IMPROVING THE DELIVERY OF MULTIMEDIA EMBEDDED IN HTML OVER HTTP ON WIRELESS NETWORKS

Adam Serbinski
Department of Computer Science
Ryerson University
350 Victoria Street
Toronto, Ontario
Canada M5B 2K3
aserbins@ryerson.ca

Abdolreza Abhari
Department of Computer Science
Ryerson University
350 Victoria Street
Toronto, Ontario
Canada M5B 2K3
aabhari@scs.ryerson.ca

ABSTRACT

The purpose of this work is to reduce the delivery time for the initial portion of multimedia objects over wireless networks using HTTP protocol. The multimedia files we consider are those embedded in web pages. By prefetching embedded media from server to client, we are able to overcome the effects of network latency.

We implement prefetching in a custom built HTTP server capable of anticipating future requests from the client, and delivering data without it explicitly being requested.

To allow the client to receive files not requested, we use a custom built proxy, to run on the client system.

Our custom server and proxy implement modifications to the HTTP protocol to allow multiple files to be delivered in a single transmission.

KEYWORDS

Multimedia networking, wireless networks, web server, prefetching, network latency.

1. INTRODUCTION

Wireless networking commonly uses the same protocols that are used in wired networks. Though the techniques we use are not strictly wireless techniques, they can be applied to both wired and wireless networks. Specifically our techniques improve the efficiency of the HTTP protocol, which is shared between wired and wireless networking.

MOBIMEDIA 2007, August 27-29, Nafpaktos, Greece

DOI 10.4108/ICST.MOBIMEDIA2007.1602

In the HTTP/1.0 protocol, every file being retrieved from the server requires its own connection to be established between the client and the server[1]. This places additional resource strain on the server and on the client than what would be experienced using a single connection. Since embedded objects are referenced within HTML, these files cannot be requested until after the HTML file has already been retrieved. In the worst case scenario, where an embedded object is referenced at the end of an HTML document, this may mean that the final, or only embedded object of a web page cannot even be requested until a long enough time has elapsed to retrieve the entire HTML document.

In HTTP/1.1, this situation has been marginally improved. This version of HTTP supports persistent connections and request pipelining[2]. What this means is that even though the client may still be forced to wait until the entire HTML file has been received, as it scans through the HTML file, it can submit requests along the same connection, rather than having to establish new connections. This can result in a significant reduction in server and client workload, and reduced file transfer times. HTTP/1.1 still cannot eliminate the initial wait for the HTML file to be received, and the request for the embedded objects to be made and responded to. HTTP/1.1 can in most cases improve on page load times when compared to HTTP/1.0, however, the worst case performance of HTTP/1.1 for HTML files that do contain at least one embedded object, is no better than HTTP/1.0

We have developed an additional enhancement to the HTTP protocol to solve the problem of requiring the client to wait for the HTML file to be delivered before being able to request the embedded objects. In our solution, the server delivers the embedded objects to the client without being explicitly requested. Other work has considered anticipation of client requests for improving web server performance[12,13], however, rather than considering the

overall probable next file to be requested, we focus on the contents of HTML files that have already been requested.

Our previous work in [3] involved modifying Apache HTTP Server to prefetch embedded objects from disk into memory. This modification allowed the server to generate a response to a client request in a shorter amount of time by reducing the effects of disk latency. This work is an extension of our previous work to reduce the effects of network latency, which has a far greater impact on web page delivery time than disk latency.

In addition to solving the problem of network latency, our modification is also able to overcome TCP slow start, which is a technique applied to maximize the rate of data transmissions without full knowledge about the transmission path[4]. In TCP slow start, the server uses a congestion window, which it initially sets to 1 segment. It will send a small amount of data equal to the window size before waiting for an acknowledgment from the client. At each acknowledgment, it will double the window size until it reaches a threshold, which is near to the path's maximum capacity, before switching into additive increase. When data fails to arrive at the client, the server recognizes this by not receiving an acknowledgment and reduces the window size by half to ensure reliable delivery. Because TCP communications use slow start for congestion avoidance, the transfer of small files may not provide adequate opportunity for full bandwidth utilization. When grouping all the files of a website in a single connection, TCP is able to build up to a greater transmission rate before the connection is terminated.

This work focuses on a special case of use for HTTP and TCP protocols; delivery of multimedia objects embedded in web pages.

Media files, including multimedia, can be divided into two groups; those that play over a period of time, and those that do not. Non-playable media primarily includes images, which to be properly displayed, require the entire file. Playable media include, among other things, sounds, animations, and video. Of the forms that play, there are again two types; streaming and non-streaming. Streaming media is distinguished from non-streaming media as being delivered while at the same time being played[5], whereas non-streaming media plays after delivery.

In some cases, media that is not specifically streaming media can be streamed, such as stored sounds and videos. Some media players are capable of beginning playback on such data before the entire file has been received.

The HTTP protocol was once the most popular protocol used to deliver data on the Internet[6]. Being the only protocol handled by most web browsers, HTTP is still a highly popular protocol for delivering data on the Internet, however, it is not capable of delivering true streaming media[1]. The HTTP protocol requires files to be completed before being delivered in order to properly generate a response header, which indicates to the client what it should expect to receive. HTTP does, however, deliver files in order from the start of the file to the end, which is the same order that most playable media files are read when being played.

Given that HTTP delivers requested files in order, and that some media players are capable of beginning playback of received files before being completely received, it is possible to stream stored media files over HTTP.

The purpose of this work is to improve the delivery of multimedia files embedded in web pages such that the time between issuing a request for a web page and starting playback is minimized.

2. PROPOSED STRATEGY

HTML is a document format with a particular structure that web clients are able to parse and interpret in order to display web pages in the same manner as the author of the page intended[11]. Embedded within the HTML may be references to other files that may be required to properly display the web page, including, but not limited to, images, sounds, animations, and videos.

Our strategy for improving the load time for files embedded in HTML documents served by using HTTP protocol is to prefetch the embedded objects directly to the client along with the HTML file. The prefetching of the objects is done without the client explicitly requesting the file, therefore it reduces the time to receive the embedded files by the time that it takes for a request to be relayed to the server and back.

In using normal HTTP protocol, every file must be explicitly requested by the client in order for the server to send it[1]. Before the client can request any embedded object, it must receive and parse the HTML file. It then creates new requests to the server for those objects, which add to the network traffic, the server's queue, and force the server to spawn multiple new processes to handle those requests. This adds a significant load to the network and to the server.

With our modified HTTP protocol, implemented in our

custom HTTP Server, the client issues a request only for the HTML file, and the server appends the embedded objects in the same response as with the HTML file. In prefetching HTTP, it is not necessary for the server to spawn multiple processes or threads to service one web page. There are also fewer requests in the queue, the network traffic is decreased, and due to the greater size of the transmission, the network bandwidth may be more fully utilized.

In a standard retrieval of a web page using a single connection from the client to the server, the time, ***t***, to begin receiving a document from a web server, can be expressed as the following for HTTP/1.0;

$t = h + r + p + s$, where;

h is the time to negotiate a connection,

r is the time to send a request from client to server,

p is the time for the server to process the request,

s is the time for the first byte of the response to be sent from the server to the client.

Further, the time R_i, to receive the i^{th} document, can be expressed as;

$R_i = t + size_i \div rate$, where ***rate*** is the transfer rate of the connection.

The time to receive all of the documents related to a web page is, therefore;

$T = \Sigma R_i$, for ***i*** from ***1*** to ***n*** files in the web page.

This can also be expressed as;

$T = R_1 + t \times (n-1) + (\Sigma size_i) \div rate$, for ***i*** from ***2*** to ***n***.

In prefetching, ***T*** is reduced by ***t*** for each embedded object;

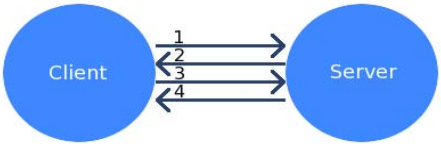
$t \times (n-1)$

So, for prefetching, the time to retrieve the entire web page is;

$T = R_1 + (\Sigma size_i) \div rate$, for ***i*** from ***2*** to ***n***

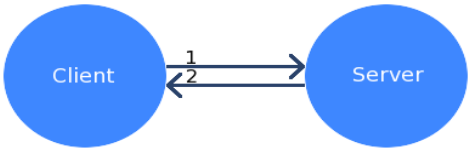
For HTTP/1.1, ***T*** is reduced only by an additional ***h*** for every additional file requested along an established connection. ***h*** is a small component of ***t***.

A typical web page retrieval using HTTP follows these steps;



1. The client requests an HTML file from the server.
2. The server delivers the HTML file to the client.
3. The client requests embedded objects from the server.
4. The server delivers embedded objects to the client.

The proposed solution to improve the delivery time of a web page follows these steps;



1. The client requests an HTML file from the server.
   - The HTTP Server parses the HTML file and creates a list of all embedded objects that can be forwarded to the client.

2. The server delivers the HTML file along with all embedded objects to the client.

A normal HTTP client, particularly a web browser, does not have the ability to receive multiple files in a single transmission unless they have been requested. In order to accomplish this, the client must be modified. There are two possible approaches to client modification;

- Implementation of the modification within the web browser.

- Addition of a special light-weight proxy on the client machine to translate between the modified HTTP protocol used by the server and the standard HTTP protocol used by the browser.

Modification to the browser is impractical, since it would require modification of all browsers in order for this system to become effective. We focused on designing a special proxy translator since it will be compatible with all web browsers.

The proxy translator receives a proxy request from the client browser. It then modifies the request to an HTTP request and appends a header indicating its compatibility with the modified HTTP protocol.

Prefetching: true

When the server receives this request and reads the added header, it formulates a response containing all embedded objects and sends that to the proxy. If the server receives a request that does not have the added header, it will generate a normal HTTP response. If the added header is received by a server that does not understand it, the header will be ignored and a normal response will be sent.

In the modified HTTP response, the path of each embedded object is added to the response header. This is to indicate to the proxy immediately, which files will be sent. This is

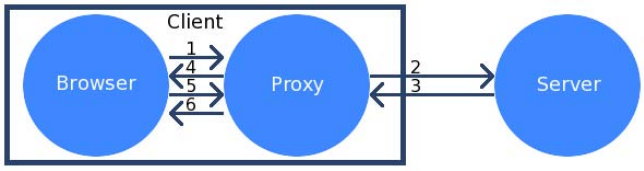
necessary because the HTML file is sent first in the response, and is forwarded immediately from the proxy to the client browser without waiting for the full transmission, which makes it possible still for the client to generate a request for an embedded object before it has arrived. It is important for the client browser to be able to generate requests before receiving the full transmission in order for it to retrieve objects stored on other servers. Prefetching can only forward embedded objects located on the same server that is performing the prefetching. Unlike HTTP/1.0 and most applications of HTTP/1.1 protocol, this modified protocol is strictly dependent on the Content-Length header, this is because the client uses the content length to judge where in the transmission the headers for the next file begins.

Files in the transmission are organized as follows;

Header for HTML file including list of embedded object paths in order of transmission
{blank line}
HTML file
{blank line}
Header for first embedded object
{blank line}
First embedded object
{blank line}
Header for second embedded object
etc.

Since the web client itself, such as a web browser, is not modified, communications between the client browser and the client proxy are also unmodified. Since the proxy is a program running on the client system, request latency is negligible.

The modified web page retrieval follows these steps;



1. Client web browser requests HTML file from the proxy.
2. Proxy forwards request for HTML file to the server.
- The HTTP server parses the HTML file and creates a list of all embedded objects that can be forwarded to the client.
3. The server delivers modified response, including all embedded objects, to client proxy.
4. The proxy delivers only HTML file to the browser.
5. The client web browser requests embedded object from the proxy.
6. The proxy delivers prefetched embedded objects to the browser.

3. EXPERIMENTAL DESIGN

We have performed two experiments, described in sections 3.1 and 3.2.

For both experiments, we use the same server and client. Our server is located on a residential cable Internet connection with maximum upload bandwidth of 800 Kb/s and maximum download bandwidth of 6.0 Mb/s. Our client system is located on a separate network with similar maximum capacities and utilizes an 802.11G Wi-Fi access point and network adapter.

3.1. NORMAL WEBSITES

Our first experiment involves testing websites with no multimedia objects. For this experiment, we mirror several popular websites and measure the performance difference in retrieving these web pages with prefetching and without.

The web pages used are obtained from lists generated by IRCache[8] and SeekingAlpha[9], and include the home pages for amazon.com, aol.com, mapquest.com, wikipedia.org, and google.ca. These websites were selected due to their popularity and their varied number of embedded objects.
To perform this experiment, we load the test websites from server to client 100 times each for both prefetching enabled and prefetching disabled. We use GNU Wget[10] as the client software, and record the difference between end time and start time for each retrieval.

3.2. MULTIMEDIA WEBSITES

Our second experiment is similar to the first. We have created several HTML documents with a number of embedded objects. One of our embedded objects is a multimedia object of length ***1 byte***. We use an embedded multimedia object of only ***1 byte*** because it is at the first byte that the media handler will be able to start processing the data. Since for continuous media playback, the data must be delivered at least as fast as the media player will require it, reception of the first byte signifies the start of uninterrupted playback.

The generated web pages also contain a number of images of the average size, given by [7] to be ***11.9 KB***. We use web pages containing ***0***, ***5***, ***10***, and ***25*** images. We serve these web pages from server to client *50* times each with prefetching disabled and *50* times with prefetching enabled,

recording the total elapsed time each time the page is served, and calculate the averages for prefetching and for standard HTTP. The web browser invokes the media handling plugin the moment that it detects that there is an embedded video within the web page, but until the first byte of the media file is received, there is nothing for the media handler. The media handler will begin working on the media file being received the moment the first byte is received, so the difference in the elapsed time for standard HTTP and for prefetching is the improvement in the time it takes for an embedded media object to begin playback on the client.

It should be noted that even though we have selected embedded objects of average size, that their size does not impact the results of prefetching. Prefetching acts on the time to deliver a request, which is independent of the file being requested. We use average file sizes to provide perspective for the overall reduction in delivery time.

4. RESULTS AND DISCUSSION

The results of our two experiments are described in sections 4.1 and 4.2.

4.1. NORMAL WEBSITES

For retrieval of normal web pages, we have obtained significant reduction in page load times.

Table 1: Normal Web Page Delivery Time (s)

Website	*Standard*	*Prefetching*	*Improvement*
Amazon	15.85	4.25	11.60
Aol	8.78	2.73	6.05
Mapquest	6.55	2.56	3.99
Wikipedia	3.72	1.44	2.28
Google	0.23	0.24	0.01

The websites tested contain differing numbers of embedded objects as shown in Table 2.

Table 2: Number of embedded objects

Website	*Number of Objects*
Amazon	60
Aol	34
Mapquest	28
Wikipedia	15
Google	1

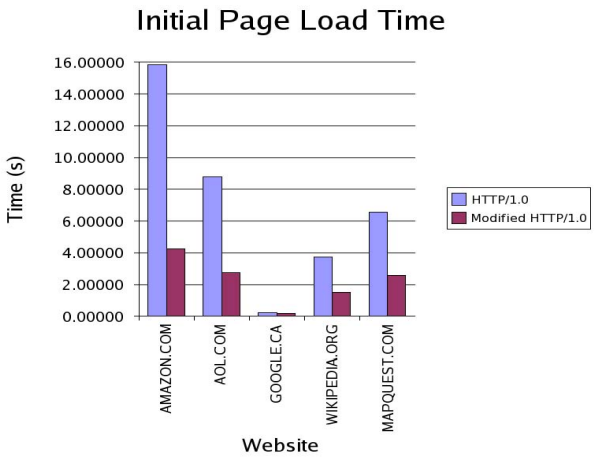


Figure 1: Improvement in web page delivery time

4.2. MULTIMEDIA WEBSITES

With our modified HTTP protocol, we are able to obtain a significant reduction in wait times for beginning retrieval of embedded objects of web pages.

With no embedded objects in addition to the multimedia file being tested, we are able to reduce wait time by 13 ms, or 21.64%. With 25 embedded objects, we are able to reduce wait time by 859 ms, or 20.76%.

With an increasing number of embedded objects, the improvement in wait time tends to increase. This is because prefetching eliminates the request time from every file requested.

When increasing the number of embedded objects, the proportional improvement in wait time tends to be close to constant. This is because the same request time is removed for each additional file transmitted.

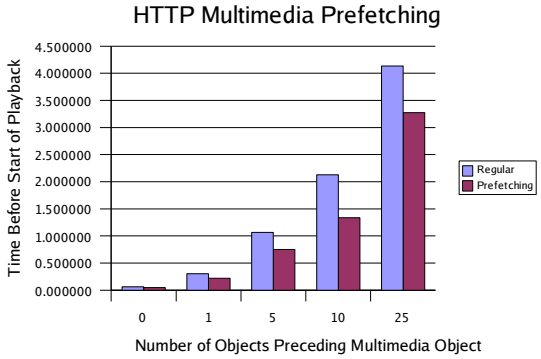


Figure 2: Improvement in Web Page Delivery Time

Table 3: Multimedia Web Page Delivery Time (s)

# of Objects	*Normal HTTP*	*Prefetching HTTP*	*Improvement*
0	0.060630	0.047508	0.013122
1	0.305684	0.219592	0.086092
5	1.067207	0.715699	0.315508
10	2.129906	1.337465	0.792441
25	4.132104	3.273403	0.858701

5. CONCLUSION

In this work we have improved the delivery time of objects embedded in HTML documents over HTTP connections.

In our test of websites without multimedia objects, we have determined that the improvement due to prefetching correlates with the number of objects as follows; the greater the number of embedded objects, the greater the improvement due to prefetching.

Our testing of generated web pages containing multimedia objects shows that prefetching allows multimedia objects to begin playback earlier than what would otherwise be experienced.

We have reduced the effects of network latency by reducing the number of requests made by the client to the server from one request for every object to one request for all objects.

6. FUTURE WORK

Our solution to improving the delivery of multimedia objects from HTTP server to client required the implementation of a custom light-weight proxy to be run on the client system. This proxy must cache data sent by the server until the data is ultimately requested by the client.

Many multimedia objects, such as videos, can be extremely large, making them impractical to cache. The delivery of very large files may be improved by retrieving the first part of the file by prefetching and requesting the balance of the file upon access. When a client requests a very large multimedia object from a proxy server, the proxy will begin delivering the start of the file while simultaneously requesting the rest of the file from the server. Before the client has consumed the entire cached portion of the file, the balance of the file has begun to arrive at the proxy server.

7. REFERENCES

[1] RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0. http://www.faqs.org/rfcs/rfc2616.html

[2] RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1. http://www.faqs.org/rfcs/rfc2616.html

[3] Abdolreza Abhari, Adam Serbinski, Miso Gusic, "Improving the Performance of Apache Web Server", 10th Communications and Networking Simulation Symposium (CNS'07), 2007

[4] Behrouz A. Forouzan, "Data Communications and Networking, Third Edition", 2004.

[5] "Streaming Media", Wikipedia, http://en.wikipedia.org/wiki/Streaming_Media

[6] Kevin Thompson, Gregory J. Miller, Rick Wilder. "Wide-Area Internet Traffic Patterns and Characteristics", MCI Telecommunications Corp., 1997.

[7] Andy King, "The Average Web Page", www.optimizationweek.com/reviews/average-web-page/

[8] IRCache Top 50 Origins, http://www.ircache.net/Statistica/Top-Fifty-Servers

[9] Seeking Alpha, "The 20 Most Popular Websites", available at: http://internet.seekingalpha.com/article/25309

[10] GNU Wget, http://www.gnu.org/software/wget, GNU Project, The Free Software Foundation.

[11] Abdolreza Abhari, Sivarama P. Dandamudi, and Shikharesh Majumdar. Structural characterization of popular web documents. *International Journal of Computers and Their Applications*, 9(1):15–24, March 2002.

[12] Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the 4th ACM International Conference on Information and Knowledge Management*, Baltimore, MD, 1995.

[13] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.