

Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing

Richard Han, Pravin Bhagwat, Richard LaMaire,
Todd Mummert, Veronique Perret, and Jim Rubas

Contact: {rhan,pravin}@watson.ibm.com

Abstract

Transcoding proxies are used as intermediaries between generic World-Wide-Web (WWW) servers and a variety of client devices in order to adapt to the greatly varying bandwidths of different client communication links and to handle the heterogeneity of possibly small-screened client devices. Such transcoding proxies can adaptively adjust the amount by which a data stream is reduced, using an aggressive lossy compression method (e.g., an image becomes less clear, text is summarized, etc.) We present an analytical framework for determining whether to transcode and how much to transcode an image for the two cases of store-and-forward transcoding as well as streamed transcoding. These methods require prediction of transcoding delay, prediction of transcoded image size (in bytes), and the estimation of network bandwidth. We discuss methods of adaptation based on fixed quality as well as fixed delay (automated/dynamic transcoding). We conclude with a description of the practical adaptation policies that have been implemented in our adaptive image transcoding proxy.

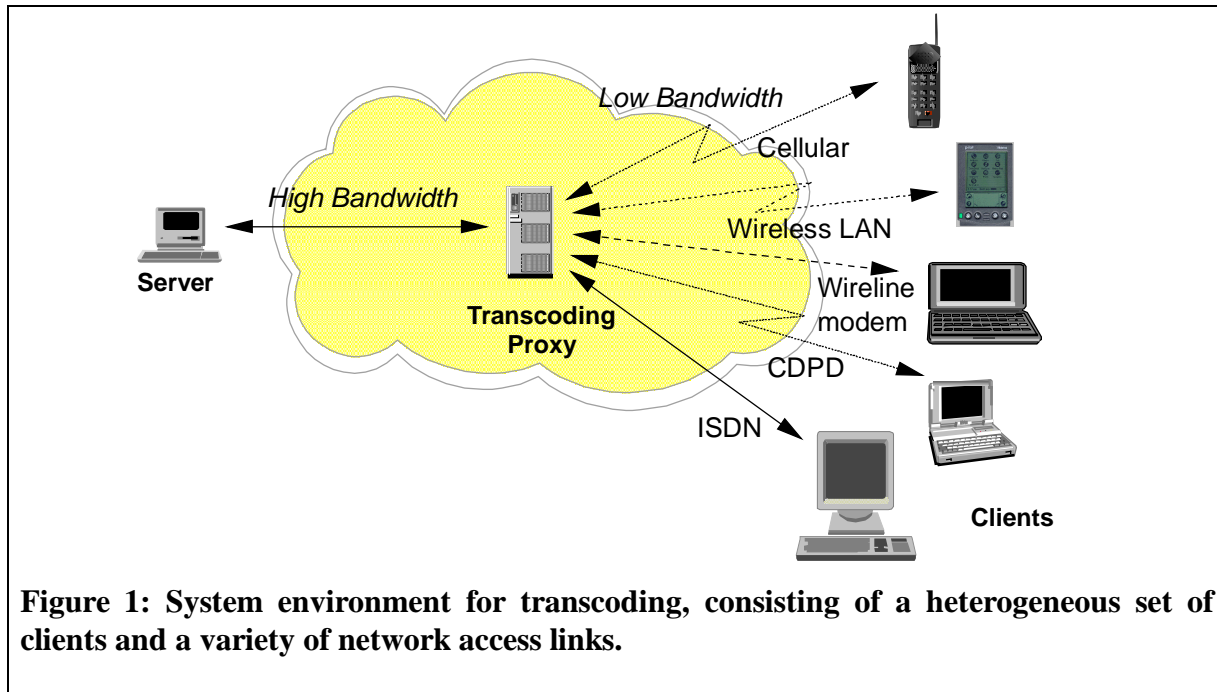
1. Introduction

Transcoding is the process by which a data object in one representation is converted into another representation. Typical examples include conversion within media types (e.g. an image encoded in one standard is transcoded into an image encoded in a second standard), as well as conversion between media types (e.g. speech to text). In addition to format conversion, transcoding also allows a data object to be compressed. Transcoding controls both the output number of transcoded bits as well as the semantic meaning of those transcoded bits.

Recently, transcoding of images and/or text has been integrated into an HTTP proxy [1][2][3]. As shown in Figure 1, transcoding proxies act as intermediaries between World-Wide-Web (WWW) servers and a variety of Web-enabled client devices that are connected over communication links with widely varying characteristics. Without requiring modifications to Web servers and browsers, an HTTP transcoding proxy enables the following:

- dramatic reduction in Web download times over low-bandwidth links via data compression
- reduction of per-byte costs over tariffed links via data compression
- tailoring of Web data to a variety of client devices via format conversion

Mobile devices are frequently connected via low-bandwidth wireless or medium bandwidth wireline modem links that make the viewing of rich Web content, such as images, very cumbersome due to the very long download times that result. In addition, over tariffed wide-area wireless networks the cost of such downloads can be prohibitive. A transcoding Web proxy can greatly reduce the size (in bytes) of Web data while maintaining most of its semantic value.

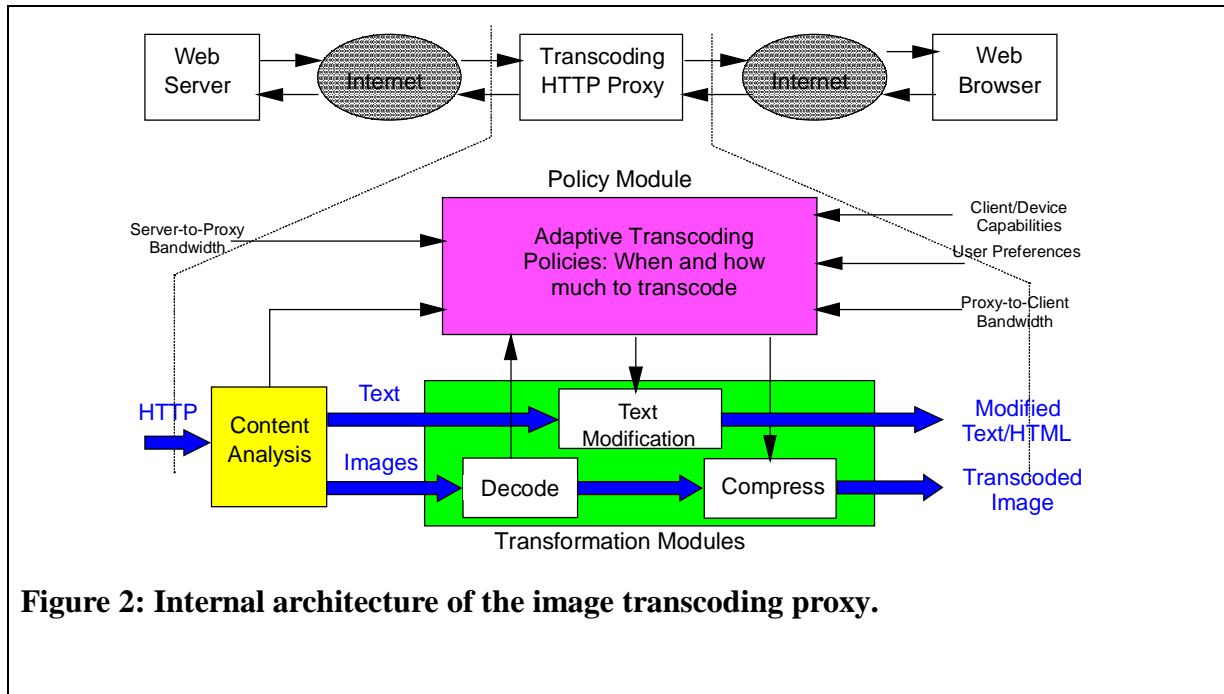


Download time reductions of six to ten times can be achieved using lossy compression techniques for typical Web images without losing their intelligibility.

As a second important benefit, such an intermediate proxy is also capable of tailoring text and images for the multitude of small, weakly connected, but Web-enabled, mobile devices that are now available. The capabilities of these mobile devices to receive, process, store and display Web content varies widely. An active Web proxy can transcode/change the Web content to best fit the resolution, color-depth, and dimension constraints of a small-screened device and to reduce the byte size of the stored data to a fraction of its original byte size.

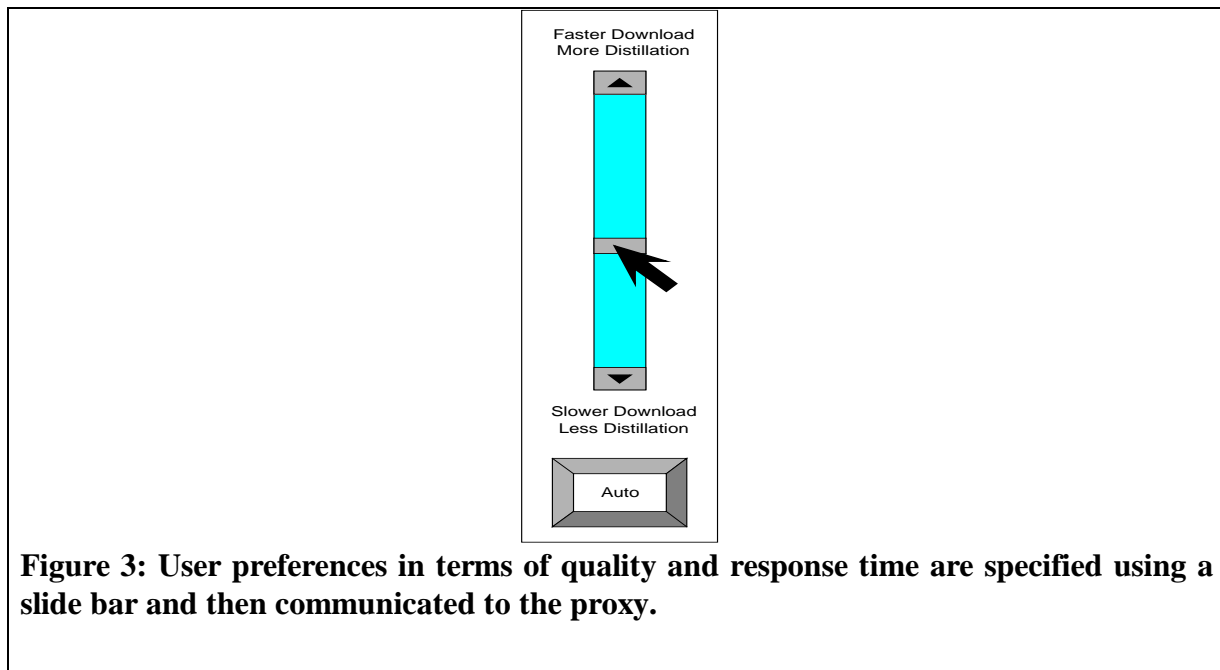
The basic architecture of the transcoding proxy is described and shown in Figure 2. The transcoding Web proxy is built by integrating a transcoding subsystem into an HTTP proxy. The transcoding subsystem can be separated into two primary components: the policy module and the transformation modules. The transformation modules modify the downstream data (i.e., HTML pages and GIF [4] and JPEG images [5]) that are being returned as responses to the client Web browser. The decision concerning what transcoding policy (i.e., the transcoding algorithm along with its parameters) to use is made by the policy module based on a number of criteria, including: 1) the characteristics of the data (e.g., byte size of the images, current encoding efficiency, structural role in the HTML page) as determined by the content analysis portion of the figure, 2) the current estimate of the bandwidths on the client-to-proxy and proxy-to-server links, 3) the characteristics of the client, particularly the client display capabilities, and 4) the user preferences concerning the preferred rendering of the data.

An example of the user preferences interface that we, and others [6], have implemented is shown in Figure 3. The user preference slide bar is a means for the user to interact with the transcoding proxy to dynamically change the tradeoff between image quality and download time. The user



sets a level or index value on the slider bar, which is then fed back to the transcoding proxy's policy module. Ultimately, this index value maps onto a set of transcoding parameters that are passed to the data transformation modules. We discuss two types of mapping in Section 3: a static fixed-quality mapping; or a dynamically varying mapping based on adjusting the parameter vector to keep the response time fixed.

The policy module generates a set of transcoding parameters, or transcoding vector, that controls the extent and types of compression performed by the transformation modules. The scaling



parameter determines how much an image is downsampled. Quantization parameters control how an image is quantized in the pixel domain and/or the frequency domain. The number of colors in a colormapped image can be reduced, or a 24-bit color image may be converted to 8-bit grayscale, or even a monochrome representation. Given N parametrizable compression options, then the N -

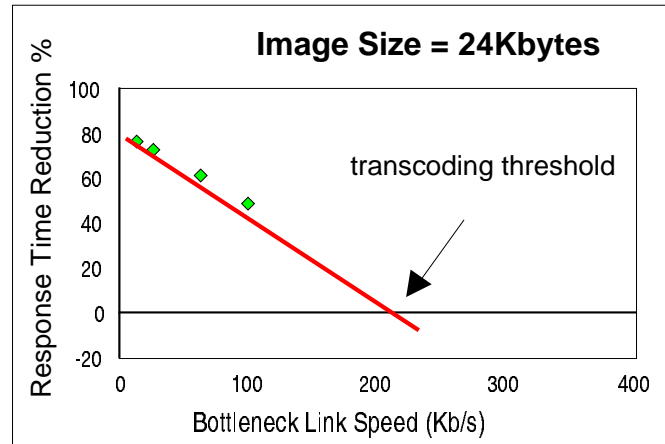


Figure 4: Response time is reduced, and transcoding should be performed, only when the bottleneck bandwidth is below an image-dependent threshold.

tuple space of possible combinations becomes quite large, which poses a problem for optimized transcoding, as we shall see in Section 3.

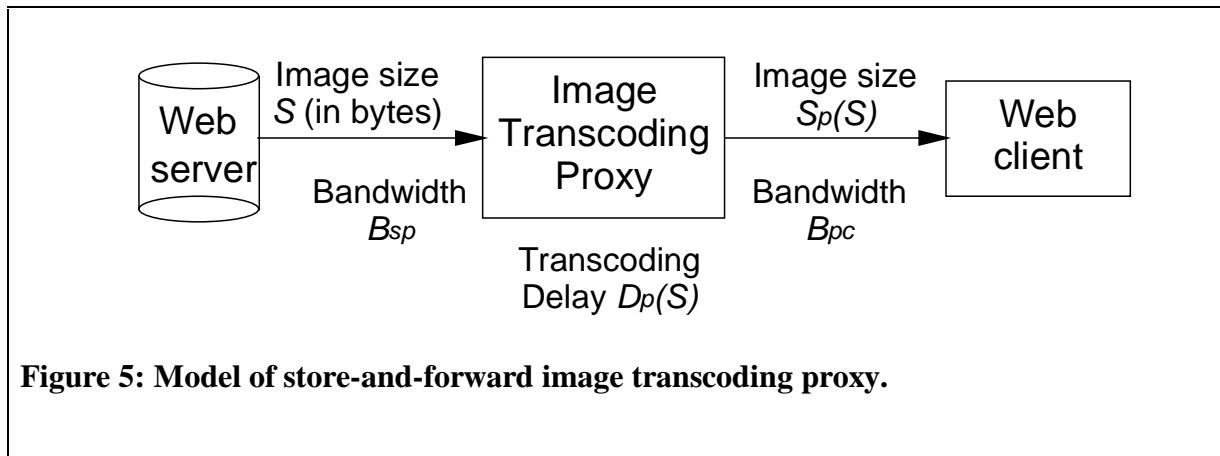
To maximize the benefits of transcoding, the policy module needs to consider each of the above criteria to determine under what conditions transcoding is able to reduce the response time. Transcoding of an image introduces delay, which must be offset by the reduction in transmission time due to compression. For very low bandwidth proxy-client access links, the reduction in response time due to aggressive image compression typically far outweighs the addition to response time caused by compute-intensive transcoding. However, Figure 4 shows that as the bandwidth of the proxy-client link increases, there comes a point at which it is no longer beneficial to transcode since the reduction in response time due to aggressive compression decreases as a function of the bottleneck link's bandwidth, while the transcoding time remains constant. We will formulate conditions in the following sections which more completely identify when it is beneficial for a proxy to transcode an image.

In the next four sections, we describe our work on adaptive transcoding, beginning with a mathematical analysis and ending with a description of our implemented set of transcoding policies:

- Store-and-forward image transcoding
- Automated store-and-forward transcoding
- Streamed image transcoding
- Actual transcoding policies implemented within our image transcoding proxy

2. Adaptive Store-and-Forward Image Transcoding

In this section, we develop an analytical framework determining when it is beneficial to transcode for a class of proxies that store and forward images. We define a store-and-forward image transcoder as an image transcoder which must wait to accumulate an entire input image before transcoding can begin on this image and then must wait to generate a transcoded image in its entirety before it is made available to be output, i.e. the input image cannot be read partially nor can the output image be written partially by the transcoder. Typical command-line interfaces to image conversion libraries are store-and-forward transcoders that require the input image in its entirety before image processing can commence, and internally generate a transcoded image in its entirety before making it available as output.



Consider the analytical model of the system shown in Figure 5. The original image of size S (bytes) is downloaded into the store-and-forward proxy over the server-proxy connection with effective bandwidth B_{sp} (bits/sec). The transcoder introduces a delay $D_p(S)$ and generates an output image of size $S_p(S)$. Both the transcoding delay and output image's byte size are denoted to be dependent upon the input image's byte size S , though in fact they are dependent upon many other factors, including the image's content, its dimensions, the set of transcoding parameters applied, the compression algorithms used, the efficiency of the algorithm's implementation, and the desired user preferences (quality and response time). The transcoded image is then transmitted over a proxy-client connection having effective bandwidth B_{pc} .

2.1 To transcode or not to transcode

Suppose R_o is the response time of fetching a Web object of byte size S from the Web server with transcoding turned off. Similarly, let R_p denote the response time of fetching the transcoded version of the same Web object through the transcoding proxy. For the purpose of the following discussion we assume that caching is not supported at the proxy.

The client perceived response time with transcoding turned off is the sum of the following three terms:

$$R_o = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + \frac{S}{\min(B_{pc}, B_{sp})} \quad (1)$$

RTT_{pc} is the network roundtrip time latency between the client and the proxy and, similarly, RTT_{sp} is the latency between the proxy and the server. Fetching the Web object requires a TCP SYN/ACK exchange as well as an HTTP request/response, thereby contributing $2 \cdot RTT_{pc} + 2 \cdot RTT_{sp}$ to the delay term. In addition, 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.

When transcoding is turned on, the proxy operates in a store and forward mode. $2 \cdot RTT_{pc} + 2 \cdot RTT_{sp}$ is again the fixed component of the response time. $D_p(S)$ is the additional term that represents the transcoding delay. The object download time is the sum of the transfer time over the server-proxy link and the proxy-client link. The exact relationship can be expressed as:

$$R_p = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} \quad (2)$$

Transcoding will reduce response time if $R_p < R_o$. That is,

$$D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} < \frac{S}{\min(B_{pc}, B_{sp})} \quad (3)$$

The above inequality precisely characterizes the regime for which transcoding reduces response time. We can also use the same inequality to define an objective policy for making transcoding decisions. The proxy can then use this policy to decide on an image-by-image basis whether transcoding should be applied or not. Except for S , the byte 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 estimated.

Clearly, when $B_{pc} > B_{sp}$, it is always the case that $R_p > R_o$. Thus, when the Internet backbone's server-proxy connection is the bottleneck, then a store-and-forward image transcoder should never transcode.

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

$$D_p(S) + \frac{S}{B_{sp}} < \frac{S - S_p(S)}{B_{pc}} \quad (4)$$

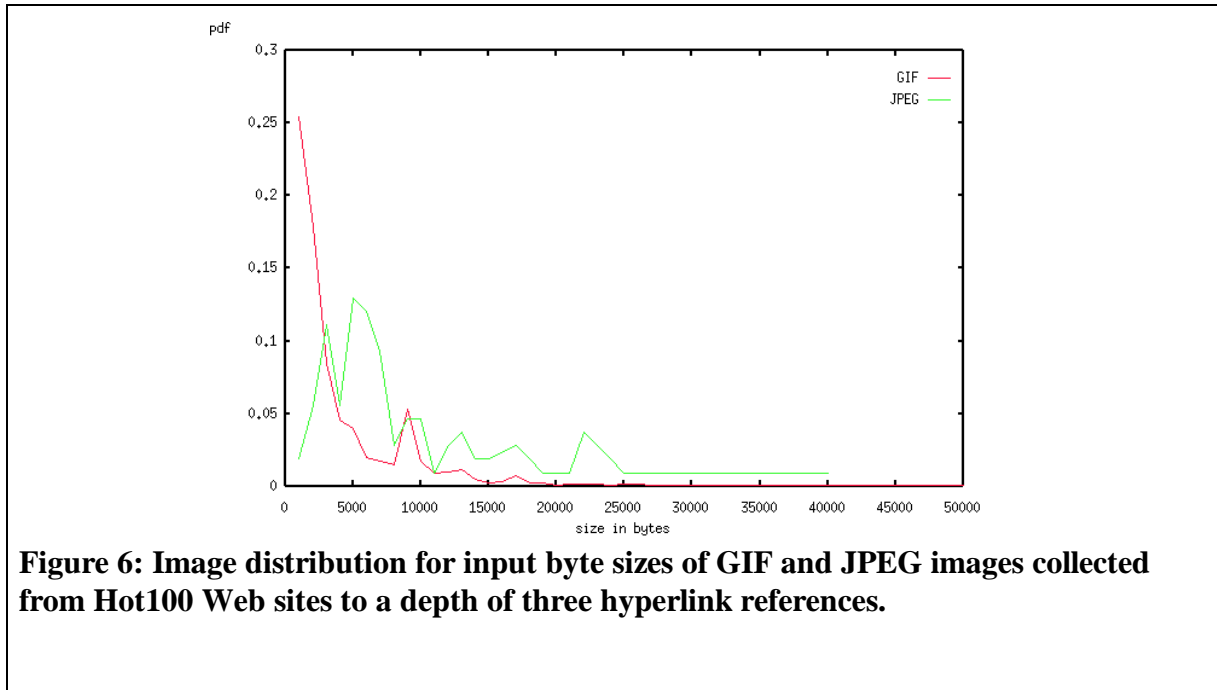
2.2 Prediction of the transcoded image's output size in bytes

In order to determine whether response time can be decreased by transcoding, we need to be able to predict the transcoded image's byte size $S_p(S)$ in Inequality 3. The output byte size depends on a variety of factors, including the content of the image (i.e. whether it is a natural image, or an artificially rendered text/graphic image), the image dimensions, the input byte size S , the output compression format, the transcoding parameters (e.g. depth of quantization and/or scaling), and the implementation efficiency of the compression algorithm.

Prediction must be performed in advance of transcoding. This requires that we infer as much information as possible from such sources as the image header and prior image statistics. The image header typically provides the input byte size S and image dimensions. In addition, we assume that the transcoding parameters and output compression format are chosen in advance and given to the output byte size prediction module. Images that share the same input byte size S and dimensions, and that are transcoded with the same parameters to the same output format, will nevertheless exhibit a variation in transcoded output byte size due to inherent variation of image content across images. Consequently, output byte size prediction requires an analysis of the statistical distribution of output sizes for previously transcoded images. Extracting correlation patterns between output byte size and the various input parameters is the objective of the rest of this section.

In the following, we confine our analysis of output byte size prediction to transcoding of color GIF and color JPEG images to the grayscale JPEG output format. We do not analyze image transcoding to the GIF output format in this paper. Also, we consider only the influence of the JPEG quality factor q on the output byte size. The JPEG quality factor q ranges from 0 to 100, as defined in the Independent JPEG group's code [7], where high quality values correspond to low compression, while low quality values correspond to aggressive compression. Though scaling is another useful transcoding parameter for compression, we do not consider its influence on output byte size here.

We collected images from the Hot100.com list of most visited Web sites, to a recursive search depth of three levels. In Figure 6, we illustrate the probability distribution of the collected input images as a function of their input byte size. The collected set of images consists of 2616 GIF images, and 108 JPEG images. The bias towards GIF images is presumably because most Web sites (e.g. CNN.com) are filled with artificially-rendered advertisements, logos, and navigation bars that compress well under GIF, rather than natural images that compress well using JPEG. The average byte size of a GIF was 3.6 KB, and the average byte size of a JPEG was 10.9 KB.



The collected images were transcoded using an internal IBM image conversion library, whose JPEG implementation is essentially the same as the Independent JPEG Group's implementation. For each transcoded image, we recorded the input format, input byte size S , height, width, input bits per pixel bpp , quality factor q that was applied, output image's byte size, and transcoding time. Figure 7 shows scatter plots generated using various sets of transcoding parameters. We restricted the viewing area of each scatter plot in order to reveal the differences between different methods of prediction. Consequently, not all points are shown. A small fraction of outlying GIF images were larger than the 30 KB upper limit of the GIF scatter plots, and a small fraction of outlying JPEG images were larger than the 100 KB upper limit of the JPEG scatter plots.

First, consider Figure 7(a), which is a scatter plot that attempts to predict output byte size as a function of the input image's byte size S . Each input GIF in our sample was transcoded to an output JPEG using a JPEG quality factor of $q=5$. To quantify the correlation between output and input values, we applied the *matlab* package's *corrcoef()* function to our set of data points. *Corrcoef* uses the standard definition of a normalized correlation coefficient [8] to generate a number ρ between -1 and 1 . For Figure 7(a), we calculated $\rho=0.54$.

Our scatter plots in Figure 7 reveal that output byte size can be approximated by a linear function of the area of the input image, i.e. the number of pixels in the image. The output byte size in Figure 7(b) exhibited a much higher correlation ($\rho=0.90$) on the area of the image than it did to the input byte size ($\rho=0.54$) in Figure 7(a) given the same set of input parameters and conditions. All of the Figures 7(b)-(e) that are functions of the area of the image generated very high correlation values ($\rho>0.9$). High correlation values near 1 indicate a strong linear relationship between input and output. Consequently, Figures 7(b)-(e) each show a stronger linear relationship between the output byte size and the image area. We have used *gnuplot* to generate a

lines that have been curve-fitted to these sets of data points to emphasize the approximately linear relationship between output and input.

The strong linear correlation between the number of output bytes and the area in Figure 7(b) explains the relative lack of correlation between the output byte size and the input byte size in Figure 7(a). A large-dimension image and a small-dimension image may share the same input byte size S , yet produce vastly different output byte sizes based on the linear prediction obtained from Figure 7(b). This results in a wide spread around each value S and lower correlation in Figure 7(a). A topic of ongoing research is trying to explain the general linear dependency of the number of output bytes on the image area, which seems to be independent of input format and quality factor.

We have several final remarks on our data. The JPEG scatter plots had many fewer images than the GIF input plots due to the skew of the Hot100 sites towards GIF encoding. Also, we note that several vertical lines appear in Figure 7(b). These are due to a concentration of images in our collected sample which all share the same input area. It is possible that a standard byte size or set of dimensions exist on the Web that would cause this phenomenon, though we haven't yet verified this. Finally, it appears at first glance in Figure 7(b) that zero input pixels fail to produce zero output bytes, as one would normally expect. In fact, a small input GIF (whose header takes about 120 bytes for a 32-color image) may occupy less than 200 bytes total and appear to be a zero input given that the scale of the x-axis is in thousands of bytes. In addition, a typical JPEG grayscale header incurs about 300 bytes in overhead due to quantization and Huffman table definitions. Given the different scales of the x and y axes, a small JPEG generated from a small input GIF may appear to cause a sudden jump to hundreds of output bytes starting from a smaller number of input bytes that look deceptively close to zero.

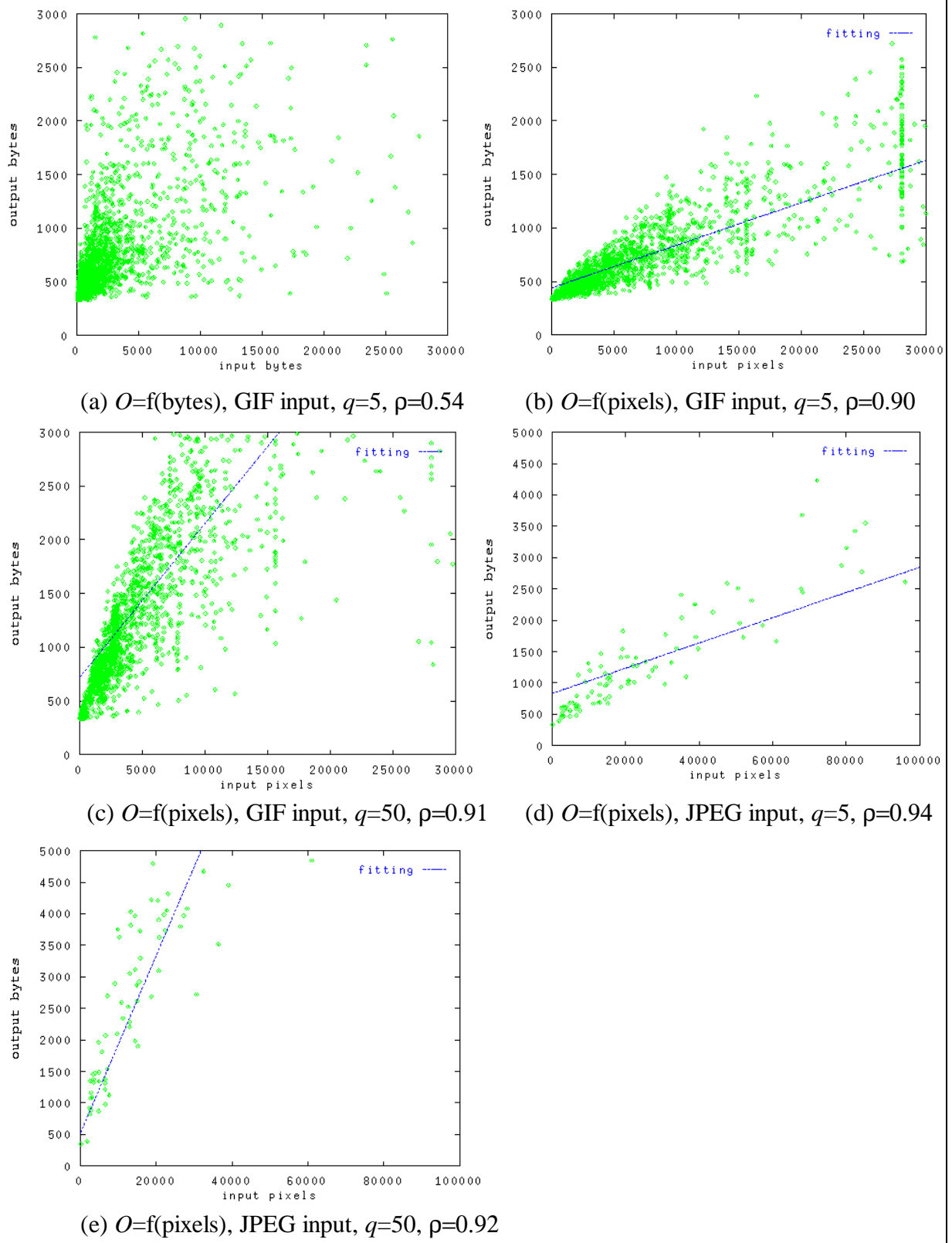


Figure 7: Prediction of the output size O (in bytes) of a transcoded image.

2.3 Prediction of the transcoding delay

In order to evaluate Inequality 3, we also need to be able to accurately predict the transcoding delay $D_p(S)$ of each image. The transcoding time is more difficult to predict than the output byte size because $D_p(S)$ depends not only on image processing time, but also on queuing delay caused by the operating system's sharing of the CPU among multiple processes and threads. The variable delay introduced by the operating system is especially difficult to predict in non-real-time operating systems.

Suppose we desire only to predict the time due to image processing. $D_p(S)$ will depend upon the speed of the processor, and the implementation efficiency of both decoders and encoders in the image conversion library, in addition to the factors from Section 2.2 that influenced the output byte size, namely the content of the image, the image dimensions, the input byte size S , the output format, and the transcoding parameters. Similar to the case of output byte size prediction, transcoding time prediction will require an analysis of the statistical distribution of previously transcoded images, so that correlation patterns between output delay and input parameters can be extracted.

We assume the same conditions for transcoding as the previous section. In addition, the transcoding time measurements of this section were performed on an internal IBM image conversion library on a 200 MHz Pentium Pro running NT 4.0.

Our scatter plots in Figure 8 reveal that transcoding time can also be approximated by a linear function of the area of the input image. We hypothesize that Figure 8(d)'s linear dependency is due to the fact that the largest component of time in JPEG encoding and decoding is the DCT. The number of DCT's invoked per image is a direct linear function of the number of blocks, hence the area, of the image. Thus, for JPEG-to-JPEG conversions, there should be a strong linear relationship between transcoding time and area. Figure 8(d) ($\rho=0.98$) and Figure 8(e) ($\rho=0.98$) exhibit this strong linear relationship for different values of q . For GIF-to-JPEG conversions, we would expect GIF decoding times to be highly image-dependent, so that the overall transcoding time would exhibit less linear correlation than the JPEG-to-JPEG cases. In fact, the correlation values of Figure 8(b) ($\rho=0.82$) and Figure 8(c) ($\rho=0.60$) are both lower than their JPEG-to-JPEG counterparts at the same value q . This hypothesis would also explain the relative lack of linear correlation between the transcoding time and the input byte size in Figure 8(a). The linear correlation between area and output transcoding delay in Figure 8(b) would predict that large and small images sharing similar input byte sizes will have a wide spread in transcoding times.

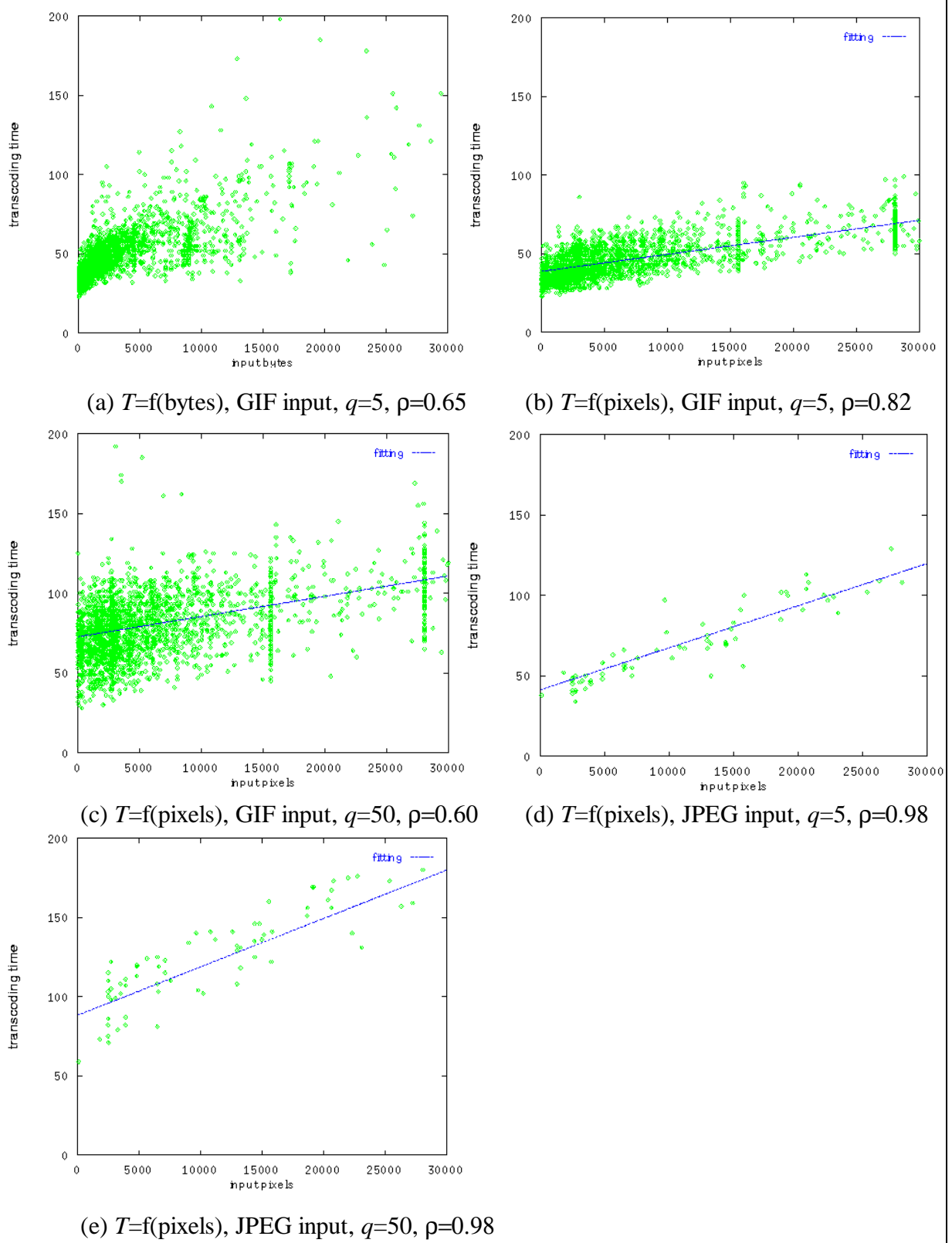


Figure 8: Prediction of the image transcoding time T (in ms) for a transcoded image.

2.4 Connection monitoring and download time estimation

The automated policy decisions depend to a large extent on the accuracy of image download time estimates. To evaluate Inequality 3, we require accurate estimation of the download time of image objects, as represented by the terms $S_p(S)/B_{pc}$ and S/B_{sp} . Our method of performance prediction involves monitoring the ongoing connections and estimating image download times by performing statistical analysis on the collected traces. These functions are split into two modules: viz., *Connection Monitor* and *Statistical Analyzer*. The connection monitor is a transparent shim layer (a winsock style dll on windows platform) which is inserted between the application and the socket layer. It records every send and receive event at the proxy. Whenever an application (e.g., Web browser, or proxy server) makes a socket layer call, a stub routine inside the shim layer is executed, which, after creating a log entry, passes control to the requested function inside the socket layer. The overhead of monitoring is minimized by keeping the code path through the shim layer short and moving all computation to the statistical analyzer module. For each monitored connection we record the following information:

- time when a new connection is established
- source address, source port number & destination address, destination port number of each network connection
- number of bytes sent and received on each connection and the respective timestamp of every send and receive event
- time when connections are closed

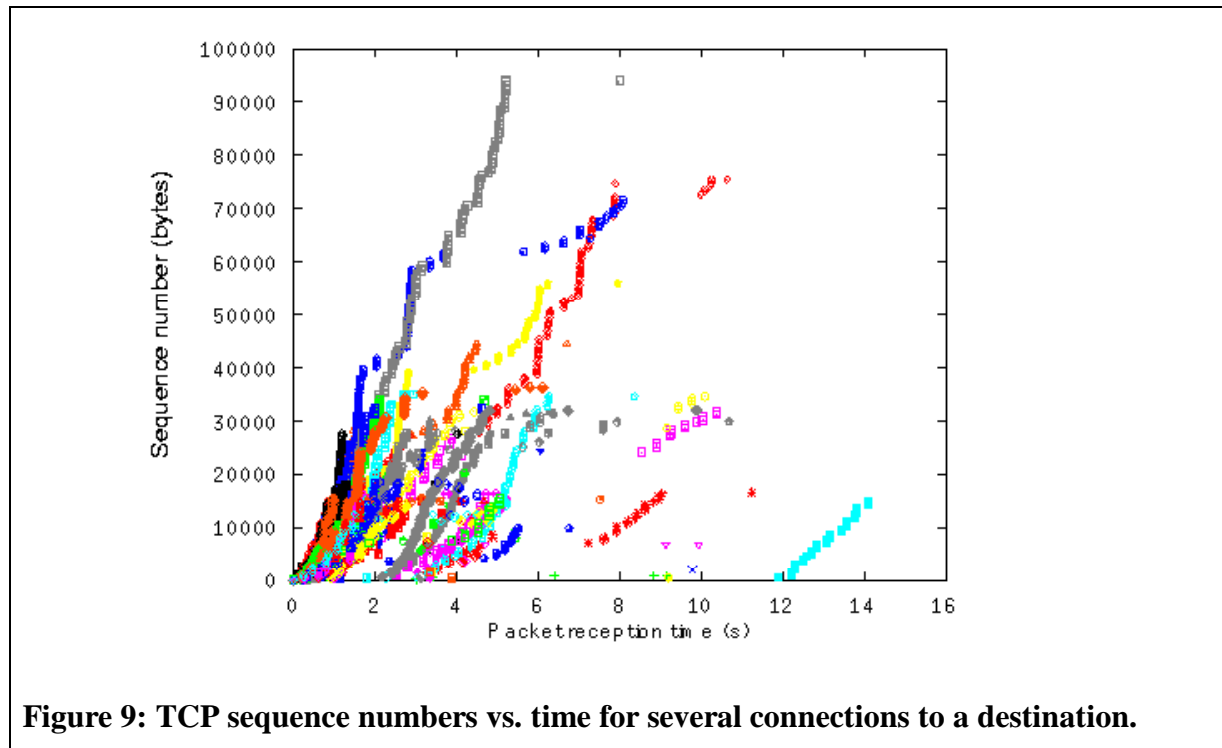
The statistical analyzer maintains a database of all past and all current active connections. Based on the history of collected samples the statistical analyzer is able to make predictions about the download time of image objects. This approach is similar to the passive monitoring method, called SPAND that was originally proposed in [9]. In fact, SPAND's monitor and statistical analyzer can alternatively be used in conjunction with our proxy. The modular architecture of the proxy allows us to flexibly place the monitor and analyzer function anywhere in the network, though co-locating the two functions at the proxy minimizes the overhead of communication between them.

For predicting the server-to-proxy download time and the proxy-to-client download time we use different heuristics. From server to proxy traces we estimate the download time, rather than the bandwidth directly. This is because throughput of small sized wide-area TCP connections is typically a non-linear function of object sizes. The non-linear behavior is clearly visible in Figure 9, which is a typical example of a sequence number vs. time plot for a set of http connections to a fixed Web server¹. To predict the download time of an object of a given size, we look at the history of connections to the chosen destination and compute the distribution function of the download time for all objects that have roughly the same size. The median, or some other appropriate statistical function of this distribution, is returned as the download time estimate.

In contrast, proxy-to-client TCP behavior is dominated by the effects of having a bandwidth constrained link, which is typically the last hop. Because there is a bottleneck link, the aggregate of all active TCP connections to a client typically saturate the bottleneck link. Thus, for providing

¹ Based on wide area TCP traces provided to us by Srinivasan Seshan.

proxy-to-client download time estimates, we aggregate all active connections to a client into a single group. For each group of connections we plot a time vs. number of bytes plot and then perform a linear curve-fit on the data. This gives us a reasonably accurate estimate of the current available bandwidth between the proxy and the client. Although this value changes with time, the oscillations in most cases are bounded. For example, we can easily detect whether we are connected by a 14.4 Kb modem, 28.8 Kb modem, or 56 Kb modem by looking at the output of our curve fitting algorithm. The predicted values are not exact (20 - 30% deviation from the correct values), but we expect the policy decisions based on these estimates to outperform any adaptation methods solely dependent on user selected preferences.



3. Automated Store-And-Forward Image Transcoding

Two classes of transcoding policies that incorporate the test condition of Inequality 3, namely fixed-quality and fixed-delay (automated) adaptation, are discussed in this section. The distinction between the two classes is based on whether the user preference index value fed back by the slide bar in Figure 3 is interpreted as a quality specification or as an upper bound on tolerable delay.

The simplest case results from interpreting the slide bar's index value as a fixed-quality specification. In this scenario, the user is manually choosing the set of transcoding parameters with which to transcode an image. A fixed-quality store-and-forward transcoder would evaluate Inequality 3 exactly once using the prespecified vector of parameters, which affect the prediction

of the transcoding delay $D_p(S)$ and output byte size $S_p(S)$. The following pseudocode summarizes the fixed-quality store-and-forward transcoder's policy:

```

If (  $D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} < \frac{S}{\min(B_{pc}, B_{sp})}$  ) then /* evaluate with the user's parameter vector */
    transcode according to the user's parameter vector
else
    send original untranscoded image

```

It should be noted that the above algorithm is not purely “fixed” quality, since the original untranscoded image may be returned. It is only constant quality up to the point that the delay limit imposed by the store-and-forward inequality is satisfied. If the chosen quality vector introduces too much delay, so that transcoding is no longer beneficial, then the original image is returned. While it is possible to return an image other than the original, we have not evaluated the alternatives.

The second more complex case results from interpreting the slide bar's index value as a fixed-delay specification. In this scenario, the amount of compression is automatically being adapted to dynamically varying network bandwidths subject to a maximum tolerable delay or response time, hence the term *automated* adaptation. In contrast, the fixed-quality approach also fixed the amount of compression.

Fixed-delay automated adaptation is useful when the end user does not know in advance the available network bandwidth, byte size of the image, and/or proxy utilization. Automating the task of compression and transcoding parameter selection not only simplifies the end user interface but also improves performance. For each Web access, the system can automatically select the optimal transcoding parameters and provide consistent and predictable response to the end user.

Conceptually, we think of the user as having pushed the “Auto” button in Figure 3 after setting the slide bar's level. At the transcoding proxy, the slide bar's index value is mapped on to a fixed response time which is the upper bound on delay that the user is typically willing to tolerate. We define D_{max} to be the upper bound on the downstream delay of image delivery. D_{max} can be easily derived from the overall response time by subtracting the roundtrip terms of R_o in Equation 1.

Adapting to the user-specified maximum downstream delay D_{max} requires some revision of the right hand side (RHS) of Inequality 3. First, consider the case when D_{max} is less than the original RHS term representing the no-transcoding bound, $\frac{S}{\min(B_{pc}, B_{sp})}$. Under these conditions, we obey the user's request for a tighter delay bound and substitute D_{max} in place of the RHS no-transcoding bound when evaluating Inequality 3. However, when D_{max} is greater than the no-transcoding bound, we stay with the tighter no-transcoding bound, because turning off transcoding for this image would still result in an image that satisfies the requested delay bound D_{max} . As a bonus for not transcoding, the tighter no-transcoding bound results in extra delay slack, equal to $D_{max} - (\text{RHS term})$, which may be exploited for increased traffic capacity through rearrangement of the scheduled order of transmission for various images. In summary, the

transcoding proxy replaces the RHS term $\frac{S}{\min(B_{pc}, B_{sp})}$ of Inequality 3 with the expression $\min(D_{\max}, \frac{S}{\min(B_{pc}, B_{sp})})$, i.e.

$$D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} < \min(D_{\max}, \frac{S}{\min(B_{pc}, B_{sp})}) \quad (5)$$

Evaluating this new test condition requires the proxy to find a feasible set of transcoding parameters which satisfy the new inequality. This feasibility test is what makes automated transcoding considerably more complex than fixed-quality transcoding, which predetermined the parameter vector. Given N compression parameters, it is not immediately clear to the authors how to swiftly find a single feasible vector that satisfies Inequality 5 within the N -tuple space formed by the transcoding parameter combinations. Assuming that such a feasible vector could be found, then the next step is to find the optimal vector in a feasible set of vectors, namely the set of transcoding parameters which maximizes the image quality subject to the upper bound on delay. Again, it is not immediately clear whether the search for optimality can be conducted swiftly, or whether a fast approximation to optimality can be found. If no feasible vector can be found, then the proxy cannot meet this maximum response time with any combination of transcoding parameters, and so our default policy is to send the original full-fidelity/untranscoded image.

The following pseudocode summarizes the policy executed by an automated/fixed-delay store-and-forward transcoder:

```

If (there exists a set of transcoding parameters such that
     $D_p(S) + \frac{S}{B_{sp}} + \frac{S_p(S)}{B_{pc}} < \min(D_{\max}, \frac{S}{\min(B_{pc}, B_{sp})})$ ) then /*Test For Feasibility */
    search space of transcoding parameters to find optimal set that maximizes
    "quality" subject to this response time /* Search For Optimality */
    transcode using optimal transcoding parameters
else
    send untranscoded image
    
```

It should be noted that the above algorithm is not purely "fixed" delay, since the original untranscoded image may be returned. The user's response time is satisfied only up to the point that the automated store-and-forward inequality is satisfied. Network conditions may dictate that no transcoding vector can be found that meets the upper bound on delay, in which case the original image is sent to the user, thereby creating the impression of "variable" delay. While it is possible to send an image other than the original, we have not investigated the alternatives.

4. Adaptive Streamed Image Transcoding

In this section, we derive conditions under which it is beneficial for a streamed image transcoder to engage in transcoding. A *streamed* image transcoder is an image transcoder which starts writing out image data encoded in an output format before having fully read in the complete input stream of bytes corresponding to the whole image encoded in the input format.

We begin our analysis of streamed transcoding with a timing diagram shown in Figure 10. The input image arrives as a stream of bits spaced apart by $1/B_{sp}$. The streaming image transcoder will take a group of G bits for transcoding, incurring a small store-and-forward delay D_1 . The group of bits is then transcoded into a group of G_p output bits, incurring a delay D_2 . If $D_2 < D_1$, then the image transcoder can convert each input group G to its corresponding output group G_p before the next input group G needs to be processed. In this case, the streaming image transcoder's internal memory requirement is bounded. However, if $D_2 > D_1$, then the image transcoder will not be able to process input bits fast enough. In this latter case, given a continuous input stream, then the image transcoder's internal memory requirement grows without bound, i.e. the image transcoder's finite-length internal RAM buffers will overflow. Therefore, we desire that the transcoding delay D_2 satisfy $D_2 < D_1$. Clearly, $D_1 = \frac{G}{B_{sp}}$. To find D_2 , let $D_p(S)$ = the predicted image transcoding time for an image of S bits ($D_p(S)$ actually depends on other parameters, such as image content and dimension, but we use $D_p(S)$ for simplicity of notation). Then, $D_2 = \frac{D_p(S)}{S/G}$. To avoid overflowing the RAM buffer, the group transcoding delay must satisfy $\frac{D_p(S)}{S/G} < \frac{G}{B_{sp}}$, or

$$D_p(S) < \frac{S}{B_{sp}} \quad (6)$$

Assuming that Inequality 6 holds true, then the output transcoded groups G_p will be uniformly spaced by a delay equal to D_1 . The transmission channel can send each transcoded group of bits G_p in time $D_3 = \frac{G_p}{B_{pc}}$. In case i), we illustrate $D_3(i) < D_1$, i.e. each output group G_p can be sent before the next output group is ready for transmission. In case ii), $D_3(ii) > D_1$, so that the output transmission link cannot send the produced bits fast enough to keep the output queue empty. In case ii), the transmission link's output queue grows without bound given a continuous stream of transcoded bits, causing overflow for finite-length link buffers. Therefore, we desire that the delay caused by the transcoded output group size satisfy $D_3(i) < D_1$. Clearly, $D_3(i) = \frac{G_p}{B_{pc}}$. To avoid overflowing the transmission link's output buffer, the transcoded output image group size G_p must satisfy $\frac{G_p}{B_{pc}} < \frac{G}{B_{sp}}$, or

$$\gamma > \frac{B_{sp}}{B_{pc}} \quad (7)$$

where $\gamma = \text{group image compression ratio } G/G_p$, which we assume to be on average equivalent to the overall image compression ratio S/S_p . In summary, the streamed image transcoder should only perform transcoding when both Inequality 6 and Inequality 7 are satisfied.

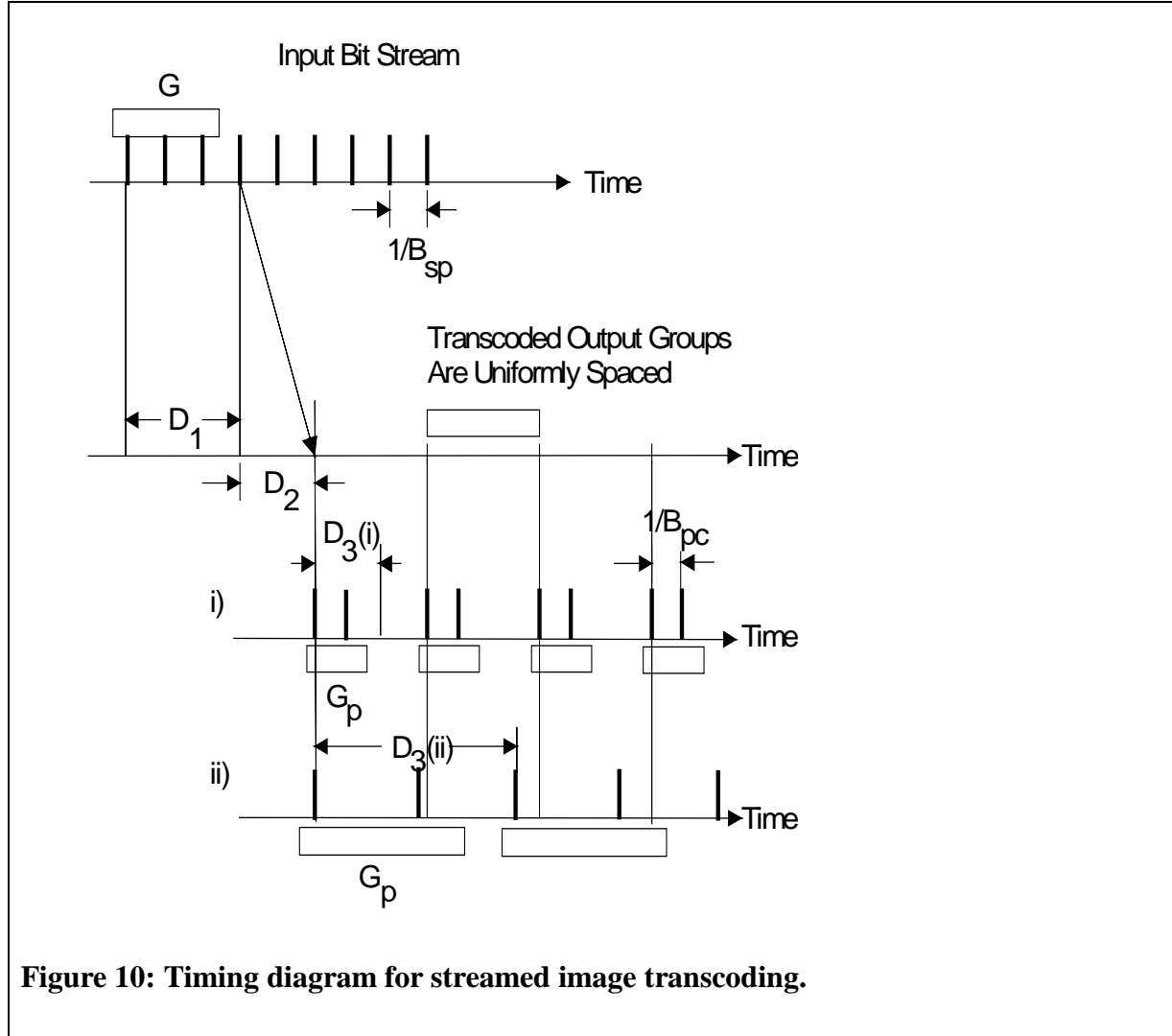


Figure 10: Timing diagram for streamed image transcoding.

If the server-proxy link is the bottleneck, i.e. $B_{sp} < B_{pc}$, then Inequality 7 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. In this case, only Inequality 6 must be satisfied in order for transcoding to not be disadvantageous. In fact, when the server-proxy link is the bottleneck, Inequality 7 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. GIF->2-bit grayscale (Palm PDA format). The above inequality allows us to determine when such a format conversion will increase the chances of buffer overflow, and when format conversion will not cause buffer overflow. For example, if we have $B_{sp} = 1$ bps, $B_{pc} = 2$ bps, and $G = 1$ bit, then Inequality 7 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 7 says that the image compression ratio γ must be greater than the ratio of server-proxy to proxy-client bandwidths in order for transcoding to be worthwhile. In addition, Inequality 6 must also be satisfied.

Note that Inequality 6 and Inequality 7 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.

5. Practical Policies for an Image Transcoding Proxy

In previous sections, we developed an analytical framework for determining the conditions under which it is beneficial to transcode for store-and-forward as well as streaming proxies. In this section, we discuss the set of practical transcoding policies that we have implemented in our store-and-forward transcoding proxy. These policies were developed incrementally by feeding back the lessons we learned throughout the development and evaluation of our transcoding proxy. The implemented policies adapt the transcoding to the client's device type, user quality preferences, and image content, but do not currently adapt to network bandwidth, and do not currently perform prediction of the image transcoding delay and output byte size.

Our current set of transcoding policies is summarized in pseudocode in Figure 11. Our transcoding proxy first makes a distinction in line 1 between transcoding to a laptop and transcoding to a Palm Pilot PDA. Unless otherwise notified, the proxy assumes that it is transcoding to a PC laptop, namely a client device that supports GIF and JPEG decoding.

Given a PC/laptop, the proxy next checks in line 2 to see if the image is sufficiently large for transcoding. Images smaller than a threshold of 1000 bytes are deemed to be not worth the savings in download time brought about by compression and hence are not transcoded. This threshold is obtained by applying Inequality 3's test condition, except that we assume that B_{pc} and B_{sp} are fixed, and that the common case is where $B_{pc} < B_{sp}$. Rearranging terms in Inequality 3, we find that the input byte size must satisfy the following in order for transcoding to be worthwhile:

$$S > \frac{D_p(S) + \frac{S_p(S)}{B_{pc}}}{\left(\frac{1}{B_{pc}} - \frac{1}{B_{sp}}\right)} \quad (8)$$

Let $B_{pc} = 50$ kbit/s, $B_{sp} = 1$ Mbit/s. For small images of input length about 500 bytes, we have measured average transcoding times of approximately 40 ms through our transcoding routines. We assume that it is typically hard to squeeze significant compression out of small images, say no more than 2 to 1, so that input and output sizes are closely related, and that the transcoding time does not vary much within this small range. Consequently, producing $S_p(S) = 500$ output bytes starting from a small image will take about $D_p(S_p(S)) = 40$ ms of processing delay. Substituting values, Inequality 8 shows that the input byte size S must exceed about 800 bytes in order for transcoding to reduce response time, hence our threshold of 1000 bytes.

Next, in line 3, our transcoding policy makes a distinction based on the input image format, such that input GIF images are transcoded differently than input JPEG images. If the input image is in GIF format, then we make a further check in line 4 to see if the image is already well-compressed in GIF format. The GIF format is well suited to compressing graphics-type artificially rendered images that have very few colors, such as a map or a cartoon. GIF is poorly suited to compressing natural images with many shades of color, such as a photograph of a person. Conversely, JPEG is well suited to compressing natural images with many colors, and is poorly suited to compressing artificially rendered graphics that have embedded text and sharp edges. If the input GIF is already well compressed, then the image should be transcoded from GIF to GIF (line 5), because transcoding the image from GIF to JPEG will likely result in an expansion of the image byte size. Otherwise, if the input is not already well compressed in GIF format, then we transcode from GIF to JPEG (line 7), assuming that the image has enough natural variation to make it suitable for JPEG encoding.

Our test for determining whether the GIF is already well-compressed is based on calculating the bits per pixel value bpp of the input image. Extracting the image dimensions height h and width w from the image header, and given the image file length S , then $bpp = \frac{S}{h \cdot w}$. After experimenting with a variety of input GIF images, we found that $bpp < 0.1$ was a good indicator of well-compressed GIF maps and logos from GIF images. GIF images with more color invariably exceeded this threshold value.

If the input image format is in JPEG format (line 9), then the transcoder performs JPEG-to-JPEG transcoding. Our proxy never converts a JPEG to a GIF format, because a JPEG image typically has many colors, and will compress poorly in GIF format. In fact, GIF encoding of a decoded JPEG image will likely result in expansion rather than compression of the image.

<i>If (not to Palm) /* e.g. to laptop/PC */</i>	<i>line 1</i>
<i>If (input byte size > 1000 bytes) /* static evaluation of Ineq. 3 */</i>	2
<i>If (input is GIF)</i>	3
<i>if (well-compressed GIF)</i>	4
<i>GIF->GIF as f(user preferences)</i>	5
<i>else</i>	6
<i>GIF->JPEG as f(user preferences)</i>	7
<i>else /* input is JPEG */</i>	8
<i>JPEG->JPEG as f(user preferences)</i>	
9	
<i>If (output byte size > input byte size)</i>	10
<i>send original image</i>	11
<i>else</i>	12
<i>send transcoded image</i>	13
<i>else /* to Palm */</i>	14
<i>GIF/JPEG -> Palm 2-bit grayscale as f(user preferences)</i>	15

Figure 11: The transcoding policies implemented in the image transcoding proxy.

In lines 5, 7, and 9, once the output format has been determined, the image is transcoded according to the user preferences (e.g. slide bar value) received from the user. The same slider index value can be mapped onto different transcoding vectors depending on the output format. Transcoding to GIF format limits us to scaling and colormap reduction to achieve compression. Transcoding to JPEG format allows us to specify quantization of frequency coefficients, as well as scaling. Occasionally, the choice of transcoding vector may actually result in expansion of the image. In this case, we send the original image (line 11), rather than the transcoded image.

For the case of transcoding to the Palm PDA, we transcode GIF images and JPEG images to the output format, namely a 2-bit grayscale bitmap [10], applying a degree of compression that is again a function of the user's preference (line 15). The HTTP Palm client informs the HTTP transcoding proxy that this is a Palm device by modifying the Accept field in the HTTP get request to say "image/palm".

6. Conclusion

We presented an analytical framework describing when to transcode and when not transcode for streaming and store-and-forward proxies under certain idealized assumptions. Transcoding is generally performed only when response time is reduced. While this idealized analysis provided insight that we later incorporated into our actual adaptation policies, it had practical limitations, such as requiring accurate prediction of image transcoding times and image output sizes, as well as accurate network bandwidth estimation of proxy-client and server-proxy links. We also described automated transcoding, a process by which the transcoding proxy adapts its image coding to network variability while trying to meet an upper bound on the delay tolerated by the end user. Automated adaptation requires a feasibility test to find transcoding vectors that satisfy the test for response time reduction, as well as an optimality search. Finally, we described our experience developing a practical set of policies that have been implemented within a transcoding HTTP proxy. These policies adapt to client heterogeneity, image content, and user preferences. These policies are largely motivated by the behavior of the GIF and JPEG image compression algorithms. They also incorporate a static evaluation of the store-and-forward test condition for reduced response time, thereby avoiding most of the test's practical limitations.

Acknowledgements

We wish to thank John R. Smith of IBM T.J. Watson for providing us with the set of images that he collected from the Hot100 Web sites using his Web spider. We also wish to thank Srinivasa Rao of IBM T.J. Watson for generating the *matlab* correlation statistics and for his proofreading help.

7. Bibliography

- [1] A. Fox, E. Brewer, "Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation", *Fifth International World Wide Web Conference, Computer Networks and ISDN Systems*, vol. 28, no. 7-11, May 1996, pp. 1445-56.
- [2] A. Joshi, S. Weerawarana, and E. Houstis, "On Disconnected Browsing of Distributed Information", *Proc. IEEE RIDE '97* , 1997, pp 101-107.
- [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] M. Nelson, J. Gailly, *The Data Compression Book, Second Edition*, M&T Books, 1996.
- [5] W. Pennebaker, J. Mitchell, *JPEG Still Image Data Compression Standard*, Chapman & Hall, 1993.
- [6] M. McIlhagga, A. Lightm and I. Wakeman, "Towards a Design Methodology for Adaptive Applications," *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 1998.
- [7] <http://www.iig.org/>.
- [8] A. Papoulis, *Probability, Random Variables, and Stochastic Processes, third edition*, McGraw Hill, 1991.
- [9] S. Seshan, M. Stemm, R. Katz, "SPAND: Shared Passive Network Performance Discovery," *Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97) Monterey, CA*, December 1997, pp. 135-46.
- [10] 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.