*To appear in 2nd ACM International Workshop on Wireless Mobile Multimedia (WOWMOM) 1999, in conjunction with Mobicom 99*

# Factoring a Mobile Client's Effective Processing Speed Into the Image Transcoding Decision

Richard Han
IBM T.J. Watson Research Center
rhan@watson.ibm.com

## ABSTRACT

An image transcoding proxy decides whether to transcode an image fetched from the Web based either on the criterion of reducing the overall response time (store-and-forward proxies), or the criterion of avoiding buffer overflow (streamed proxies). In this paper, we introduce a new parameter, namely a mobile client's effective processing speed, into the analytical formulation of both transcoding decisions, and study the practical importance of this parameter when transcoding is applied for standard PDA clients. CPU-intensive operations like image decompression, colorspace conversion and scaling can together add excessive delay to the perceived response time when performed on a mobile client that has severely limited processing capability. Under certain conditions, a transcoding proxy that sends GIF or JPEG images can incur greater delay due to decompression on a PDA client than a better informed transcoding proxy that chooses instead to send bit-mapped equivalents that incur little to no client-side decoding delay. We designed three experiments that partitioned image processing functions between a proxy and a standard PDA in order to assess the importance of a client's CPU limitations on the image transcoding decision. First, images were fetched by a standard Web browser that decompressed GIF's on the PDA. This browser also enforced scaling on every image to fit within the PDA's small screen. Second, we added a transcoding proxy that pre-scaled images, thereby bypassing the browser's scaling function but still requiring the browser to decompress the scaled GIF's. Third, we migrated both scaling and decompression off of the PDA on to the proxy. The proxy pre-scaled as before, and also transcoded GIF's to grayscale bitmaps that required no client-side decompression. We measured response times from each of these three experiments and quantified how much improvement in response time can be achieved when a proxy assists a CPU-limited PDA by performing some or all CPU-intensive image processing tasks on the transcoding proxy.

## Keywords

Transcoding, proxy, partitioning, mobile, PDA, CPU, image processing.

## 1. INTRODUCTION

Due to slow response times, wireless Web access via Personal Digital Assistants (PDA's) can be a frustrating experience for the end user. One proposal for improving the response time of wireless Web browsing involves placing a proxy between the Web server and Web browser in order to aggressively compress or transcode images so that download times may be reduced [1][2][3]. In previous work, an analytical framework was presented describing under what conditions a proxy should transcode each image [1]. Many factors affected the image transcoding decision, including the current bandwidth on both the server-proxy and proxy-client links, as well as the tolerable delay and desired severity of compression induced by transcoding. Two types of proxies were identified: store-and-forward proxies wait to receive an entire image before transcoding and wait to send until the entire image has completed transcoding; streamed proxies begin transcoding as soon as part of the image has been received at the proxy and begin sending as soon as part of the image has been transcoded. For store-and-forward proxies, the criterion for whether to transcode was based on whether the overall response time was reduced. For streamed proxies, the criterion for whether to transcode was based on whether buffer overflow could be avoided at both the entering and exiting buffers to the proxy.

One criterion omitted from the analytical frameworks developed previously for both store-and-forward and streamed proxies was the effective processing speed of the mobile client. Some mobile clients, such as laptops, may have sufficient processing speed to perform with reasonable delay the tasks of image decompression, HTML parsing, and layout required by Web browsing. Other mobile clients, such as handheld PDA's, can have raw processing speeds an order of magnitude or more slower than laptops, due to power, cost, and other considerations. In addition, software inefficiencies in the browsing application written for the PDA, such as inefficient image processing operations or excessive memory copying, can introduce substantial delay -- on the order of minutes as measured in this paper -- in the overall response. A proxy that is aware of the "effective" processing speed of the PDA, including software inefficiencies and hardware speed, can better decide what CPU-intensive image processing operations should be performed on the proxy instead of the mobile client, so that the overall response time is reduced in comparison to a proxy which ignores effective decoding times on the CPU-limited handheld.

In this paper, the effect of client processing speed on the image transcoding decision is studied both analytically and practically. In Section 2, we update the analytical formulation concerning whether to transcode or not for store-and-forward proxies. In Section 3, we revise the streamed proxy's formulation. In Section

4, we analyze the practical effect of CPU limitations combined with browser inefficiencies on the image transcoding decision. Image processing functions are partitioned between a proxy and a mobile PDA client and response times are measured for various cases to determine whether migrating some or all CPU-intensive image processing operations on to the proxy from the PDA achieves a reduction in response time.

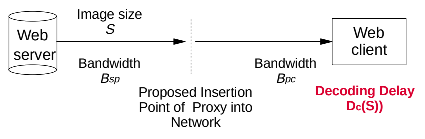


**Figure 1: Model of standard Web image retrieval in the absence of a transcoding proxy.**

## 2. STORE-AND-FORWARD IMAGE TRANSCODING

We first analyze the impact on the overall download time of inserting a store-and-forward transcoding proxy between the Web server and the Web client. First, consider the model outlined in Figure 1 of standard Web image retrieval in the absence of a proxy. Let $T_o$ denote the download time of sending a Web object of size $S$ (bits) from the Web server to the Web client in the absence of transcoding We conceptually divide the network into two segments based on the proposed insertion point of the proxy. Define the bandwidth or overall bit transmission rate from the Web server to the proposed proxy insertion point as $B_{sp}$, and similarly define the bandwidth from the proposed insertion point to the Web client as $B_{pc}$. For the purposes of the following discussion we assume that caching is not supported at the proxy.

The download time $T_o$ from Web server to Web client in the absence of transcoding consists of the sum of three terms. First, $D_{prop}$ is the propagation latency from the server to the client, i.e. the time required for the first bit of the image to propagate from the server to the client. Second, a Web image incurs a transmission delay equal to the spread in time between arrival of its first and last bits. Let $\min(B_{pc},B_{sp})$ denote the bottleneck bandwidth between the client and the server. In the absence of a proxy, the first and last bits of an image will be spread in time by $\frac{S}{\min(B_{pc},B_{sp})}$. This spread corresponds to the effective transmission time of the image over the concatenated server-to-proxy-to-client connection.

A third and new component that we introduce into the analytical framework is the time $D_c(S)$ required to decode and display an image on the mobile client. This factor models the Web client, e.g. a Web-enabled PDA, as a black box, and measures only the cumulative delay, or equivalently the "effective" processing speed of the mobile client. Both the raw CPU speed as well as measured software inefficiencies in the Web browsing application and operation system are accounted for by $D_c(S)$. As we shall see, this factor can introduce substantial latency and can therefore affect the image transcoding decision. Consequently, the overall image download time in the absence of a transcoding proxy can be expressed as:

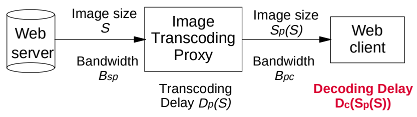


**Figure 2: Model of store-and-forward image transcoding proxy.**

$$T_0 = D_{prop} + \frac{S}{\min(B_{pc},B_{sp})} + D_c(S) \qquad (1)$$

Next, a store-and-forward transcoding proxy is inserted between the Web server and Web client, as shown in Figure 2. Let $T_p$ denote the download time of sending the same Web image from the server through a store-and-forward transcoding proxy and then onward to the Web client. $T_p$ consists of the sum of five terms. First, the server-to-client propagation latency $D_{prop}$ experienced by the image is the same as defined earlier, given the same network transmission path. Second, the image download time from the server to the proxy is given by $\frac{S}{B_{sp}}$. Third, the proxy introduces a delay of $D_p(S)$, which is the time required to transcode the image. Fourth, the image download time from proxy to client is given by $\frac{S_p(S)}{B_{pc}}$. Finally, we introduce a new component influencing the download time, namely the decoding and display time $D_c(S_p(S))$ of the transcoded image on the mobile client. This factor is similar to $D_c(S)$, except that $D_c(S_p(S))$ measures the latency incurred by decoding and displaying the *transcoded* image rather than original image. Consequently, the overall image download time through a store-and-forward transcoding proxy can be expressed as:

$$T_p = D_{prop} + D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} + D_c(S_p(S)) \qquad (2)$$

Transcoding will reduce response time only if $T_p < T_o$. That is,

$$D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} + D_c(S_p(S)) < \frac{S}{\min(B_{pc},B_{sp})} + D_c(S)$$ , or

$$D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} + [D_c(S_p(S)) - D_c(S)] < \frac{S}{\min(B_{pc},B_{sp})} \qquad (3)$$

The above inequality precisely characterizes when transcoding will reduce response time, and therefore is the key expression used by the transcoding proxy to determine whether each incoming image should be transcoded, how much compression is needed, and, indirectly, to what compression format an image should be transcoded. Except for $S$, the size of the original image which can be determined from the content-length header of HTTP response message, the rest of the parameters in the above inequality need to be predicted when the image arrives at the proxy and before initiating transcoding. In particular, prior work has shown the complexity of predicting $D_p(S)$ and $S_p(S)$ [1].

Let us next consider several special cases arising from Inequality 3. First, suppose $B_{pc} > B_{sp}$, i.e. the Internet backbone's server-proxy connection is the bottleneck. In this case, Inequality 3 reduces to

$$D_p(S) + \frac{S_p(S)}{B_{pc}} + D_c(S_p(S)) < D_c(S) \qquad (4)$$

If the effective processing speed of the mobile client is slow enough, i.e. the right hand side $D_c(S)$ of Inequality 4 is large enough, than there is sufficient margin for transcoding to be of some use; there is sufficient freedom to adjust the left-hand side parameters $D_p(S)$, $S_p(S)$, and $D_c(S_p(S))$. However, as mobile clients become faster due to inevitable software and hardware improvements, $D_c(S_p(S))$ converges toward $D_c(S)$ and it becomes difficult to adjust the left-hand side parameters without degrading the image beyond what is permissible.

Second, consider the more typical case when the proxy-client access link is the bottleneck, i.e. $B_{pc} < B_{sp}$. In this case, store-and-forward proxy-based transcoding is useful only if

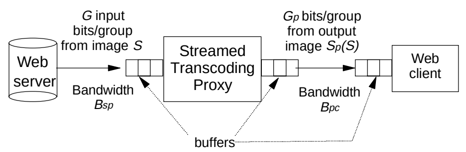
$$D_p(S) + \frac{S_p(S)}{B_{pc}} + D_c(S_p(S)) < D_c(S) + \frac{S}{B_{pc}} - \frac{S}{B_{sp}} \qquad (5)$$

Finally, let us consider the third case in which transcoding is placed at the server, and no proxy is present between the server and client. For example, a Web site may chose to place a transcoding proxy directly in front of its Web server, rather than at some intermediate node in the network. This situation is neatly summarized by setting $B_{sp} = \infty$. Consequently, the image transcoding decision's inequality reduces to the following:

$$D_p(S) + \frac{S_p(S)}{B_{pc}} + D_c(S_p(S)) < D_c(S) + \frac{S}{B_{pc}} \qquad (6)$$

## 3. STREAMED IMAGE TRANSCODING

A streamed transcoding proxy can output a partial image even before the entire image has been read into the transcoder. This gives streamed transcoding an advantage in terms of response time over store-and-forward transcoding since the transcoding delay $D_p(S)$ is virtually nonexistent, or at least very small. We defer our discussion of delay to first focus on the problem of avoiding buffer overflow with streamed proxies.



*G* input bits/group from image *S*
Web server
Bandwidth $B_{sp}$
Streamed Transcoding Proxy
$G_p$ bits/group from output image $S_p(S)$
Bandwidth $B_{pc}$
Web client
buffers

**Figure 3: System model of a streamed image transcoding proxy and mobile client.**

If we model the input as a stream of bits, then the transcoder takes a small group of $G$ input bits (from a total image of size $S$) and transcodes them to a small group $G_p$ of output bits (eventually producing a total transcoded image of size $S_p(S)$), where $G < S$, and $G_p < S_p(S)$. As shown in Figure 3, the streamed proxy system is modeled as having three buffers of relevance: a proxy input buffer, a proxy output buffer, and a mobile client input buffer - the latter buffer introducing a new factor not included in prior work.

To avoid overflowing the RAM input buffer between the arriving bits and the proxy, each input group must be processed before the next input group arrives, i.e. groups must be transcoded/emptied out of the input buffer at a rate faster than they enter. Equivalently, the group transcoding delay must be less than the interarrival times between groups, namely $\frac{D_p(S)}{S/G} < \frac{G}{B_{sp}}$, or

$$D_p(S) < \frac{S}{B_{sp}} \qquad (7)$$

To avoid overflowing the buffer between the proxy and the proxy-client transmission link, the transcoded output image group size $G_p$ must be transmitted over the proxy-client link at a rate faster than output groups are produced by the transcoder. Equivalently, the time to transmit each output group must be less than the interarrival time between output groups. Assuming that Inequality 7 is satisfied, then the interarrival time between output groups is the same as the interarrival time between input groups. Therefore, to avoid buffer overflow, we require $\frac{G_p}{B_{pc}} < \frac{G}{B_{sp}}$, or

$$\gamma > \frac{B_{sp}}{B_{pc}} \qquad (8)$$

where $\gamma$ = group image compression ratio $G/G_p$, which we assume to be on average equivalent to the overall image compression ratio.

In this paper, we introduce a new factor affecting the analysis of streamed transcoding, namely the impact of a mobile client's limited processing speed on the image transcoding decision. At the client, avoiding buffer overflow requires that the mobile decode and display transcoded image groups faster than transcoded groups arrive at the client. Equivalently, the time to decode and display a proxy group $G_p$ should be less than the time to generate a proxy group $G_p$. Assuming Inequality 8 holds, the interarrival times of proxy groups is the same ultimately as the interarrival times of groups at the input to the proxy. Therefore, $\frac{D_c(S_p(S))}{S_p(S)/G_p} < \frac{G}{B_{sp}}$, or rearranging terms,

$$D_c(S_p(S)) < \gamma \cdot \frac{S_p(S)}{B_{sp}}, \quad \text{or}$$
$$D_c(S_p(S)) < \frac{S}{B_{sp}} \qquad (9)$$

If the mobile client is especially slow, then Inequality 9 tells us that even if the streamed transcoding proxy satisfies its buffer overflow requirements, the downstream mobile client will not be able to process the input groups fast enough to avoid buffer overflow. In this case, no transcoding should be performed.

In summary, the streamed image transcoder should only perform transcoding from a buffer overflow standpoint when Inequalities 7, 8, and 9 are satisfied.

If the server-proxy link is the bottleneck, i.e. $B_{sp} < B_{pc}$, then Inequality 8 reduces to $\gamma > N$, where $N$ is a number less than 1. Normally, the compression ratio is always greater than 1, so Inequality 7 will always be satisfied. Hence, only Inequalities 7 and 9 must be satisfied in order for transcoding to not be disadvantageous. In fact, when the server-proxy link is the bottleneck, Inequality 8 could be interpreted as providing an upper bound on the ratio of expansion allowed for a transcoded image, namely $\frac{1}{\gamma} < \frac{B_{pc}}{B_{sp}}$. Expansion of an image may occasionally be necessary when format conversion is mandatory, e.g. the mobile client only supports one image decoding format. The above inequality allows us to determine when such format conversion will increase the chances of buffer overflow, and when format conversion will not cause buffer overflow. For example, if $B_{sp}$ = 1 bps, $B_{pc}$ = 2 bps, and $G$ = 1 bit, then Inequality 8 says that the output group $G_p$ can expand to a maximum of 2 bits.

If the proxy-client link is the bottleneck, i.e. $B_{sp} > B_{pc}$, then Inequality 8 says that the image compression ratio $\gamma$ must be greater than the ratio of server-proxy to proxy-client bandwidths in order for transcoding to be worthwhile. In addition, Inequalities 7 and 9 must still be satisfied.

Note that Inequalities 7, 8 and 9 are tight bounds that assume that the buffer must never be allowed to overflow. Looser constraints may be derived given that images are of finite-length, rather than the continuous stream assumed in the analysis. More relaxed constraints would permit more time for transcoding and/or allow less aggressive compression.

Returning to the topic of delay, our analysis of buffer overflow provides the intuition on how to measure latency for the streamed proxy. Assuming that Inequalities 7, 8, and 9 are enforced, then the input arrival rate of image groups is preserved at each buffer throughout the path, so that the spread between the first and last image bits is always $\frac{S}{B_{sp}}$. Next, we can examine the delay introduced for the first image bit. The first bit accumulates the same propagation delay $D_{prop}$ as in the previous section. In addition, there is a small component of transcoding delay $D_G$ introduced by the streamed proxy due to processing of the first group of $G$ bits in the stream. Finally, the mobile client also introduces a small delay $D_{Gp}$ while processing the first group of $G_p$ bits that it receives in the stream. The overall download time $T'_p$ for a user in the streaming case will be given by

$$T'_p = D_{prop} + \frac{S}{B_{sp}} + D_G + D_{Gp} \qquad (10)$$

Streamed transcoding will reduce response time when $T'_p < T_0$, namely

$$\frac{S}{B_{sp}} + D_G + D_{Gp} < \frac{S}{\min(B_{pc}, B_{sp})} + D_c(S) \qquad (11)$$

Since the quantities $D_G$ and $D_{Gp}$ are typically very small, then we can approximate Inequality 11 with the following inequality:

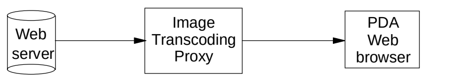
$$\frac{S}{B_{sp}} < \frac{S}{\min(B_{pc}, B_{sp})} + D_c(S) \qquad (12)$$

When the server-proxy backbone link is the bottleneck, i.e. $B_{sp} < B_{pc}$, then Inequality 12 is always satisfied and streamed transcoding is always beneficial in terms of reducing delay. When the proxy-client access link is the bottleneck, i.e. $B_{pc} < B_{sp}$, then Inequality 12 is again always satisfied.

In summary, when Inequalities 7, 8, and 9 are enforced, then Inequality 12 is always satisfied and streamed transcoding always reduces the response time.

## 4. PRACTICAL MEASUREMENTS

To understand whether proxy-based transcoding can reduce the overall response time for images delivered to mobile clients with limited effective processing speed, we conducted three experiments that partitioned the image processing functions between a PDA client and a transcoding proxy. The concept of partitioning of an application's functionality between a mobile device and the wired network has been mentioned in the literature [4]. Figure 4 summarizes the three cases we considered for partitioning of a browser's image processing functionality. For our first case, we measured the overall response time for fetching a single image from a Web server for display on a commercially available PDA client (a Palm IIIx with Palm OS v3.1) running a commercially available Web browser (HandWeb 2.0.2). No transcoding proxy was involved in this standard scenario of Web browsing. The browser performed scaling, image decompression, and rendering (colorspace conversion to the PDA's frame buffer format). For the second experiment, we migrated the scaling function alone to a transcoding proxy. The proxy pre-scaled images to fit within the dimensions of the PDA's screen so that



| Case | Image Transcoding Proxy | PDA Web browser |
|---|---|---|
| I. | --- | Scaling, Decompression, & Rendering |
| II. | Scaling | Decompression & Rendering |
| III. | Scaling & Decompression | Rendering |

**Figure 4: Partitioning of image processing functions between a client PDA's Web browser and a transcoding proxy can be used to reduce image download delays. In our study, Case III resulted in the fastest download times.**

the browser no longer had to scale on the PDA and could therefore focus exclusively on the tasks of image decompression and rendering. For the third case, we migrated both scaling and image decompression to the transcoding proxy. The proxy transcoded each image to its equivalent uncompressed bitmapped format. This latter format precisely matched the PDA's screen depth, so that the image could be written directly to the frame buffer and no decoding was necessary on the client. Transmitting bitmaps to the PDA effectively migrated the task of decompression off the PDA to the proxy. Since the commercial browser was unable to view the transcoded bitmapped format, we developed our own image viewer/browser for this PDA capable of displaying the raw bitmaps sent from the proxy.

In our experiments, the PDA was connected to the Internet via a serial cable rate-limited to 19.2 kbps.[1] For the 19.2 kbps trials, a Windows RAS server served as the gateway to the Internet. The Windows RAS server ran on an unloaded 200 MHz Pentium Pro Intellistation Z Pro running Windows NT 4.0 on 64 MB RAM. Our commercial PDA contained a 16 MHz 2.7 MIPS processor and a 160x160 screen at 2 bits/pixel grayscale depth. The browser response time for each image was obtained by directing the browser to fetch the specific image URL, and then by measuring manually the roundtrip latency. The commercial browser was only able to decode GIF images, and not JPEG images. In those experiments that required transcoding, the transcoding proxy ran on an unloaded 200 MHz Pentium Pro Intellistation Z Pro, running Windows NT 4.0 on 128 MB RAM. The proxy's machine was connected to the Windows RAS server's host by a 16 Mbps intranet LAN. Table 1 lists the images used in our trials as well as their individual characteristics. A scaling factor of $s$=0.$x$ represents a reduction of $x$% in both the horizontal and vertical dimensions. Each image's scaling value was chosen so that image would just fit within the PDA's 160x160 screen.

**Table 1. Test images and their properties.**

| Image | Com-pressed size (KB) | Area X x Y | Scaling to fit within 160x160 screen | Progressi-vity |
|---|---|---|---|---|
| Calvin.gif | 19.0 | 600x197 | s=0.25 (150x50) | interlaced |
| News1.gif | 4.38 | 155x145 | s=0.9 (140x131) | interlaced |
| Map.gif | 6.97 | 473x378 | s=0.3 (142x114) | non-interlaced |
| Enya.gif | 97.5 | 357x450 | s=0.3 (108x135) | non-interlaced |
| Index.gif | 29.3 | 510x270 | s=0.3 (153x81) | non-interlaced |

In Table 2, we present manually timed measurements of the overall response time for fetching a single image from a Web server for display on a PDA client running a Web browser. The measurements are representative numbers taken from multiple trials for each image; precise averages were not calculated. For this particular browser, our initial observation was that each image took *three to five minutes* to fully decode and display (see Case I column in Table 2). Each image was slowly rendered onto the screen, such that the user could observe each pixel row being drawn (for non-interlaced GIF's) at a glacial pace in raster scan fashion, left to right, top to bottom. In addition, the commercial browser enforced scaling of each image to fit within the dimensions of the screen; we could not find a way to disable scaling.

Our next objective was to determine the source of the excessive latency. Communications factors (e.g. a bandwidth bottleneck, or TCP flow control that might throttle the throughput due to small PDA receive buffers, or excessive packet loss causing TCP retransmissions, etc.) were considered, but ultimately our investigation led to CPU limitations (e.g. image processing, or inefficiently written browser, etc.) as the source of the latency. Our prior experience with image processing libraries suggested that image decompression was highly compute-intensive, and moreover that poorly written algorithms (decompression and/or scaling) can introduce significant delay. To test the hypothesis that image processing and/or browser-related inefficiencies were the primary causes of the delay, we decided first on the partial step of migrating the scaling function alone to the proxy. Each GIF image would be pre-scaled to fit within the screen's dimensions, so that the PDA's browser could at least avoid having to perform scaling on each image. The browser would still be left to perform decompression and rendering.

**Table 2. Measured response times for Web images using a commercial browser on a commercial PDA (with and without pre-scaling by a transcoding proxy).**

| Image | Predicted download time of original GIF @ 19.2 kbps | Response time @ 19.2 kbps (via proxy, No transcod-ing) | Case I. Response time @ 19.2 kbps (no proxy) | Case II. Response time @ 19.2 kbps (via proxy, proxy pre-scales GIF's) |
|---|---|---|---|---|
| Calvin.gif | 7.92 sec | 3 min | 3 min | 8 sec ($s$=0.25) |
| News1.gif | 1.83 sec | 40 sec | 40 sec | 18 sec ($s$=0.9) |
| Map.gif | 2.90 sec | 3 min 20 sec | 3 min 20 sec | 15 sec ($s$=0.3) |
| Enya.gif | 40.6 sec | 5 min 15 sec | 5 min 10 sec | 18 sec ($s$=0.3) |
| Index.gif | 12.2 sec | 3 min 10 sec | 3 min 10 sec | 15 sec ($s$=0.3) |

The measured response times resulting from migration of the scaling function to the proxy are shown in the Case II column of Table 2. These measurements indicate a dramatic reduction in delay, from three to five minutes down to ten to fifteen seconds. Pre-scaling reduced delay in several ways. First, pre-scaling of all GIF's by the transcoding proxy bypassed the commercial browser's internal scaling algorithm. Prior experience suggested to us that a scaling algorithm that is not optimized for speed, e.g. each scaled pixel is interpolated from neighboring pixels, can add significant delay. Second, pre-scaling resulted in the browser's

[1] Image response times were also measured over a 56 kbps handheld modem and a Ricochet wireless LAN modem. Preliminary results suggest similar behavior in terms of delay, but the measurements were not yet complete at the time of this paper's writing.

decompression algorithm having to process less image data on the PDA. A scaling factor of *s*=0.3 implies that the decompression algorithm only needs to process 9% or about 1/11 of the amount of data that originally had to be decoded. For each of the images, the factor by which the pre-scaling reduced the amount of data corresponded roughly with the factor by which the measured response times were reduced, e.g. scaling of *s*=0.3 for *map.gif* reduced the data by a factor of eleven, while the measured response time was reduced by a similar factor of about thirteen from 3 min 20 sec down to about 15 sec. Consequently, the delay savings due to pre-scaling on the browser appears to be more a result of reducing the amount of data that must be decompressed and processed by the browser, rather than a result of avoiding the scaling function on the PDA's browser. However, more study is needed to fully verify this conclusion.

The dramatic reduction in delay brought about by the migration of scaling to the transcoding proxy suggests that even a partial partitioning of browsing functions towards the proxy can bring large benefits to a mobile client, without requiring any modifications to the mobile client's software. If the application name and version, the OS name and version, and the processor name and speed can be communicated to the transcoding proxy, then the proxy will be in a position to understand the liabilities of the mobile client's complete system. This full knowledge can be used to select the appropriate transcoding functions and appropriate format for each transcoded image so that the overall response time can be minimized. While it is true that users can always install a newer faster browser for their PDA, buy a PDA with a better OS, or find PDA's with ever faster processors, the proxy will always retain the capability to adapt to whatever collective software/hardware limitations are posed by the mobile client's system, a feature that is especially useful for legacy mobile systems.

**Table 3: Measured response times for Web images transcoded by a proxy to 2-bit grayscale (with and without scaling) and displayed by a custom image browser.**

| Image | Size in KB of 2 bpp bitmap | Predicted download time of 2 bpp bitmap @ 19.2 kbps | Response time @ 19.2 kbps & no scaling | Case III. Response time @ 19.2 kbps w/ scaling to fit screen |
|---|---|---|---|---|
| Calvin.gif | 29.5 | 12.3 sec | 20 sec | ~3 sec (*s*=0.25) |
| News1.gif | 5.62 | 2.34 sec | ~5 sec | ~5 sec (*s*=0.9) |
| Map.gif | 44.7 | 18.6 sec | 30 sec | ~4 sec (*s*=0.3) |
| Enya.gif | 40.2 | 16.7 sec | 25 sec | ~5 sec (*s*=0.3) |
| Index.gif | 34.4 | 14.3 sec | 20 sec | ~4 sec (*s*=0.3) |

In our third experiment, both scaling and decompression were transferred to the proxy from the PDA, and a custom browser was built for the PDA. The transcoding proxy converted GIF and JPEG images into 2-bit grayscale images by first transforming input images into a colorspace representation that included a luminance component (e.g. YIQ, or YUV). The two most significant bit planes of the luminance component were then used to create the 2-bit grayscale bitmap sent to the PDA. More advanced dithering algorithms were not applied in this study. All scaling operations for fitting images to the PDA's screen were performed on the proxy, rather than on the PDA. Each 2-bit grayscale bitmap received by our custom browser was then written directly to the PDA's 2-bit deep frame buffer, without requiring any additional decoding, scaling or colorspace conversion on the PDA. Our custom browser permitted images to be displayed even if clipped by the limited screen dimensions, providing a way to scroll/pan images in all directions. Images were not displayed until the full bitmap was received. Rendering of images was whole and instantaneous, unlike the slow row-by-row rendering observed for the commercial browser. Timing was stopped as soon as the image appeared on screen, though some of the image may not have been visible due to clipping by the PDA's screen.

Table 3 summarizes the response times measured for fetching Web images that were pre-scaled and/or transcoded by a proxy to 2-bit grayscale and then displayed by our custom image browser on the commercial PDA. When the proxy performed only transcoding of images to 2-bit grayscale, response times were on the order of 20 seconds. When the proxy performed both transcoding to 2-bit grayscale as well as pre-scaling, then the response times were further reduced to four to five seconds (Case III of Table 3). These values represent a reference target

While our previous example of pre-scaling in Case II demonstrated that delay could be significantly reduced without requiring any modifications to the PDA's commercial browser, the Case III measurements indicate that response times can be further reduced by permitting modifications to the PDA's browser to incorporate low-complexity decoding techniques. One way for the commercial browser to achieve delays of four to five seconds would be to add support for grayscale bitmaps and patch any other software inefficiencies within the browser. Adding support for other image decoding formats, such as vector quantization, that achieve an intermediate degree of compression yet retain low-complexity decoders may lead to even lower delays, depending on the tradeoff between lower download times due to increased compression and higher decompression times.

Optimizing the existing browser decoders without adding any new formats is another avenue to improved performance. Our next step is to see how much delay performance can be enhanced by implementing optimized GIF and JPEG decoding on the PDA. As part of this effort, we need to better understand the complexity of GIF and JPEG decoding. These studies should also be able to determine how much savings in delay can be achieved by partitioning some of the GIF and JPEG decoding to execute at the proxy.

Preliminary analysis suggests that GIF decoding times should not contribute more than ten seconds in latency. GIF decompression is based on decoding a dictionary of encoded words [5]. Previous measurements of GIF->JPEG transcoding, which includes GIF decoding, on a 200 MHz system have shown collective latencies of no more than a few hundred milliseconds [1]. Extrapolating down to our 16 MHz PDA, which is about an order of a magnitude slower, yields an estimate of GIF decoding delays of ten seconds or less. Since a variety of factors could invalidate this extrapolation, the estimate is recognized as risky. Another rough estimate of GIF decoding times can be obtained from Table 2 (Case II, GIF decoding on a commercial browser). If the Case II delays are entirely attributed to GIF decoding, rather than to other browser inefficiencies, then GIF decoding should take no more than about ten to twenty seconds. on this PDA This roughly confirms our earlier extrapolated values. More analysis of the computational costs (add, multiply/shift, memory copy) of GIF decoding is needed in order to predict GIF decompression times on other PDA architectures. Fast hardware GIF decoding has been studied in the literature [6]. Additional GIF transcoding techniques have also been analyzed [7].

As a cautionary note, for certain GIF's, no amount of optimized GIF decoding can reduce the overall response time below that incurred by transcoding to the equivalent 2-bit grayscale bitmap of the GIF. For the image *enya.gif*, the download time alone of the unscaled compressed GIF (40.6 sec from Table 2) exceeds the overall response time (download delay plus transcoding time) for the equivalent unscaled 2-bit grayscale bitmap (25 sec from Table 3). In this case, no degree of optimization can make GIF decoding, whether optimized or not, can achieve faster delivery than transcoding to the bitmap format. Transcoding to a raw bitmapped representation can improve the response time for certain images, regardless of the CPU speed of the PDA.

Preliminary analysis suggests that JPEG decoding times should not contribute latency on the order of minutes. Previous measurements of JPEG->JPEG transcoding on a 200 MHz again show latencies on the order of several hundred milliseconds [1]. Extrapolating down to our slower PDA again yields a risky estimate of JPEG decompression time of roughly ten seconds or less. In the case of JPEG, it also possible to partition JPEG decompression into smaller discrete components: inverse DCT, inverse quantization, and inverse Huffman+RLC decoding [8][9]. JPEG is consequently a candidate for partitioning some of its image processing complexity off of the client device on to the proxy in order to improve delay performance. More analysis of JPEG is necessary to predict JPEG decoding delay and partitioned performance on other PDA architectures. Low-complexity proposals for JPEG decoding have been analyzed in the literature [8][10].

The general trend we observed for our experimental setup in comparing Tables 2 and 3 was that the response time kept improving as more functionality was migrated off the PDA client and onto the transcoding proxy. If a browser supports multiple image decoding formats of varying complexity, then a transcoding proxy is in a position to choose the lowest-complexity output format supported by the browser that minimizes delay. In particular, the proxy will need to calculate the overall download delay from Equation 2 for each output format, make sure that this delay satisfies the transcoding Inequality 3, and then pick the minimum-delay output format. The choice of output format and degree of pre-scaling will influence the factors $D_c(S_p(S))$ and $S_p(S)$.

.

## 5. CONCLUSION

The analytical framework for determining whether a proxy should transcode a given image was made more complete by incorporating the additional factor of a mobile client's effective processing speed. For store-and-forward image transcoding proxies, transcoding is only invoked when response time is reduced. The inequality that evaluated the transcoding was modified to consider whether the client was slow or fast. For streamed image transcoding proxies, the buffer overflow conditions governing when transcoding should be invoked were modified to include the client's effective processing speed. When all buffer overflow conditions were met, it was shown that streamed transcoding always reduced the overall response time. To understand the influence of the client's processing ability on the transcoding decision, we devised three experiments that partitioned image processing functionality between a transcoding proxy and a standard PDA client, and then measured the image fetch delays over a modem link. When no transcoding was performed, and the PDA client was forced to perform scaling, decompression, and rendering, we found response times to be on the order of minutes. When the transcoding proxy pre-scaled images, thereby freeing the browser to perform only GIF decompression and rendering, the delay was reduced to multiple tens of seconds. When the proxy pre-scaled and transcoded images to raw bitmaps, thereby migrating scaling and decompression tasks from the PDA's browser to the proxy, response times were reduced to less than ten seconds. These experiments demonstrate that a proxy that understands the CPU limitations of a PDA can dramatically reduce the response times experienced by an image by migrating some or all CPU-intensive tasks from a slow PDA client to a transcoding proxy.

## 6. ACKNOWLEDGMENTS

I wish to thank Jim Rubas of IBM T.J. Watson for his initial observation on the lengthy delay of PDA browsing, as well as his work on incorporating scaling into the transcoding proxy. I wish to thank Todd Mummert for his development of an HTTP-compliant image viewer for the PDA.

## 7. REFERENCES

[1] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, J. Rubas, "Dynamic Adaptation In an Image Transcoding Proxy For Mobile Web Browsing," *IEEE Personal Communications*, vol. 5, no. 6, Dec. 1998, pp. 8-17.

[2] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, "Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives", *IEEE Personal Communications*, vol. 5, no. 4, Aug. 1998, pp. 10-19.

[3] J. Smith; R. Mohan; C. Li, "Content-based Transcoding of Images in the Internet," *Proceedings of the International Conference on Image Processing (ICIP)*, 1998.

[4] T. Watson, "Application Design for Wireless Computing," *Workshop on Mobile Computing Systems and Applications*, Dec. 1994, pp. 91-94.

[5] M. Nelson, J. Gailly, *The Data Compression Book, Second Edition*, M&T Books, 1996.

[6] C. Su, C. Yen, J. Yo, "Hardware Efficient Updating Techniques for LZW CODEC Design," *IEEE International Symposium on Circuits and Systems (ISCAS)*, June 1997, pp. 2797-2800.

[7] N. Memon, R. Rodila, "Transcoding GIF Images to JPEG-LS," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 3, Aug. 1997, pp. 423-429.

[8] W. Pennebaker, J. Mitchell, *JPEG Still Image Data Compression Standard*, Chapman & Hall, 1993.

[9] S. Chandra, C. Ellis, "JPEG Compression Metric as a Quality Aware Image Transcoding", *USITS* 1999.

*[10]*K. Lengwehasatit, A. Ortega, "DCT Computation Based on Variable Complexity Fast Approximations," *Proceedings of ICIP*, 1998.