Submitted to INFOCOM'99 - please do not redistribute

Globally Progressive Interactive Web Delivery

Jeffrey M. Gilbert
University of California, Berkeley
Electrical Engineering & Computer Sciences
Berkeley, CA 94720-1770
gilbertj@EECS.Berkeley.EDU
Tel: (510) 215-6336
Fax: (510) 642-2739

Robert W. Brodersen
University of California, Berkeley
Electrical Engineering & Computer Sciences
Berkeley, CA 94720-1770
rb@EECS.Berkeley.EDU
Tel: (510) 642-1779
Fax: (510) 642-2739

Abstract - *This paper suggests that since web browsing is an interactive process and downloading a web page can take several seconds to several minutes over slow links, the information presented to the user during this time is important. We present new metrics and visualization techniques to illustrate and quantify web page loading. Given the insight afforded by the metrics, we propose a methodology to improve web access using a new technique, globally progressive interactive web delivery. This technique entails applying progressive coding to the document transmission process in its entirety, as well as allowing the user to explicitly direct link bandwidth to images of interest. This globally progressive interactive framework has been developed without modifying either existing web browsers or servers through the use of a web proxy and browser-side Java applets. The framework allows for both protocol and image compression research in a platform-independent manner. Directions of future work on integrating the architecture into existing web infrastructure for greater performance and scalability are discussed.*

1 Introduction

Delivery is everything. "The web experience" is much more than the web pages visited - it also encompasses the speed of access and the quality of information delivered. While browsing over a T3 line can be highly productive, web access over a dial-up or wireless modems often leads to the phenomenon known as the "World Wide Wait". Network congestion and bottlenecks within the Internet can also limit the gains of a high-speed last link. However, as will be shown, web performance over these slow links can be dramatically improved through efficient link scheduling and data compression tailored to web transmission. Although some previous work has accelerated web access through lossy image compression [9][15][30], this, by necessity, results some reduction in image quality. Transport protocol modifications designed to reduce the total web page delivery time [22][4][24] also improve web performance, but further gains can be achieved by combining image coding and networking techniques as described in this paper.

Web browsing is an *interactive* process where the user often has specific goals when browsing. The goals may be to read the text, glance at all of the pictures, or closely examine a single picture. In order to maximize the utility to the user, it is important to incorporate these goals as best as possible. Currently the level of interaction is limited to aborting the downloading of a page or suppressing all image downloads until they are explicitly requested. Thus, the user is at the mercy of the delivery system and must wait for the image or images of interested in be delivered. We propose a method for incorporating feedback to allow the user to guide the bandwidth utilization throughout the downloading process: by simply moving the mouse cursor into an image's window, the image will gain full use of link bandwidth.

Once new protocols have been designed, it is beneficial to prototype and deploy them in order to investigate operation in realistic circumstances and allow end-user evaluation. We present a design framework which allows new protocols to be prototyped without modifying either the browser or server, using a web proxy and Java applets. We have implemented our proposed system through this framework. Additionally, we discuss a more permanent manner in which the delivery methodology can be integrated into the existing web infrastructure to obtain greater performance and scalability.

1.1 Background / previous work

Web pages are multi-object documents. They consist of a main HTML object[1], which contains the text and formatting information, and zero or more additional objects such as inline images and Java

1. A page can also be composed of multiple HTML objects using frames.

applets. The main HTML object contains references to other objects by specifying their uniform resource locator (URL). These objects are retrieved individually by the web browser from the web server using the HyperText Transfer Protocol (HTTP).

HTTP/1.0 [5] uses a separate TCP connection to retrieve each object. Several concurrent connections are maintained between the browser and server to load multiple objects at once in order to hide TCP connection establishment and slow-start [16] delays. However, this increases overhead and network congestion, thus impacting scalability[30]. Typically the number of open connections is limited to about 4.

HTTP/1.1 [8] improves upon the delivery protocol through support of persistent connections [24] between the servers and browsers. By using a single connection to sequentially deliver all of the HTML and images in a web page, the connection establishment and slow-start delays can be amortized over the cost of the entire page, the connection overhead packets can be reduced, and better responsiveness to network congestion is achieved. HTTP/1.1 also supports compression of the HTML objects via the deflate coding of the public domain zlib compression library [11]. (This is a hybrid LZ77 [33] / Huffman coding [14]). When applied to HTML data, it can typically result in a greater than 3x reduction in the size of the HTML file [22].

TCP sessions [4][23] have been proposed to allow sharing of state between related TCP connections, such as those connecting the same host pair, at the transport layer. If any connection in the session experiences congestion, all connections can reduce their windows. In this way the network-friendly behavior of HTTP/1.1 can be obtained while decoupling sensitivity to losses between different image transfers. Additionally only the server-side TCP stack needs to be modified - no browser or server modifications are necessary.

The MUX protocol [21], a work in progress which is part of the next generation HTTP effort (HTTP-ng) [20], provides a method to layer lightweight multi-session delivery on top of a reliable stream-oriented transport such as TCP. The MUX protocol can be used by multiple applets to share the same transport stream as the main HTTP connection in an application independent manner.

Previous work in Web Acceleration

UC Berkeley's Transend [9][10], Intel's QuickWeb [15], and Spectrum's FastLane [30] improve upon web access over slow links by reducing image size via lossy compression[2]. Web proxies are used to transform the images through resolution and color reduction. Users of QuickWeb have noted that it can cause significant degradation of image quality [17] and the service has recently been discontinued. Use of FastLane also results in image quality degradation [3]. The three systems allow the user to explicitly load the original undegraded images but no further user interaction is supported. In contrast, the globally progressive interactive system described here does not cause reduction of the final quality of the images delivered, and allows greater user interaction.

2 Quantifying Web Page Downloading

Since downloading web pages can take from several seconds to several minutes using modem links, the information presented to the user during this period is critical. For instance, if it requires one minute to download a page, it is clearly better to have most of the text and images present after 10 seconds with the remaining filling in over the minute, than having nothing until the whole page appears at the one minute mark. Yet in both cases the total time to load the page is the same. **Thus the total time to load a given page is often not an accurate measurement of the utility of the loading process.**

In order to fully encapsulate the loading process as viewed by the user, it is necessary to visualize and quantify the loading of the constituent parts of the page, and not just the time to load everything. A *web-page loading graph* can be used to more precisely illustrate and quantify the effects of different transfer protocols on a typical web document. The CNN Interactive (tm) home page (http://www.cnn.com/) will be used as an example. The constituent objects are listed in Table 1 in order of their appearance and the shaded entries correspond to images that the user would have to scroll to when using a 1024x768 pixel web

2. Transend is also designed to operate with clients of varying computational capabilities and perform other types of content manipulations.

browser window because they are not initially visible.

TABLE 1. Contents of typical web page (taken from CNN Interactive - www.cnn.com). Shading denotes images that are not initial visible when viewed on a 1024x768 screen.

Object	Name	Size (bytes)		Object	Name	Size (bytes)
HTML	index.html	41,261		Image1	horoscopes120.gif	4,313
Image2	ad_info.gif	92		Image3	still_468x60.gif	9,374
Image4	cnnin_logo.gif	2,225		Image5	icon_arrow_left.gif	261
Image6	white.gif	35		Image7	roof_top.gif	1,205
Image8	search_infoseek.gif	441		Image9	right_corner.gif	102
Image10	top_ruins_ap.jpg	17,774		Image11	custom_top_clinton_church.jpg	6,112
Image12	video_transparent.gif	170		Image13	cnn_website.gif	2,980
Image14	web_services.gif	11,623		Image15	custom_arrow.gif	137
Image16	tv_generic.gif	1,291		Image17	health2.gif	2,124
Image18	thumbnail.jpg	2,580		Image19	week_in_review_icon.gif	986
Image20	diana.jpg	2,507		Image21	infoseek_logo.gif	555
Image22	pointcast.gif	6,983		Image23	red_468.gif	10,486

An example of concurrent HTTP/1.0-style loading

Figure 1 shows a web page loading graph of the document loaded using HTTP/1.0-style concurrent loading protocol over a constant 3KB/sec link (similar to a 28.8kbps modem without any transport latencies - future revision will incorporate the transport latencies). The objects in the page are listed on the vertical axis while time traverses the horizontal axis, with the lighter gray used for initially visible images. The bars indicate the time during which a particular object is being delivered.

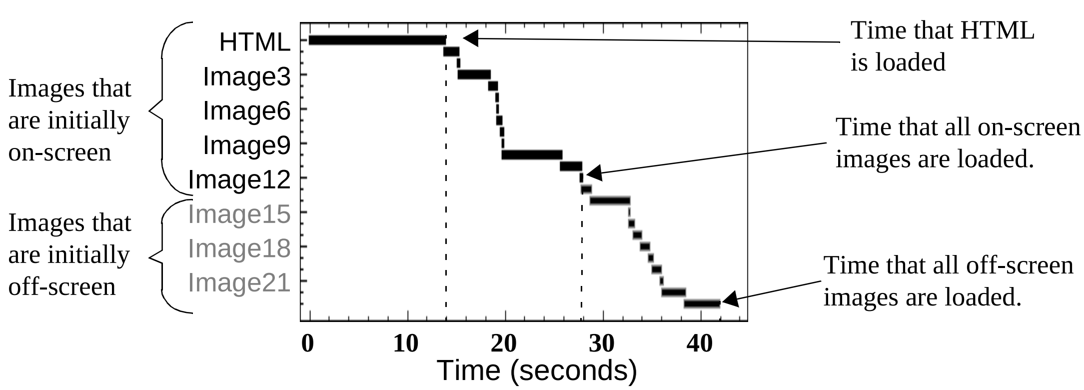

FIGURE 2. Web-page loading graph of sequential loading protocol

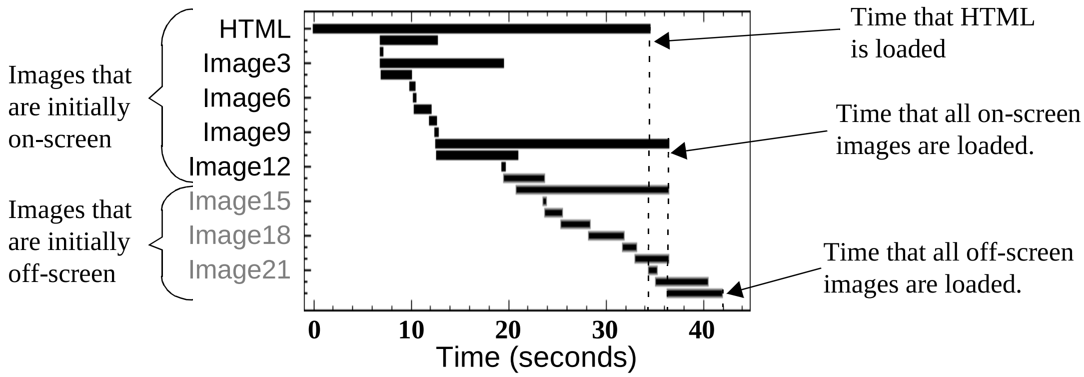

FIGURE 1. Example web-page loading graph using concurrent loading of up to 4 connections

In order to quantify the loading process, it is useful to consider the time that the HTML is loaded, the time that initially visible images are loaded, as well as the time to load the entire page. The time to load the HTML determines how long it will take to view the text on the page[3], and the time to load the initially visible images will determine when the user thinks that the page has loaded. In the example we can see that the HTML is loaded within 34.5 seconds, all visible images are loaded within 36 seconds, and the entire page is not loaded until 42 seconds have elapsed.

An example of sequential HTTP/1.1-style loading

The web-page loading graph for the same web document loaded using an HTTP/1.1-style *sequential loading protocol* is shown in Figure 2. Using the sequential loading, the total time to load the entire web document remains 42 seconds but the time to see the visible images has dropped to 28 seconds, simply by reordering the data being sent. Additionally, the time to see the text is reduced to 14 seconds - less than half of the time required using concurrent loading. Thus, despite the fact that the total loading time is the same as the previous case, the delivery orders have very different consequences for the user.

3. An exception to this rule is that some browsers require the headers of images whose sizes are not explicitly specified in image tags to be loaded before the text after the images can be displayed. Additionally the same sort of dependencies often occur with tables where they cannot be formatted until all image sizes are known. Both of these problems may be fixed in future releases of popular browsers.

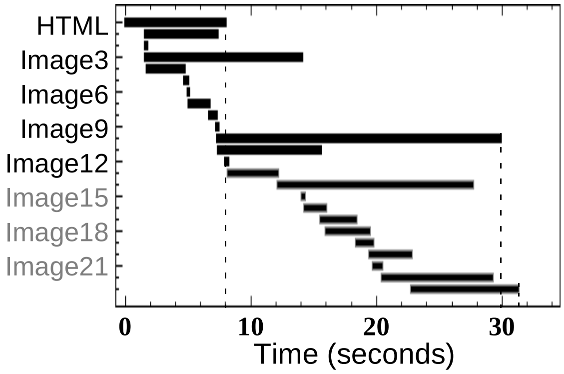

FIGURE 3. Web-page loading graph using concurrent loading and HTML compression

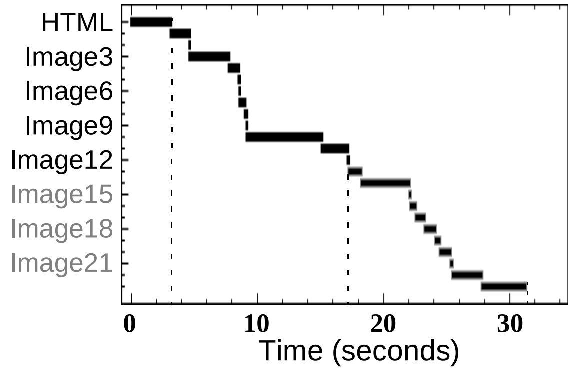

FIGURE 4. Web-page loading graph using sequential loading and HTML compression

HTML Compression

If HTML compression is used, the size of the HTML object in our example is reduced by about a factor of 4, from 41261 to 9950. The corresponding web-page loading graphs are shown in Figure 3 and Figure 4. While the total time to load the document is only reduced by about 25% from 42 seconds to 31.5 seconds, the time to see the text is reduced by a factor of 4 and the time to see the entire visible part of the document is reduced by about 20% and 40% for the concurrent and sequential modes respectively. However, the delivered document is identical to the original. These results are summarized in Table 2.

TABLE 2. Summary of concurrent vs. sequential loading

	Time to text	Time to complete visible document	Time to complete document
Concurrent loading (HTTP/1.0)	34.5 sec	36.0 sec	42.0 sec
Sequential loading (HTTP/1.1)	14.0 sec	28.0 sec	42.0 sec
Concurrent w/html compression	8.0 sec	30.0 sec	31.5 sec
Sequential w/html compression	3.0 sec	17.0 sec	31.5 sec

3 Globally Progressive Interactive Web Delivery

Once metrics and methodology are in place to examine web document delivery, we can investigate alternatives. Although sequential transmission is an improvement over concurrent delivery, using progressive and interactive techniques, significant improvements can be realized.

3.1 Globally Progressive Transmission

Progressive image transmission entails sending a layered coding consisting of a low quality version initially, followed by refinements. Thus, after the initial bytes are received, a complete, albeit non-aesthetically pleasing, version can be displayed, and as time progresses, the image quality improves. Progressive techniques are ideally suited for transmission of human-viewed data over band-limited links since they allow the user to quickly deduce the salient features and only need to wait for the full image load times to see full detail.

Existing progressive image formats

Progressive image formats are already common on the web. CompuServe's Graphical Interchange Format (GIF) [7] has an "interlaced" mode in which rows are sent in a progressive manner as shown in Figure 5. Portable Network Graphics (PNG) [26], a graphics format which has been formally accepted as a standard MIME type by the World Wide Web Consortium, has a mode that is progressive in both rows and columns as shown in Figure 6. This "doubly progressive" mode allows for better subjective quality after a smaller amount of data has been received. Both GIF's and PNG's progressive modes result in some loss in compression rate due to the increase in local entropy. PNG also enjoys a patent-free status and typically achieves 5%-25% better compression than GIF [26] by using zlib's LZ77 / Huffman compression instead of GIF's Lempel Ziv Welch (LZW) [34][32] compression and packing sub-byte pixels.

FIGURE 5. GIF Interlace order. Numbers indicate on which pass the data is sent.

1	1	1	1	1	1	1	1
4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2
4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4

FIGURE 6. PNG Interlace order. Numbers indicate on which pass the data is sent.

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

The Joint Picture Expert Group's JPEG [26] is a DCT-based coding technique which has a progressive mode that divides layers by spatial frequency and quantization level. The number of layers and their

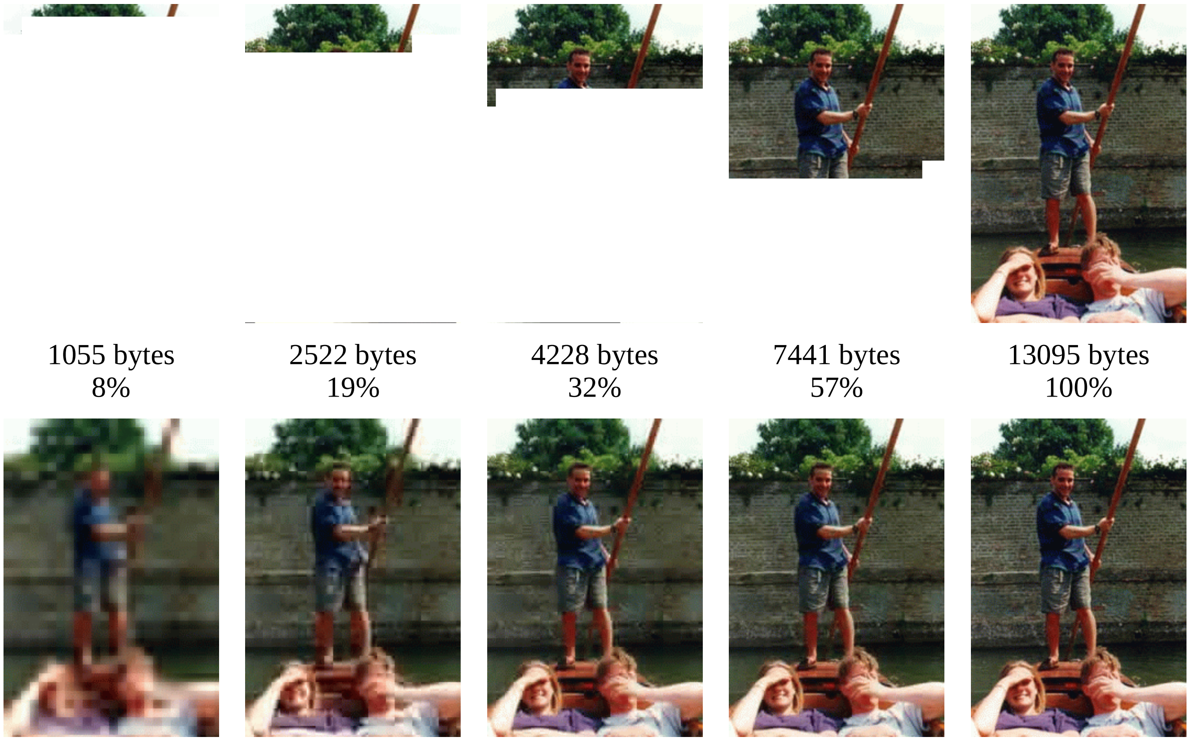

FIGURE 7. Example of JPEG (top) and PJPEG (bottom) showing display of incomplete data. Image was quantized to JPEG default level yielding 1.84 bits/pixel coding.

composition can be varied, and are specified in the image header. Figure 7 shows an example comparing JPEG's non-progressive and progressive (PJPEG) modes[4]. As can be seen, a very crude version of the image is available after receiving only 8% of the progressive data. After 19%, enough detail is available to easily discern the contents of the image. Subsequent data provides further detail, with the later layers only becoming noticeable under closer inspection. It is important to note that this is exactly what lossy image compression proxies such as QuickWeb, TranSend, and Fastlane rely upon - by performing lossy image compression they are removing some of the fine detail, similar to that in the end layers of PJPEG. While the degradation is not severe to cursory inspection, it does reduce the final image quality. However, by using progressive techniques, the end quality need not be sacrificed.

Progressive image formats are of limited benefit when using standard loading protocols since they are

4. The JPEG baseline quantization levels were used, resulting in a coding rate of 1.84 bits/pixel which, from an informal survey, is typical of JPEG images on the web.

locally progressive - each image is progressively coded but the document as a whole is not. Otherwise stated, the loading process itself is not progressive. Figure 8 and Figure 9 show web page loading graphs of locally progressive images loaded concurrently and sequentially. The lighter parts to the left correspond to coarser layers while the darker parts to the right correspond to refinements. For illustrative purposes, the layers have been divided according to the division of the image in Figure 7.

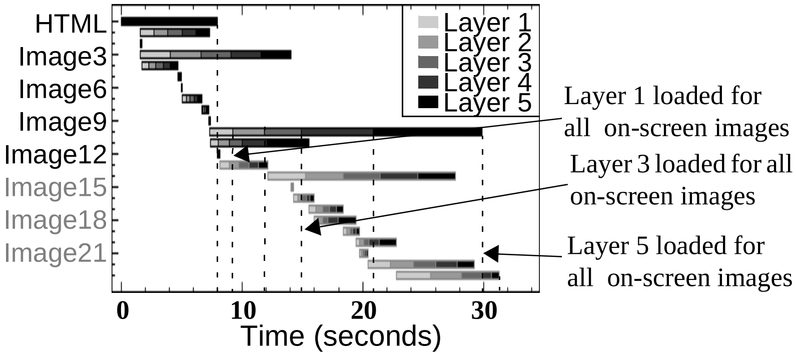

FIGURE 8. Simultaneous loading with locally progressive images (and HTML compression)

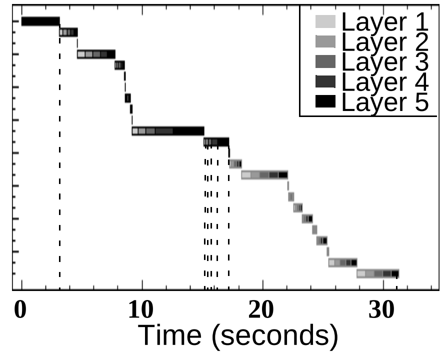

FIGURE 9. Sequential loading with locally progressive images (and HTML compression)

The amount of time to deliver enough of all of the images to see a certain number of layers in each image can be used to quantify the progressive delivery. As can be seen from Figure 8, the time to concurrently load at least the first layer of each of the images is about 9 seconds, to load the first two layers is about 12 seconds, three layers is 15 seconds, four layers is 21 seconds, and the time to load all layers for all visible images is still 30 seconds. Figure 9 shows that the layers in each image for the sequential case are loaded in rapid succession since the images have access to the full link bandwidth.

However, by using *globally progressive* loading, further benefits can be achieved. Globally progressive loading considers the whole document as a progressive object and seeks to display a coarse version followed by more refined versions. In the case of a web document, the HTML might be considered the coarsest layer since it is required to decode the rest of the document and conveys the bulk of the information. The next layer of the document would include the first layer of all images and subsequent layers of the document would include subsequent layers of each image. By using globally progressive loading, the user is very quickly presented the text and coarse versions of all visible images and can rapidly proceed to

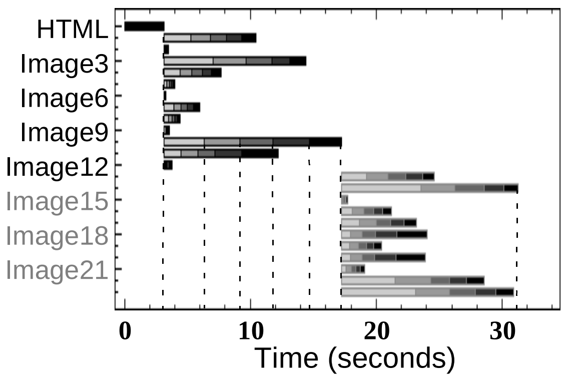

FIGURE 10. Globally byte-wise progressive loading (with HTML compression)

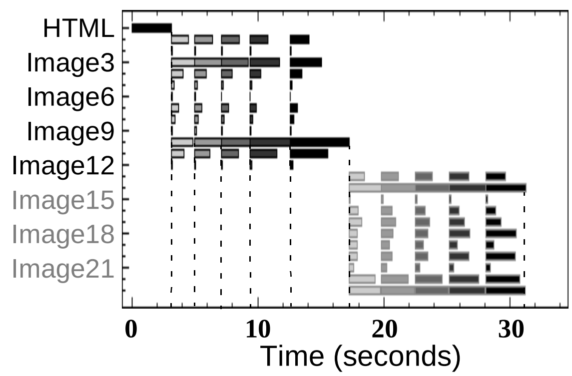

FIGURE 11. Globally layer-wise progressive loading (with HTML compression)

analyze the page's contents. This is particularly useful when combined with interactive loading described in the next section.

Figure 10 shows globally progressive loading using a byte-wise equality metric such that the scheduling attempts to keep the number of transmitted bytes of each visible image similar and Figure 11 shows globally progressive loading using a layer-wise equality metric dictating that a given layer in one visible image should not be loaded until the previous layers were loaded in all other visible images. While the layer-wise metric allows earlier layers to be loaded for all images sooner, the byte-wise metric is typically preferable since it allows link scheduling to be independent of image coding and also prevents large images from severely delaying smaller images.

Table 3 shows a summary of the comparison between locally and globally progressive loading. As can be seen, the time to load all visible layers is the same for the sequentially loaded locally progressive case as the globally progressive cases, but in the locally progressive case, the benefits of progressive loading are minimal. In the simultaneously loaded locally progressive case, there is a delay until the base layers of some of the later images are loaded, as well as a delay in finishing the images due to competition with images that are not visible. However, in the globally progressive cases, the text is shown very quickly, followed quickly by coarse versions of all images and then refinements.

If an image that is not initially displayed is made visible during the loading process, it is boosted to

TABLE 3. Summary of delivery methods and performances

Protocol			Timings						
HTML Compress	**Concur / Sequen**	**Progressive**	**Text**	**Visible Layer 1**	**Visible Layer 2**	**Visible Layer 3**	**Visible Layer 4**	**Visible All Layers**	**Complete Document**
No	Conc	No	34.5 sec	n/a	n/a	n/a	n/a	36.0 sec	42.0 sec
No	Sequen	No	14.0 sec	n/a	n/a	n/a	n/a	28.0 sec	42.0 sec
Yes	Conc	No	8.0 sec	n/a	n/a	n/a	n/a	30.0 sec	31.5 sec
Yes	Sequen	No	3.0 sec	n/a	n/a	n/a	n/a	17.0 sec	31.5 sec
Yes	Conc	Locally Progressive	8.0 sec	9.0 sec	12.0 sec	15.0 sec	21.0 sec	30.0 sec	31.5 sec
Yes	Sequen	Locally Progressive	3.0 sec	15.0 sec	15.3 sec	15.6 sec	16.3 sec	17.0 sec	31.5 sec
Yes	**Conc**	**Byte-wise Globally Progressive**	**3.0 sec**	**6.5 sec**	**9.0 sec**	**12.0 sec**	**15.0 sec**	**17.0 sec**	**31.5 sec**
Yes	**Conc**	**Layer-wise Globally Progressive**	**3.0 sec**	**5.0 sec**	**7.0 sec**	**9.5 sec**	**12.5 sec**	**17.0 sec**	**31.5 sec**

the same priority as the other visible images so that bandwidth is initially dedicated to only that image until it has loaded to the same degree as the other images. Then all visible images resume loading in unison. An example is shown in Figure 12. In this example, images 13 and 14 were previously off screen but then scrolled into view after about 9 seconds.

Globally progressive delivery is also well suited to take advantage of networks with variable Quality of Service (QoS). There is an implicit ordering of the importance of the data with the text being most important, the initial bytes or layers of images being a little less important, bytes or later layers being still less important and off-screen images being even less important. While it is currently used to prioritize data within the web connection, it could be used to prioritize data transmission across multiple connections, web or otherwise. For instance, if text over any web connection was given higher priority than images over any web connection then even as the network became congested, the text delivery performance would not suffer as much. Likewise, if off-screen images were given lower priorities, the downloading of long pages by some users would not hamper the interactive operation of other users. The prioritization could be used within a web server to ensure timely service of text and coarse image requests.

3.2 Interactive Operation

While globally progressive loading rapidly delivers coarse versions of all visible images, the user must still wait for all refinements, even if they are only interested in a single image in detail. However, by allowing the user to easily instruct the system which image they are interested in, this image can be loaded more quickly by dedicating all available link bandwidth to it. Since the layers are loaded in synchrony, the user is able to quickly determine which image or images are most interesting. One method for incorporating user image preference is to detect when the mouse cursor is inside a particular image, and give that image priority. Figure 13 shows the effect of selecting image 3 after its two first layers have been displayed. As can be seen, in this way, the entire refined image can be loaded by the 9 second mark, about half of the time required in the absence of user intervention. Further interactivity could include targeting

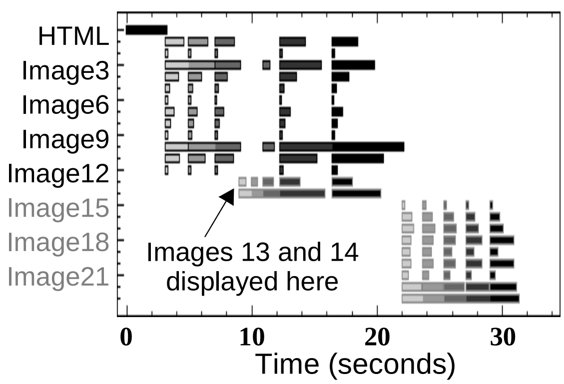

FIGURE 12. Globally progressive loading with images scrolled to during loading

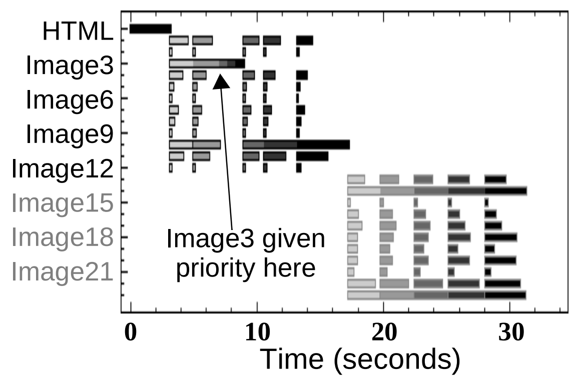

FIGURE 13. Interactive loading with image interactively selected by user

exactly which parts of the image are transmitted first. Explicit targeting may be useful in cases of large images where semantic quality requires high fidelity, such as maps.

When interaction is allowed, the amount of data buffered in the connection between the web server and web browser must be limited to allow rapid response. Once the user preference is detected and transmitted to the web server (or proxy), the high-priority image can be queued, but this data will not reach the browser until all other data queued in that connection has been delivered (unless out-of-band signalling or more sophisticated transport protocols are used). For instance, for a 28.8k baud modem connection, to obtain a one-second response time, at most 3.6K bytes can be buffered assuming no network congestion. The buffered data consists of the data queued in the kernel buffers as well as the packets queued in the network routers. Kernel-level scheduling of the images could be used to eliminate delay due to the former[25] while TCP window-size limiting would have to be used to reduce the later. Severely limiting the amount of queued data can reduce link utilization, particularly for high-bandwidth, high-latency links such as satellite links.

4 Transport protocol prototyping via web proxies and Java applets

In order to prototype our globally progressive interactive delivery scheme, we have designed a proxy-based architecture that allows full control over image delivery and display. This architecture can be used as a test-bed to develop and experimentally deploy a range of network protocols not restricted to HTTP or even TCP. Additionally, non-standard image compression techniques can be used over the link.

4.1 Overview

Motivation for prototyping by proxy

While altering web browsers and servers provides the greatest flexibility and performance when investigating protocol modifications, and thus is useful for final deployment, it has some drawbacks as a prototyping method. First, unless a state-of-the-art browser is modified, many of the features and compatibility that users are accustomed to will not be present. Additionally, deployment requires users to download and install a new browser. The proxy / applet implementation described here, allows users to take advantage of globally progressive interactive loading by simply changing a setting on their existing browser once. For the researcher, the proxy / applet architecture allows the development of new protocols without having to modify web browser code, allowing rapid prototyping that can take advantage of state-of-the-art browsers without having to commit to a single browser.

Despite these disadvantages, integration into an existing browser has some advantages in terms of performance and transparency of integration, which will be discussed in section 5. It is viewed as a viable end-solution for mass-deployment while the proxy operation described here is better suited for initial prototyping.

Proxy operation

While conventional web proxies can transform the content of the web pages, they must work within the confines of the HTTP protocol specifications. For instance, they cannot alter the number of connections that the browser opens, change the ordering of the requests, or respond to fine-grained user interactivity - thus prohibiting the implementation of a protocol similar to the one just described. However, through the use of Java applets to load and display the images on the page, much greater control is possible, both in terms of networking protocols and image compression techniques.

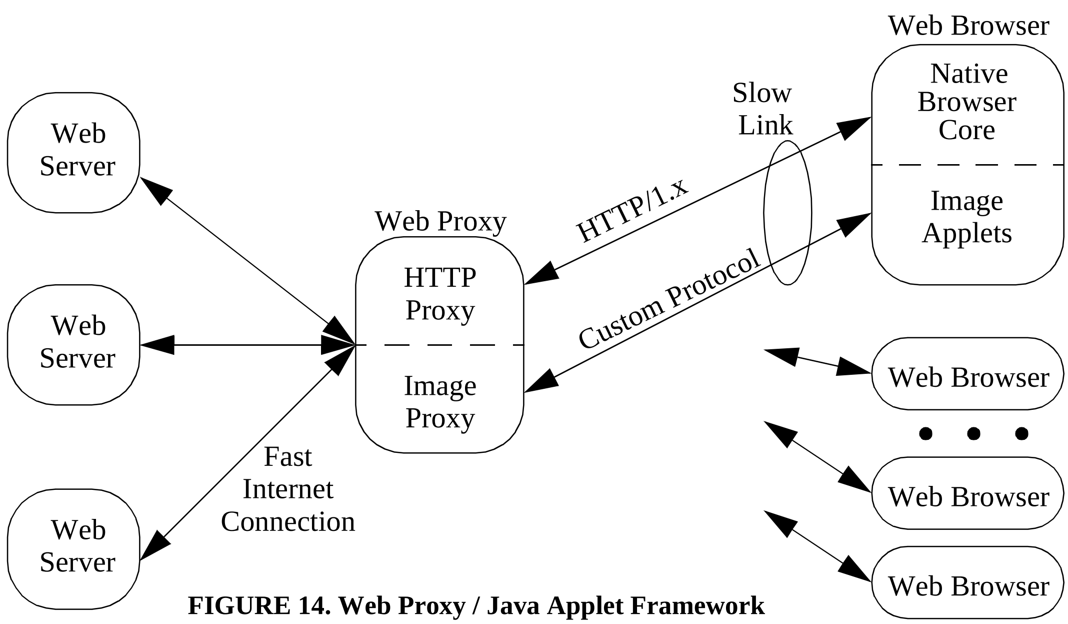

FIGURE 14. Web Proxy / Java Applet Framework

A block diagram of the framework is shown in Figure 14. The HTTP Proxy and Image Proxy are two parts of the Web Proxy. The HTTP Proxy transforms the HTML received from the web server as it sends it to the browser, and the Image Proxy sends the images to the browser's Image Applets. The Image Applets which render the in-line images are instantiated by the web browser in response to the modified HTML received from the HTTP Proxy. Further details on these three parts are given in the following sections.

Original HTML:

```
For my homepage, click <A HREF=/~me> here <IMG SRC=home.gif></A>
```

Modified HTML:

```
For my homepage click <A HREF=/~me> here
<A HREF="~me">
<APPLET CODE="SpeedImage.class" WIDTH=54 HEIGHT=39
   ARCHIVE="http://host.com/SpeedImage.jar"
   CODEBASE="http://host.com/">
   <PARAM NAME=imageHostPort VALUE=9999>
   <PARAM NAME=src VALUE="http://myhome.com/~me/home.gif">
   <PARAM NAME=href VALUE="http://myhome.com/~me">
</APPLET>
</A>
```

FIGURE 15. Example of HTML modification to embed image applets

4.2 HTTP Proxy Design

When the user requests a document, the request is sent to the web proxy instead of directly to the designated web server. The proxy then retrieves the requested pages and substitutes applet tags for image tags, as it sends the pages to the browser, causing the web page to appear with applets where the images were.

The HTML is sent from the HTTP Proxy to the browser using standard HTTP/1.1 protocols, allowing the use of persistent connections and HTML compression. The HTML is transformed by replacing the IMG tags which specify conventional images with APPLET tags which embed Java applets as shown in Figure 15. The applets are supplied the URL of the image to display as well the host name and port of the Image Proxy. If the width and height are not specified in the IMG tag, they must be determined by prefetching the image and decoding its header since the applet dimensions *must* be specified in the APPLET tag (unlike IMG tags where the dimensions *should* be specified.)

Streaming HTML conversion

One important technique used in both the HTTP Proxy and the Image Proxy is that of *streaming conversion*. Streaming conversion entails converting both images and HTML text "on-the-fly" whenever possible, instead of first retrieving the entire objects, and then processing them. This is important if the response from the web server is slow due to network congestion or server loading since waiting for the loading to complete can in fact double the total latency. For the HTTP Proxy, streaming conversion means that the HTML has to be transformed and compressed on the fly. While compression and simple textual substitution would not be problematic, incorporating image size information in applet tags requires asynchronous retrieval and parsing of image headers to determine their sizes.

Link Scheduling

The HTTP Proxy performs limited link scheduling by tracking the amount of data outstanding in HTML and non-HTML (typically image) links. It does not send data on non-HTML links until the amount of HTML data outstanding is below a given threshold. The image proxy, described next, will also suppress sending custom image data until the amount of HTML data outstanding is below a given threshold in order to effect the text prioritization needed for globally progressive transmission as described in section 3. The fine-grain link scheduling between the images takes place in the Image Proxy.

4.3 Image Proxy Design

The Image Proxy is responsible for retrieving the images from the web servers, transforming them, and sending them over the custom managed link to the Image Applets. A new Image Proxy is spawned when a new set of Image Applets connect to the Web Proxy. Image Applets send requests for new images' data as well as messages dynamically specifying the relative priorities of the images. The Image Proxy then sends chunks of image data to the Image Applets for display. Images are given unique IDs used for communication manner similar to the multiplexing scheme used in MUX [21].

Image Conversion

As the images are retrieved, their type is determined from the HTTP meta-data. Type-specific conversion is performed to generate progressive versions. JPEG images are converted to Progressive JPEG (PJPEG) format. This conversion is lossless and does not significantly effect compression rate. GIF images are converted to a lossless format similar to interlaced PNG.

Additionally, conversion from GIF to PJPEG is attempted and the PJPEG image is used whenever a high-quality lossy JPEG conversion yields results in a reduction in size compared to the PNG-like coding. By keeping the JPEG quality setting high, discrete-tone images not well suited to lossy compression will compress less compactly with JPEG than with a lossless coding such as GIF or PNG, and thus the lossless coding will be used. However, photographic images are often stored on web sites in lossless GIF while lossy JPEG coding dramatically reduces the size of the image while not resulting in any perceptible degradation. Since progressive coding is used, a high quality setting can be used while still allowing quick delivery of a coarse version.

Streaming image conversion

As described previously, an important aspect of effective proxy operation is forwarding data as soon as possible to reduce the amount of time that the user has to wait to see something. However, conversion to progressive formats requires the entire image to be present before the conversion can occur. In order to minimize perceptual delay, the original data is streamed on to the browser until the entire image is loaded and converted. A switch to the progressive coding is done when advantageous. For the case of JPEG to PJPEG conversion, a heuristic of requiring less than half of the data being sent is used. For the case of GIF to PJPEG conversion, the size of the PJPEG data has to be the number of bytes of the GIF image remaining. The Image Applets flush old non-progressive data when they switch to progressive codings in the middle of an image.

Link Scheduling

The Image Proxy explicitly manages the link to the Image Applets. A strict priority round-robin system is used. The image priorities are determined by the Image Applets (described next) as dictated by the globally progressive interactive delivery scheme, and communicated to the Image Proxy. Within a priority class, images sending processed data (progressified images) are given priority to those sending unprocessed data since the unprocessed data may have to be flushed, as previously described, if the processing proves to be advantageous.

4.4 Image Applet Design

Image Applets request the data for their images from the proxy and display it. They are indistinguishable from images they are replacing, responding to mouse clicks to follow web links as standard images do. The Image Applets also respond to keyboard commands to create new windows which are copies of the images as well as zoom and pan within the images. Thus the document is an active entity that can be manipulated.

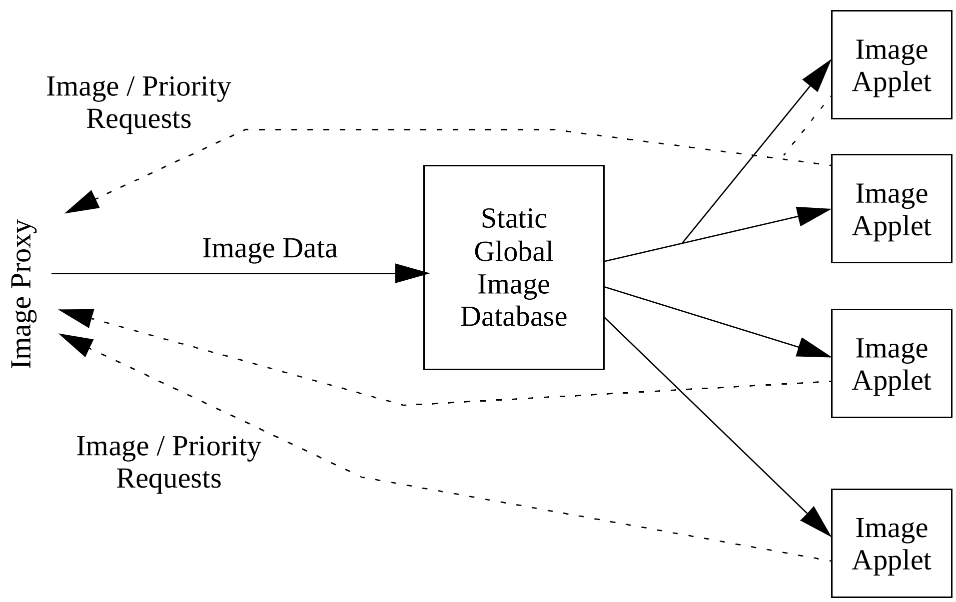

FIGURE 16. Image Applet Design

Figure 16 shows the internal architecture of the Image Applets. There is one applet per image on the page, though the images communicate through shared static objects. Static Java objects are shared among all applets running in the same Virtual Machine - i.e. all applets running in the same browser. In particular, a shared image database is used to track which images have been loaded already and allows the same image to be shown in multiple applets without requiring the image to be loaded more than once. The image database also allows the loading of the images to be centralized via a single Java thread which contacts the Image Proxy and manages all browser-proxy communication.

Problems with a Java implementation

Although using Java for the browser-side hooks proved to be an effective way to rapidly prototype the architecture, there are some limitations which may render it suboptimal as an end-solution. First, since Java is not yet fully mature, its support varies from browser to browser, although it should improve over time. In particular, the lack of Just-In-Time (JIT) compiled implementations under Unix makes it too slow to fully realize the speedups of the globally progressive interactive delivery. Additionally, it is difficult to correctly display transparent GIFs since transparent applets are not supported. However, the Web Proxy / Java Applet framework is quite useful for rapid prototyping and easy deployment. Integration into a browser is being investigated as a long term solution as described in the next section.

5 Future Work: Integration with Existing Web Infrastructure

While the Web Proxy / Java Applet architecture described in the previous section was useful to evaluate and optimize the globally progressive interactive web delivery, further gains can be achieved by incorporating the methodology into existing web servers, proxies, and browsers by building upon current HTTP protocols and related work described in section 1.1.

The existence of free public domain source code for high-performance, widely deployed browsers and servers is integral to this mission. The Apache web server / proxy [1] is the most widely deployed web server on the Internet, and is supported on a number of different operating systems. The Netscape Corporation [19] recently made source code to their popular Navigator browser available[18]. The functionality of the Image Applets can be directly incorporated into the browser while the functionality of the web proxy can be integrated into the Apache web server / proxy. The use of a modified server as well as proxy will allow further scalability as described in the next section. Interoperability will be achieved through protocol negotiation on connection.

5.1 Link multiplexing and scheduling

One of the core requirements for globally progressive interactive web delivery is a transport mechanism that will allow many images to be delivered simultaneously in an explicitly managed manner without consuming too many network resources. While HTTP/1.1 range requests could be used, this would result in a large amount of overhead, particularly if the requested chunks were small. Additionally it would require receiver-based multiplexing which would be more difficult, especially if data streaming is used.

A more general mechanism such as MUX can be used to support transmission of multiple images over a single link. One disadvantage of MUX using TCP is its use of a single buffered connection, which forces multiplexing to occur before buffering. This can result in additional buffering delay, impacting interactive switching of image priorities as described in section 3.2.

Alternatively, TCP sessions could be employed if explicitly prioritized queuing is added [25]. TCP sessions have the advantages of application independence and removing false-dependencies across images. However, the use of TCP sessions requires server kernel modifications and may result in greater kernel overhead and resource utilization since a full TCP connection is established for each image.

5.2 Transparent Content Negotiation

Another requirement of the globally progressive interactive web delivery is the ability to send modified versions of web objects. For instance, when a particular object which is stored as a standard non-progressive JPEG image is requested, it must be delivered as a progressive JPEG image instead. Yet it is important that the original can also be retrieved, as a web delivery system must allow for verbatim object transmission for certain applications. Additionally, there may be various trade-offs between image delivery speed and quality that the user would like to effect such as the lossy conversion of GIF images to PJPEG images. The underlying protocol should be able to specify which conversion should occur and how such decisions should be made. While HTTP/1.0 provides very crude content negotiation via "accept" headers, it lacks sufficient expressive capabilities, which can lead to name-space conflicts. Transparent

Content Negotiation, however, as described in [8] allows for a more flexible, powerful mechanism which allows "variants" of an object to be described and named differently than the original object.

Integration of an automatic, yet parameterizible, content conversion mechanisms into a web proxy or server has additional benefits. Web sites typically maintain images in the most compatible formats possible - GIF and JPEG. While this choice assures that the greatest number of users will be able to access the images, it can result in worse compression than more advanced methods such as PNG, JBIG [2][29], wavelets [28], or other coding techniques yet to be developed or standardized. However, if the web servers or proxies could automatically negotiate with the browsers to determine the best mutually supported format, and the servers or proxies could automatically convert images into this format, if necessary, then the best of both words can be achieved: the latest image coding techniques can be used while older browsers can continue to operate using the conventional formats. By using the scalable conversion caching techniques described in the next section, the computational requirements of the image conversions can be dramatically reduced. Transparent content negotiation can also be used to specify variants of the HTML objects transformed for quicker transmission or display. An example would be to include the sizes of images with previously sizeless IMG tags in order to speed rendering.

5.3 Scalability through Server / Proxy Caching of Processed Data

An important aspect of the progressive image formats is that a single image coding will work for both high-bandwidth and low-bandwidth links, allowing the processed data to be cached and reused. In this way, documents which are accessed multiple times need not be converted multiple times, amortizing processing costs over multiple accesses, reducing computations load and increasing scalability. When coupled with streaming conversion, it allows conversions to be delayed when the load on the server gets high, and yet assure that performance is never worse than without the conversions.

While the use of proxies that convert the image formats and transport protocols allows for enhanced performance when retrieving pages from conventional web servers, it still requires the proxies to convert each object the first time it is retrieved[5]. If several users access a given web site via different web proxies, the objects on this site have to be converted by each of the proxies (although each object needs only be converted by each proxy once unless it is flushed out of the cache).

However, by integrating the functionality of the proxy into the web server, the server / proxy can process the objects on the site only once, regardless of how many geographically separated users access the objects. Thus, each object need only be converted once after it is created or updated at the server. The necessary "cache" size is simply the size of all of the objects on that site. Additionally, by transmitting the converted objects over the Internet, the delay of the wide-area access is mitigated by the globally progressive interactive delivery. Otherwise stated, the entire connection from the remote browser to the server is considered the slow link.

The same scalability for dynamically created objects cannot be obtained since they cannot be cached. In this case, the server can either pass these objects through directly, convert them before sending, or use streaming conversion to achieve the best of both worlds. Fortunately, most images are not dynamically created even at sites that use dynamically created content.

References

[1] The Apache Group, "The Apache Web Server Project." http://www.apache.org.

[2] Arps, R.B. and T.K. Truong, Comparison of International Standards for Lossless Still Image Compression. *Proceedings of the IEEE*: 82:889-899, June 1994.

[3] Backman, Dan, "Spectrum Lets You Dial In On The FastLane," Network Computing Online, http://techweb.cmp.com/nc/907/907sp4.html.

[4] Balakrishnan, Hari, and Venkat Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements." Proceedings of IEEE Infocom '98. March 1998.

[5] Berners-Lee, Tim, R. Fielding, H. Frystyk., "Informational RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0," MIT/LCS, UC Irvine, May 1996. http://www.w3c.org/Protocols/rfc1945/rfc1945

5. Here object conversion encompasses both image recompression conversion and HTML compression.

[6] Boutell, T., T. Lane et al. "PNG (Portable Network Graphics) Specification," W3C Recommendation, October 1996, RFC 2083, Boutell.Com Inc., January 1997. http://www.w3c.org/Graphics/PNG.

[7] CompuServe Incorporated, "Graphics Interchange Format - Version 89a," 1987-1990. http://www.w3c.org/Graphics/GIF/spec-gif89a.txt

[8] Fielding, R., J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, "RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1," UC Irvine, Digital Equipment Corporation, MIT. http://www.w3c.org/Protocols/rfc2068/rfc2068

[9] Fox, Armando, Steve Gribble, Yatin Chawathe, and Eric Brewer, "The Transend Service," http://transend.cs.berkeley.edu/about

[10] Fox, Armando, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Elan Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," Proceedings Seventh International Conference on Architectural Support for Programing Languages and Operating Systems (ASPLOS-VII), October 1997, Cambridge, MA.

[11] Gailly, J. and M. Adler, "ZLIB documentation and sources," available at ftp://ftp.uu.net/pub/archiving/zip/doc

[12] "GVU's 8th WWW User Survey," Georgia Tech. December 1997. http://www.gvu.gatech.edu/user_surveys/survey-1997-10/

[13] Holtman, Koen, "Transparent Content Negotiation in HTTP" http://gewis.win.tue.nl/~koen/conneg/

[14] Huffman, D.A., "A Method for the Construction of Minimum Redundancy Codes," Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101.

[15] Intel Corporation, "QuickWeb homepage". http://www.intel.com/quickweb

[16] Jacobson, Van, "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM '88, p. 314-329. Stanford, CA, August 1988.

[17] McWilliams, Brian, "Intel Announces Server-Side Browser Accelerator," PC World Online, January 20, 1998. http://www.pcworld.com/news/daily/data/0198/980120192859.html

[18] Mozilla Home Page, http://www.mozilla.org/

[19] Netscape Home Page, http://www.netscape.com/

[20] Nielsen, Henrik Frystky, "Hypertext Transfer Protocol - Next Generation" homepage. http://www.w3c.org/Protocols/HTTP-NG/

[21] Nielsen, Henrik Frystky, "MUX Overview" W3C Architecture Domain. http://www.w3c.org/Protocols/MUX

[22] Nielsen, Henrik Frystky, et al., "Network Performance Effects of HTTP/1.1, CSS1, and PNG." W3C NOTE-pipelining-970624. June 24, 1997. http://www.w3c.org/Protocols/HTTP/Performance/Pipeline.html.

[23] Padmanabhan, Venkat and Randy H. Katz, "Addressing the Challenges of Web Data

Transport," unpublished.

[24] Padmanab, V.N. and J. Mogul, "Improving HTTP Latency," Computer Networks and ISDN Systems, v.28, pp.25-35, Dec. 1995.

[25] Padmanabhan, Venkat - personal correspondence.

[26] Pennebaker, William B. and Joan L. Mitchel, JPEG Still Image Data Compression Standard, New York: Van Nostrand Reinhold, 1993.

[27] Roelofs, Greg., "PNG (Portable Network Graphics) Home Page" http://www.cdrom.com/pub/png

[28] Said, Amir, and William A. Pearlman, "An Image Multiresolution Representation for Lossless and Lossy COmpression," SPIE Symposium on Visual Communications and Image Processing, Cambridge, MA, Nov. 1993.

[29] Sayood, K., *Introduction to Data Compression*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1996.

[30] Spectrum Information Technologies homepage, http://www.spectruminfo.com

[31] Spero, Simon E, "Analysis of HTTP Performance Problems," July, 1994. http://www.w3c.org/Protocols/HTTP/Performance/Pipeline.html

[32] Welch, T., "A Technique for High-Performance Data Compression," Computer, June 1984.

[33] Ziv J. and A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[34] Ziv, J. and Lempel, A., "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, March 1977.