which compression was applied. The overhead of sending the header information and connection id information for streaming compression is also shown. This same data is summarized in Table 2.

The most salient feature of these results is that they are mostly the same for the different traces, aside from the amount of incompressible data transmitted. Table 2 shows that HTTP requests are compressed by 26% to 34% using per-request compression, and 86% to 90% using streaming compression. This extreme compression is possible because the browser resends a lot of the same information with each request, for example the content types which it supports. The compressible HTTP responses are compressed by 59% to 80% using per-response compression, and 69% to 82% using streaming compression. Comparing streaming compression with the per-response compression, the additional savings obtained are 7% to 25%.

6.2 Extended Use

In another test, I used the stream compression proxy to do all of my web browsing over a period of about two days, and convinced three of my colleagues to use it as well. We all configured our browsers to use a single client proxy. The statistics for HTTP responses collected during this time are shown in Figure 7. Almost 3000 HTTP responses were sent, comprising a total of about 14 MB of data. 43% of the data was determined to be compressible, and the streaming compression reduced the size of this data by 80%; overall, this reduced the number of bytes sent by 35%. 1.3 MB of HTTP request bytes were sent during this test. Streaming compression was not used for requests in the version of the proxies which were used; the savings achieved with per-request compression were 30%.

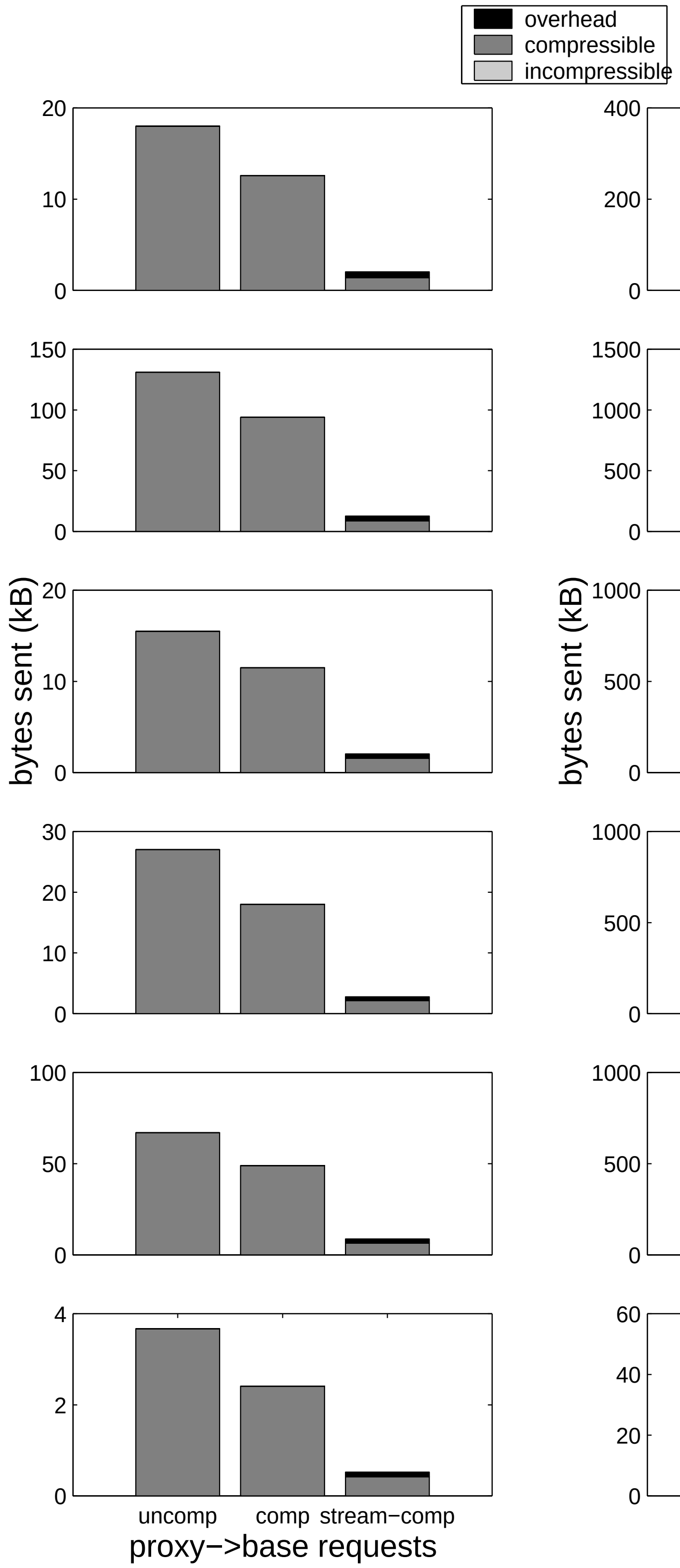

Figure 6: Compression results for traces of Table 1. Results are shown for traces 1 through 6 from top to bottom, with request results on the left and response results on the right. In each graph, the three bars correspond to no compression (uncomp), per-message compression (comp), and streaming compression (stream-comp). Each bar is divided between compressible, incompressible, and overhead bytes.

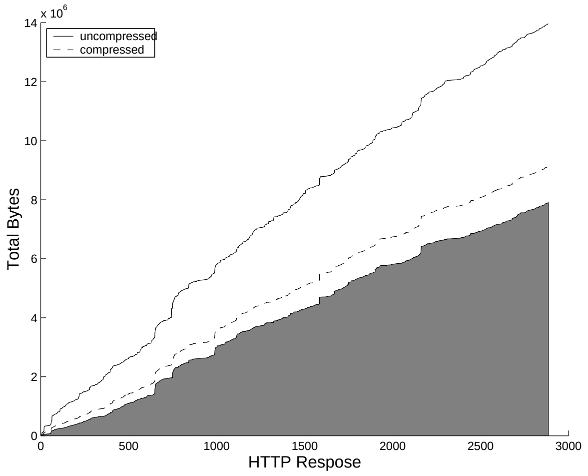

Figure 7: The cumulative number of bytes transmitted over a sequence of many HTTP response messages. The solid line shows data for the original stream and the dashed line shows the data obtained using the stream compression strategy. The shaded region represents HTTP response messages which were determined to be incompressible.

7 Delay and Energy Considerations

Ultimately the goal of compressing the data communicated between the mobile client and the base station is to improve the delays associated with web browsing, and to improve the energy efficiency of the client. However, these metrics are very dependent on the details of the network connections and machine configurations, and were not studied in detail in this project.

For delay, the computation time for compression can most likely be considered free compared to a relatively low bandwidth wireless connection, as shown in studies such as [7, 14, 17]. Thus, the most important consideration is that the proxy system doesn't break the streaming nature of the HTTP connections, or inhibit connections from operating in parallel. The use of streaming compression algorithms enable the HTTP messages to flow through the proxies with minimal delay impact; and, multiplexing the data from many connections over a single TCP link allows many connections to operate in parallel even with streaming compression.

The main consideration for energy efficiency is the amount of extra work the mobile client must do, compared to the reduction in the number of bits which it must send and receive. In the system presented, this extra work is minimized by only applying compression to data which is not already compressed. Additionally, uncompressing data is usually much less compute intensive than compression, so the compression of HTTP responses can be highly optimized. The client can use less compute intensive compression algorithms for the compression of HTTP requests, for example [15].

8 Related Work

Related work in HTTP compression includes [14] which focuses on delta-encoding; the goal is to update a cached copy of a page by only sending information about the portions of the page which have changed. This work ultimately advocates a combination of delta-encoding and data compression to minimize the amount of data transmitted and the response time. WebExpress [12] uses a split proxy design to minimize the overhead of creating a new TCP connection for each HTTP transaction. It also focuses on reducing the excess traffic due to the repetition of header fields in each HTTP request; to do this, WebExpress takes a more direct approach and establishes a protocol in which the proxies cooperate to avoid transmitting these headers. WebSwift [2] compresses HTTP responses in an ISP or web server when communicating with browsers that support compressed data formats.

The more recent HTTP 1.1 protocol [7, 8] provides for persistent connections between a client and server to minimize some of the overheads associated with fetching multiple objects by establishing separate connections for each transaction. It would be interesting to evaluate the benefits of streaming compression when used with this protocol.

At a lower level, several studies have investigated specific techniques for compressing header information [13, 6]. This compression can be complimentary to the compression of HTTP messages, particularly in the case of TCP headers. Compression has also been performed at the lowest level in data compressing modems [16]. This compression is generally not as effective as software based compression schemes [7, 14], and can perform poorly when sending compressed data formats. Nevertheless, a more extensive comparison would be interesting, particularly relating to energy efficiency.

Several studies have been performed in which the browser itself is actually split across the client machine and base station [4, 9, 18]. The potential for optimization in this case is much greater since the base station can perform content-specific lossy compression such as scaling and dithering of images before sending them to the mobile client. The base station side of the browser can also handle the fetching of embedded images, eliminating the delay and data transmission overhead associated with requiring the browser on a mobile client to do this. This feature could actually be implemented in the split proxy architecture even without modifying the browser by having the base proxy parse HTML files and fetch embedded images; then the client proxy could avoid forwarding requests from the browser for any images that have already been requested by the base proxy.

9 Summary

This project has focused on reducing the bandwidth between a mobile client computer and a base station. Mobile computers are becoming increasingly powerful, yet they are constrained by their energy consumption and communication bandwidth. A split proxy design has been described which works transparently with existing web browsers. Compression of HTTP request and response messages between the two proxies yields a substantial reduction in link bandwidth. An improved system has also been described in which a persistent connection is maintained between the split proxies, and stream compression is applied to the sequence of request and response messages. This results in additional bandwidth savings, especially for request messages.

References

[1] http://www.pdos.lcs.mit.edu/6.894/lab/index.html.

[2] http://www.unxsoft.com/webswift/.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*, May 1996.

[4] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power browser: Efficient web browsing for pdas. In *CHI 2000*, 2000.

[5] Intel Corporation. *Intel XScale Microarchitecture Technical Summary*, 2000.

[6] M. Degermark, M. Engan, B. Nordgren, and S. Pink. Low-loss tcp/ip header compression for wireless networks. In *MobiCom, 1996*, 1996.

[7] H. F. Nielsen *et. al.* Network performance effects of http/1.1. In *ACM SIGCOMM'97*, August 1997.

[8] R. Fielding, J. Gettys, and J. Mogul *et. al. Hypertext Transfer Protocol – HTTP/1.1*, June 1999.

[9] A. Fox, I. Goldberg, S. Gribble, A. Polito, and D. Lee. Experience with top gun wingman: A proxy-based graphical web browser for the palm pilot pda. In *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 15–18, Lake District, UK, September 1998.

[10] J.-L. Gailly and M. Adler. *zlib data compression library 1.1.3*. http://www.info-zip.org/pub/infozip/zlib/.

[11] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Hawaii International Conference on System Sciences*, Maui, HI, January 2000.

[12] B. C. Housel and D. B. Lindquist. Webexpress: A system for optimizing web browsing in a wireless environment. In *The Second Annual International Conference on Mobile Computing and Networking*, pages 108–116, Rye, New York, November 1996.

[13] J. Lilley, J. Yang, H. Balakrishnan, and S. Seshan. A unified header compression framework for low-bandwidth links. In *6th MobiCom*, Boston, MA, August 2000.

[14] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *SIGCOMM '97*, Cannes, France, September 1997.

[15] M. Oberhumer. *LZO version 1.07*. http://wildsau.idv.uni-linz.ac.at/mfx/lzo.html.

[16] C. Thomborson. The v.42bis standard for data-compressing modems. *IEEE Micro*, pages 41–53, October 1992.

[17] J. R. Velasco and L. A. V. Lucianez. Benefits of compression in http applied to caching architectures. In *The Third International WWW Caching Workshop*, Manchester, England, June 1998.

[18] T. Watson. Wit: An infrastructure for wireless palmtop computing. Technical Report CSE-94-11-08, University of Washington, November 1994.