Server-directed Transcoding

Jeffrey C. Mogul
Compaq Computer Corporation Western Research Laboratory
250 University Avenue, Palo Alto, CA 94301
mogul@pa.dec.com

Transcoding, the conversion between different representations of Web content at an intermediate proxy, can ameliorate mismatches between the complex content provided by an origin server, and the limited bandwidth to, or display capabilities of, a Web client. Existing transcoding systems use implicit information, such as the HTTP Content-type, to control when and how they convert between representations. This approach must balance the risk of losing important information against the intended goal (such as bandwidth reduction), often without sufficient information to make the optimal choice.

A new approach, *server-directed transcoding*, uses explicit guidance from the server to allow a transcoding proxy to make the best possible choice. This paper describes several possible approaches to server-directed transcoding, including *transcoding applets*, the use of mobile code to provide the origin server detailed control over transcoding performed by proxies. The paper discusses security, caching, and other operational issues associated with transcoding applets.

1. Introduction

Web site designers love to provide complex, detailed content, rich with multimedia experiences. Alas, this content often encounters technical limitations between the origin server and the ultimate user: networks might be too slow or expensive, screens might be too small, clients might be underpowered, or the necessary rendering software might be unavailable.

One can cope with this mismatch between content and capabilities by *transcoding* the original representation to a more appropriate representation. Images can be reduced in size or converted to monochrome; text files can be abstracted; video formats can be converted. When used correctly, transcoding (sometimes called *distillation*) can provide the user with the most essential information of the original content, without straining the limits of feasibility.

Transcoding is lossy: while it preserves essential information, it removes inessential or unrenderable information, in order to meet goals such as bandwidth reduction. A transcoding system must often, therefore, make a tradeoff between loss of detail and loss of effectiveness at meeting its goals. Too little distillation, and the bandwidth costs (for example) will still be prohibitive; too much distillation, and the underlying message is lost.

In the Web, transcoding may be done at an intermediate proxy server, or at the ultimate client application. In either case, the transcoding system operates at a distance from the origin server (the source of the content). To make the right tradeoff, the transcoder must have information about what information is essential and what is not. Because it is remote from the origin server, this information is not directly available. Existing transcoding systems use implicit information, such as the HTTP `Content-type` header, to decide whether and how to convert between representations. Implicit information can be ambiguous, which can lead to incorrect decisions.

This paper proposes a new approach to transcoding, called *server-directed transcoding*. In this approach, the origin server provides explicit guidance to the transcoding system (proxy or client) about whether and how to convert between representations. In effect, the server provides hints to the transcoder that can reduce or eliminate the chances of making an inappropriate conversion. The server's directions can be arbitrarily complex, allowing for full extensibility of the transcoding mechanism. These hints, which are provided through a compatible extension of HTTP, are fully optional for the transcoding system.

Server-directed transcoding can be implemented either through the use of simple directives sent in HTTP headers, or through a more powerful mechanism based on applets in Java or similar languages. While the applet-based approach creates some implementation challenges, it simplifies the conceptual issues behind server-directed transcod-

ing, and represents a more natural application of the idea behind active networking.

This paper presents a conceptual overview of server-directed transcoding, including a detailed description of several different approaches, including a proposed extension to HTTP, followed by a discussion of several security and operational issues.

2. Related Work

Transcoding has a lengthy history. Perhaps the first description of a general-purpose approach to transcoding in the Web was the 1995 paper by Brooks *et al.* on HTTP stream transducers [3]. Brooks *et al.* suggested that proxy caches could apply datatype-specific transformations to HTML and other Web content: for example, adding using the BLINK tag to make HTML anchors more visible on monochrome displays. Their proposal used composable transducer ''filters,'' but these were specified at the proxy, not by the origin server.

In a much earlier and much different context, the Experimental Network Operating System project at the National Bureau of Standards [11] implemented a heterogeneous distributed database by interposing a translation mechanism between mutually incompatible database clients and servers.

Fox *et al.* [7] appear to have been the first to suggest the use in the Web of lossy transcoding (which they called distillation) to reduce the bandwidth requirements for images and other bulky content. Again, their approach used datatype-specific conversions, based on the response's `Content-type`, not guided by specific advice from the origin server.

One widely deployed system that apparently uses transcoding is WebTV [19], which allows browsing Web sites while using a standard television screen as the display. Because TV screens have far less resolution than most computer screens, WebTV converts many kinds of Web content so that they will be usable on the low-resolution screen. However, the conversion is apparently datatype-specific, not response-specific.

Chandra and Ellis [5] recognized the need to make transcoding choices based on the quality of the resulting output. They looked specifically at how much compression could be applied to JPEG images without eliminating important information, and how a transcoding proxy might automatically deduce the optimal JPEG quality factor. They also showed how a proxy might estimate both the computational cost of transcoding a JPEG image, and the resulting decrease in representation size.

Relatively little prior work addresses the use of server-supplied hints to guide the use of transcoding. One exception is the HTTP/1.1 `Cache-control: no-transform` feature, which allows an origin server to specify that a proxy must not transform the content of the response [6]. This provides simple protection against loss of vital information (although it is not known if any transcoding proxies actually obey this directive).

HTTP/1.1 also added a requirement that if a proxy transcodes a response, it must add a `Warning` header to that effect; this allows the ultimate client to alert the user to the transcoding. The user may then repeat the request using a `Cache-control: no-transform` request-header, if it is necessary to obtain the unmodified content.

Hori *et al.* [8] propose annotating HTML/XML documents to guide the adaptation (i.e., transcoding) of those documents. Instead of embedding the annotations directly in an HTML file, they use *external* annotations in an XML file. They use XML mechanisms to provide pointers from the annotations back to the adaptable elements of the original HTML file. The proposal is not specific about how a proxy discovers this XML file (that is, how annotations are ''attached'' to the original HTML resource), so this might be an potential application of server-directed transcoding. Also, these annotations are declarative, and thus do not provide the range of transcoding available via a general-purpose procedural language.

The use of applets to dynamically customize the operation of a proxy or client is similar in concept to ''active networks.'' For example, Tennenhouse *et al.* [18] describe the use of servers to provide response-specific instructions to intermediate points in the network, but primarily at the lowest level of the network (the packet level). While Tennenhouse *et al.* mention transcoding, they do not seem to consider the specific possibility of response-specific transformations, or doing these transformations at a higher level in the protocol stack.

Similarly, Bhattacharjee *et al.* [2] hint in the direction of using active networks for transcoding, but apparently they only considered addressing the problem on a packet-by-packet basis. That is, they use the capabilities of active networking to, for example, preferentially drop low-priority frames (packets) of an MPEG video stream. (The intent

of preferential drop policies is to avoid, as much as possible, dropping ‘‘valuable’’ packets during periods of congestion.) They do not seem to have considered the problem of transcoding at the application layer, or of policies that do not involve simply dropping packets or bytes.

Cao *et al.* [4] describe a technique for better caching of Web responses, which uses an HTTP extension to attach an applet to a specific response. In their technique, the applet is used to control caching of the associated response. In some cases, the applet might act to replace one piece of content with another (e.g., ‘‘rotation’’ of advertising banners referenced from a Web page, by editing the cached HTML to refer to different advertisement URLs). They do not appear to have considered using applets for transcoding transformations, and in fact their design seems to specifically prohibit the cache from taking *optional* advantage of an applet.

The IBM WebSphere Transcoding Publisher product [10] provides a framework for proxy transcoding plug-ins, using Java applets and a library of built-in transformations. The proxy is responsible for deciding what to transcode and how; this system does not expect origin servers to provide specific directives.

One can extend cache behavior either by moving the code to the data, or moving the data to the code. Applets (or ‘‘mobile code’’) follows the former strategy; one example of the latter is the Internet Content Adaptation Protocol (ICAP) [9]. ICAP extends HTTP to allow a proxy server to pass HTTP messages (either requests or responses) to a separate ICAP server, for adaptation or filtering. ICAP, unlike server-directed transcoding, does not include a means to specify what adaptations should be applied; this is beyond the scope of the current ICAP design.

The use of ‘‘hints’’ as a way to improve the performance in distributed systems also has a long history, going back at least to Lampson [13]. Mogul wrote about using hints specifically to improve HTTP cache performance [14].

3. Motivation

Previous transcoding systems have used implicit information, usually the HTTP `Content-Type` header field, to guide the decision about whether and how to convert representations. Why is this implicit information not sufficient?

Use of implicit information has several drawbacks. First, since the Web comprises a huge variety of content, any generic conversion algorithm will often choose the wrong tradeoff between optimizing the goals of transcoding (e.g., bandwidth reduction) and the preservation of important information. Second, and perhaps more important, a response-specific transformation might be able to take advantage of a simple but non-generic transformation, to preserve all of the important information while optimizing the transcoding goals.

3.1. An example

It might be easiest to see this point with the help of a simple, albeit somewhat contrived example. Consider the graphic element in figure 3-1, which originally appeared as a GIF image. (The image in the figure has been doubled in size, to improve visibility.) Note that the figure includes foreground text, some artful but redundant background text, and a green arrow on a bluish field.

If the GIF image in figure 3-1 is converted to JPEG with a quality factor of 10, we get the result shown in figure 3-2. The size of this file is about 40% of the original. The foreground text is still legible, although the green arrow now more closely resembles a half-peeled banana.

One can reduce the JPEG quality factor to the absurdly low value of 1, resulting in the image shown in figure 3-3, and a file size 26% of the original. Here, the foreground text (especially the smaller words) is at the border of legibility, and the green arrow has become unrecognizable. However, the background text is still somewhat legible, even though we really don’t care about it.

Instead of converting to JPEG and guessing a quality factor that balances size and legibility, however, we could instead make use of the fact that the original figure has ‘‘foreground’’ and ‘‘background’’ information. We want to preserve the foreground information, yet most of the image complexity is actually in the background of this particular image. We can eliminate the background by a simple transformation on the GIF color map: all dark elements are converted to black, and all light elements are converted to white, while the blue and green colors are preserved. The result, shown in figure 3-4, is 25% of the original size (smaller than figure 3-3), yet is fully legible. Even the green arrow is exactly preserved. (This transformation eliminates the anti-aliasing used for the foreground text, which might have been worth preserving.)

Note that (ignoring the green arrow), the image in figure 3-4 might be the best that one could possibly do for a monochrome display with shallow

File size: 7478 bytes

Figure 3-1: Original GIF

File size: 2974 bytes

Figure 3-2: Converted to JPEG, quality factor = 10

File size: 1978 bytes

Figure 3-3: Converted to JPEG, quality factor = 1

File size: 1859 bytes

Figure 3-4: Reduced-color GIF

pixels; such displays are common on small devices such as PDAs and cell phones.

3.2. Generalizing from the example

We can generalize a few principles from the example in section 3.1.

First, the most appropriate transcoding to apply to an HTTP response might not be obvious from the `Content-type` header. Simply converting from GIF to JPEG might not give the best tradeoff between usability and size. On the other hand, one can easily think of examples where the simple color-reduction algorithm shown above would work very badly.

Second, while it would be cumbersome to create a unique transcoding algorithm for every HTTP resource, this is not necessary. Many sets of distinct resources are sufficiently similar that every member of a set could be transcoded successfully by a single algorithm.

For example, the color-reduction algorithm used above isn't exactly generic (because, among other things, it gives special treatment to the green color used in the arrow), but it certainly does not depend on the specific text represented in the image. It might be applicable to a large set of similar text-bearing images.

Therefore, we can describe a transcoding algorithm as *set specific*, if it can be applied successfully to every member of a set of resources.

Third, a transcoding algorithm can often be compactly represented. If the original GIF image can be produced or transformed so that the entries in its color map are sorted in order of brightness, the algorithm in section 3.1 becomes this transformation on the color map:

1. If the color map entry is ''green'' or ''grayish-blue,'' preserve it.
2. Otherwise, if the color map index is less than a threshold *T*, convert the entry to black.
3. Otherwise, convert the entry to white.

In general, many transcodings need to distinguish between ''foreground'' (important) and ''background'' (superfluous) information. Therefore, if the original image-generation process can use two distinct sets of colors for foreground and background pixels, it should be simple to describe a color-reduction transcoding. More generally, a small conspiracy between the content-generation tools and the authors of transcoding algorithms can yield major benefits.

One might take the example of section 3.1 one step further, by noting that the only significant information in the original image is the text string ''preview the new inline''. This string can be represented in 22 bytes (plus perhaps a few more for a specific font and color), about 0.3% of the size of the original image, yet this preserves essentially all of the original information. (The developer's information for WebTV specifically suggests using actual text instead of text-containing images, but most Web site designers seem unwilling to give up the graphical expressiveness of these images.)

Note that for this trick to work, the client would have to be able to render a text-based response message in place of an inline image. Although this is apparently not currently supported in HTML, it might be useful for a number of similar optimizations.

Could a transcoding system automatically convert between images and text? Of course, a generic text recognition algorithm might work, but it would be expensive, and potentially error prone. One might also simply replace the image with the value of an associated `<ALT>` tag in the referencing HTML document; however, if the GIF image is being transcoded at a proxy, the proxy might not know which `<ALT>` tag goes with the image.

Instead, the content originator might hide the significant text in a comment area of the image encoding (this might require an extension to the encoding format), or in a new HTTP header. Then, the transcoding algorithm would simply be ''replace the image with the hidden text.''

4. Specific approaches to server-directed transcoding

How might server-directed transcoding be implemented in practice? There are at least five challenges to solve:

1. How does the origin server tell the transcoding system about the transcoding algorithm appropriate for an HTTP response?
2. What can a transcoding application do?
3. How can server-directed transcoding be made secure?
4. How does the server choose the appropriate transcoding algorithm (or set of algorithms)?
5. How does the transcoding system decide whether the benefits of transcoding exceed the costs?

Sections 4.1, 5.2, and 5.3 address the first challenge. Section 5.4 addresses the second. Section 5.8 addresses the third. Sections 4.1 and 5.4 touch on the fifth challenge; in general, the origin server can provide estimates of the costs and benefits of transcoding a specific response.

Since the choice of transcoding algorithm is inherently response-specific, it is not possible to provide a generic solution to the fourth challenge; this will require additional work (such as the development of heuristics).

4.1. HTTP header directives

The simplest way for the origin server to provide transcoding guidance to a proxy or client is to define a new HTTP header (which, according to the HTTP specification, would always be ignored by recipients that don't understand it). For example, the GIF image in figure 3-1 might be accompanied by these HTTP headers:

```
HTTP/1.1 200 OK
Date: Tue, 04 May 1999 22:51:34 GMT
ETag: "ZSB3b3VsZCBjb252Z"
Content-type: image/gif
Content-Length: 7478
Transcode-info: convert-ok=image/jpeg,
        min-jpeq-qual=10,
        min-jpeg-size=2974
```

to indicate that it is safe to transcode the image to JPEG as long as the quality factor is at least 10, and that the result will be 2974 bytes. The color-reduction alternative might be expressed as:

```
Transcode-info: gif-remap=black:"0-17";
                white="18-20,22-30"
```

to indicate which color map entries should be converted to black or white.

While this approach requires relatively little mechanism, it also requires servers and clients to agree in advance on the meaning of `Transcode-info` directives. Internet standardization processes work slowly, and this is not conducive to extensibility.

5. Transcoding applets

A far more powerful approach would be to use transcoding applets: set-specific transcoding programs that are downloaded by the transcoding system. In this approach, the transcoding system provides a virtual machine (such as Java or Tcl) for executing a transcoding applet (‘‘mobile code’’) specified by the origin server; this makes the transcoding system fully extensible. Once the mechanism is in place, no further agreement or standardization is necessary.

Transcoding applets are more elegant, and conceptually simpler, than explicit transcoding directives. However, they do require the transcoding system to implement a virtual machine, which introduces its own problems (see section 5.8).

5.1. Division of labor and decision-making

Historically, people have thought of proxy-based transcoding mechanisms as value-added services provided by the proxy implementation. In this model, transcoding techniques are the province of clever proxy designers, who must consider all of the potential complexities, including the safety of the transformations, the choice of which transformations to apply for specific clients, etc.

The use of transcoding applets makes it possible to think instead of each specific transcoding operation as an extension of origin server functionality, not of proxy functionality. That is, we can move away from thinking of transcoding proxies as complex collections of special-purpose extensions, and towards treating them as general-purpose platforms for executing operations on behalf of, and at the direction of, origin servers and clients (as the dictionary definition of ‘‘proxy’’ would imply).

In this view, proxies are important not because of the cleverness of their implementors, but because of their privileged location in the network (i.e., closer to the clients), because of their caching benefits (from aggregating the request streams of many clients), and because of their ability to offload computation from either origin servers or clients, or both. Intelligence about what transformations to apply should reside not in the proxy implementation, but in the origin server and/or client implementations. This is an application of the ‘‘end to end argument’’ [17]:

> The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system.

The use of mobile code to shift the locus of computation to the proxy respects the end-to-end argument, while taking advantage of the performance optimization available through the use of a proxy. (See also [16] for a discussion of the relationship between the end-to-end argument and active networking.)

This argument does not entirely eliminate the decision-making role of the transcoding proxy. The origin server cannot know, in advance, the capabilities of every client that might receive a cached response from a transcoding proxy, so the proxy must combine some information (from the origin server) about the range of possible transformations with information (for each specific client request) about the capabilities and preferences of that client and the intervening network path. The proxy may also use information about the availability of resources (CPU, RAM, etc.) to the transcoding system. With all this information, the proxy can then decide what transformations to apply, if any.

More concretely, the server might supply a set of algorithms rather than a specific conversion. For example, if the server generates a GIF response, it could provide an applet that knows how to convert GIF responses to any one of a set of formats (JPEG, PNG, smaller GIFs). The proxy can select the best transformation, depending on the capabilities of the client and the availability of resources. Note that even this choice could be performed by an applet method, as described in sec-

tion 5.4. Moreover, the client's capabilities and preferences could be embodied in a client-supplied applet, as described in section 5.5.

In summary, the use of transcoding applets supports a model in which the end points (origin server and client) supply application-specific guidance for transcoding, in the form of mobile code. The proxy provides general-purpose execution resources, and a rendezvous point for combining server-supplied guidance specific to the response with client-supplied guidance specific to the client's capabilities.

5.2. Transcoding applets by reference

The server can specify the transcoding applet specific to a response by reference; for example,

```
Transcode-Info: app=
    "http://applets.compaq.com/app37.cls"
```

If the transcoding system wants to use this applet, it downloads the specified URL. This potentially adds an extra round trip to the server. Of course, if the applets are sufficiently general, they can be profitably cached by the transcoding system. This might be true even if the content-bearing response is itself not cachable. The extra round trip might only rarely be necessary.

The example above shows how the applet could be named using a URL. In some cases, it might be more appropriate to use a location-independent name, such as a cryptographic hash (which also would solve a security problem; see section 5.8). Alternatively, an index into a list of registered transcoding applets could be a more compact form of location-independent name.

5.3. Transcoding applets by value

If the transcoding applet is very small, and the round-trip time between the origin server and the transcoding system is very large, it might pay to transmit the transcoding applet by value. That is, it would be included directly in the content-bearing response. However, this eliminates any chances for caching, and seems only beneficial in unusual environments (e.g., servers on Earth, users on the Moon).

5.4. Methods in transcoding applets

Chandra and Ellis [5] showed that a transcoding system should not just blindly apply a content transformation. It should also evaluate the cost/benefit tradeoff, which includes (for example) the expected reduction in content size, and also the expected computational cost for the transformation.

A transcoding applet can therefore provide a number of standardized methods, in addition to the actual transformation method. One method could estimate the cost of applying the transcoding; another could estimate the resulting size, quality, etc. All of the applet's methods could take as input not only the content itself, but also attributes of the recipient, such as the available bandwidth and the capabilities of the client. (These capabilities could be expressed using ''feature sets'' [12], or derived from the HTTP `User-agent` and/or `Accept` headers.)

It might take some careful thought to define a standard for the methods of transcoding applets, to balance extensibility with simplicity.

5.5. Composed transcoding applets

A transcoding applet need not be a monolithic (self-contained) application. In fact, it is quite likely that it will have to rely on some simple standardized methods provided by the proxy platform (e.g., memory allocation). A transcoding applet might also make use of a library of generally-useful components, which, due to their complexity, might be too bulky to incorporate into each applet. For example, an applet might make use of a general-purpose GIF-to-JPEG conversion component, combined with some image-specific transformations (e.g., cropping).

More generally, the behavior of a transcoding applet could be the result of composing the response-specific applet specified by the server with generic components, platform-provided methods, and even client-provided applets. In that approach, a client would specify an applet by reference, in its request headers, that a transcoding proxy could optionally employ to help determine what transformations to apply, or to implement some aspect of the transformations. For example, a PDA client might specify an applet (stored at a server operated by the PDA software supplier) that knows how to optimize audio for the PDA's tinny little loudspeaker.

A transcoding applet could also make use of other network resources, such as databases containing graphical elements. Of course, this might complicate the security concerns described in section 5.8, and would almost certainly complicate the design of standardized interfaces between proxies and applets.

5.6. Other benefits of transcoding applets

Although transcoding is normally thought of as a lossy mechanism, whose goal is to reduce bandwidth requirements, display requirements, etc., it can also be used as an extensibility mechanism. Server-directed transcoding allows a server to introduce a new media format without requiring any further revisions to clients or proxies. For example, if the clients only implement MP3 format for audio clips, but the server has content in AIFF format, it could (in theory) let transcoding proxies do the conversion on demand, by specifying an applet that can do this conversion. If the client's `Accept` request header specifies several potential target content-types, the transcoding applet can choose the most appropriate format from that list.

This approach should improve the abilities of proxies to divert work from origin servers. A transcoding proxy does not only offload the CPU costs for format conversion; because the proxy only needs content in one input format to satisfy many different clients, it improves the probability of a cache hit. This also reduces the need for clients to install media players for all of the various formats favored by site designers (a significant consideration for storage-constrained or zero-administration clients).

5.7. Caching of transcoded results

It would clearly be beneficial to allow caching of transcoding results, to the extent that caching is allowed for the untransformed response. This raises issues both of cache correctness (or ''transparency,'' the criterion that the client does not get the wrong information because of caching), and of cache performance.

For transcoded results, cache correctness depends on three requirements:

1. The semantics of the resource would allow caching of the untransformed responses.
2. The cache can identify the appropriate instance of a resource whose value may change over time.
3. The cache can ensure that a cached transcoded response has been transformed consistent with the requirements of the new request. This should not necessarily require exact matching of the client's capabilities; a ''acceptable'' cached transcoded response might be a better choice than an ''optimal'' newly transcoded response.

The first requirement is met simply by obeying the usual HTTP caching rules, based on the request and response headers for the untransformed response.

The second requirement requires some care, because a transcoded response might be stored at a cache between the transcoding proxy and the requesting client (or possibly at the client). This intermediate cache might not be aware of transcoding at all, and therefore must not be allowed to confuse the transcoded response with an untransformed response; otherwise, it might return the cache entry to a client for which it would not be appropriate.

HTTP/1.1 [6] supports the concept of an ''entity tag,'' which allows a cache to validate a stored response with the origin server. When a response is transcoded, it cannot carry the same entity tag as the untransformed response, or else sub-range retrievals would yield garbled results. But normally the entity tag is assigned by the origin server, and is opaque to intermediate proxies, so HTTP/1.1 does not allow a proxy to modify the entity tag.

One solution would be to allow a transcoding applet to modify the entity tag during the transcoding process, so as to maintain the unique mapping between the response value and the entity tag value; this could be done using a standardized method of the applet. While this appears to contradict the requirement that proxies not modify the entity tag, it is consistent with the view of a transcoding applet as remotely-executed extension of the origin server.

The third requirement also relates to the possibility of an intermediate cache. HTTP/1.1 includes a `Vary` header that allows a response to specify that it depends not just on the URL in the request, but also on other request headers. A transcoding proxy could add a `Vary` header to indicate this dependency. However, this approach might not work; we have recently discovered at least one other plausible application of the `Vary` header where it turns out, on analysis, to be insufficient.

Finally, the storage of transcoded results opens up many interesting questions related to cache performance. The most important may be how to make the tradeoff between storing each transcoded response (which adds I/O costs) and recomputing the transcoded results from a single stored copy of the untransformed response (which adds CPU costs). Previous studies by Acharya *et al.* [1] and by Ortega *et al.* [15] have looked at this problem.

5.8. Security issues with transcoding applets

The use of transcoding applets introduces several security concerns:

1. Can the transcoding applet modify inappropriate data, violate privacy, or generate excessive network traffic?
2. How can the transcoding system limit the resource consumption of a transcoding applet?
3. How can the transcoding system be sure it is using the right transcoding applet?

The first problem is solved, in theory, by languages such as Java, which by design prevent applets from inappropriate access. It might be reasonable to further restrict the capabilities of transcoding applets: for example, to prohibit them from engaging in any network communication at all, and to ignore their output if it is significantly larger than the original content.

The second problem is equivalent to a problem faced in many other active-networking contexts. For example, Cao *et al.* [4] suggest setting upper limits on the resources consumed by an applet. It might be feasible to set a relatively low resource limit on the method used to estimate the cost of a full transformation, and then to use this estimate (with a fudge factor) to limit the resource consumption of the transformation method itself.

The last problem is typically solved through cryptographic means; for example, a message digest such as MD5. In this approach, the origin server might send something like:

```
Transcode-Info: app=
    "http://applets.compaq.com/app37.cls"
      ; app-digest=
        "md5:HUXZLQLMuI/KZ5KDcJPcOA=="
```

giving the transcoding system a relatively secure means to confirm that the applet it downloads (or takes from its cache) is the one intended for use with the content of this response. Unfortunately, there is some unavoidable tradeoff between security and HTTP header length.

5.9. Non-streaming transcoders

A proxy normally should start forwarding a response message as soon as enough data arrives to fill an output buffer. However, while some transcoding operations can be applied in a streaming mode, others might require access to the entire input data. This would require the proxy to enter a store-and-forward mode, and brings up two issues. First, a store-and-forward transcoding applet needs to allocate buffer space on the proxy, in addition to CPU time. Second, this complicates the tradeoff decision, because a store-and-forward transcoding that reduces bandwidth requirements might still increase latency. A proxy might need either administrative or client input to specify relative valuations of bandwidth and latency.

6. Summary and conclusions

Transcoding has shown promise in various contexts, but suffers from inevitable problems when making tradeoffs, in the absence of sufficient explicit guidance, between information loss and transcoding's goals. Server-directed transcoding can eliminate the ambiguities that afflict these tradeoff decisions, and thus protect against any loss of significant information, while simultaneously allowing very aggressive lossy transformations.

Server-directed transcoding applets provide a completely extensible mechanism for proxy-based or client-based transcoding, freeing the use of transcoding from the drag of standardization processes. While some initial standardization of applet methods would be required, the requirements for a transcoding standard are fairly simple. The mechanism is a compatible extension to HTTP, and no other layers need modification.

References

[1] Swarup Acharya, Henry F. Korth, and Viswanath Poosala. Systematic Multiresolution and its Application to the World Wide Web. In *Proc. 15th International Conference on Data Engineering*, pp. 40-49. IEEE Computer Society, Sydney, Australia, March, 1999. http://www.bell-labs.com/user/acharya/papers/icde99.ps.gz.

[2] Samrat Bhattacharjee, Kenneth L. Calvert and Ellen W. Zegura. *On Active Networking and Congestion*. Technical Report GIT-CC-96-02, Georgia Tech, College of Computing, 1996. ftp://ftp.cc.gatech.edu/pub/people/bobby/an/publications/git-cc-96-02.ps.gz.

[3] Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. In *Fourth International World Wide Web Conference*. Boston, MA, December, 1995. http://www.w3.org/Conferences/WWW4/Papers/56/.

[4] Pei Cao, Jin Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proc. IFIP Intl. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pp. 373-388. The Lake District, England, September, 1988. http://www.cs.wisc.edu/~cao/papers/active-cache.html.

[5] Surendar Chandra and Carla Schlatter Ellis. JPEG Compression Metric as a Quality Aware Image Transcoding. In *Proc. 2nd USENIX Symposium on Internet Technologies and Systems*, pp. 81-92. Boulder, CO, October, 1999. http://www.usenix.org/publications/library/proceedings/usits99/chandra.html.

[6] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol -- HTTP/1.1*. RFC 2616, HTTP Working Group, June, 1999. ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[7] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proc. 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 160-170. Cambridge, MA, October, 1996. http://daedalus.cs.berkeley.edu/publications/adaptive.ps.gz.

[8] Masahiro Hori, Rakesh Mohan, Hiroshi Maruyama, and Sandeep Singhal. *Annotation of Web Content for Transcoding*. W3C Note 10, World Wide Web Consortium, July, 1999. http://www.w3.org/TR/annot/.

[9] ICAP Forum. Introducing ICAP: Internet Content Adaptation Protocol. http://www.i-cap.org/icap5.ppt. February, 2000

[10] International Business Machines Corporation. IBM Transcoding Technology: Architecture. http://www-4.ibm.com/software/webservers/transcoding/publications/transcoding.html. 1999

[11] Stephen R. Kimbleton and Pearl Wong. Applications and Protocols. In Butler W. Lampson, M. Paul, and H. J. Siegert (eds.), *Lecture Notes in Computer Science.* Volume 105. *Distributed Systems: Architecture and Implementation.* Springer-Verlag, Berlin and New York, 1981, pp. 308-357. http://www.informatik.uni-trier.de/~ley/db/conf/ac/ds.html.

[12] Graham Klyne. *An algebra for describing media feature sets*. Internet-Draft draft-ietf-conneg-feature-algebra-03.txt (this is a work in progress), IETF media feature registration WG, August, 1998. http://www.ics.uci.edu/pub/ietf/http/draft-ietf-conneg-feature-algebra-03.txt.

[13] Butler W. Lampson. Hints for computer system design. *ACM Operating Systems Review* 15(5):33-48, October, 1983. http://research.microsoft.com/~lampson/33-Hints/Abstract.html.

[14] Jeffrey C. Mogul. Hinted caching in the Web. In *Proc. Seventh ACM SIGOPS European Workshop*, pp. 103-108. Connemara, Ireland, September, 1996. http://www-sor.inria.fr/sigops96/papers/mogul.ps.

[15] Antonio Ortega, Fabio Carignano, Serge Ayer, and Martin Vetterli. Soft Caching: Web Cache. In *Proc. 1997 Workshop on Multimedia Signal Processing*. IEEE Signal Processing Society, Princeton, NJ, June, 1997. http://sipi.usc.edu/~ortega/SoftCaching/mmsp97.html.

[16] David P. Reed, Jerome H. Saltzer, and David D. Clark. Comment on Active Networking and End-to-End Arguments. *IEEE Network* 12(3):69-71, May/June, 1998. http://web.mit.edu/Saltzer/www/publications/endtoend/ANe2ecomment.html.

[17] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end arguments in system design. In *Proc. 2nd International Conference on Distributed Computing Systems*, pp. 509-512. INRIA/LRI, April, 1981. http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.ps.

[18] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine* 35(1):80-86, January, 1997. http://www.tns.lcs.mit.edu/publications/ieeecomms97.html.

[19] WebTV Networks, Inc. WebTV home page. http://www.webtv.com/.

Vita

Jeffrey C. Mogul received an S.B. from the Massachusetts Institute of Technology in 1979, an M.S. from Stanford University in 1980, and his PhD from the Stanford University Computer Science Department in 1986. Jeff has been an active participant in the Internet community, and is the author or co-author of several Internet Standards; most recently, he has contributed extensively to the HTTP/1.1 specification. Since 1986, he has been a researcher at the Compaq (formerly Digital) Western Research Laboratory, working on network and operating systems issues for high-performance computer systems, and on improving performance of the Internet and the World Wide Web. He is a member of ACM, Sigma Xi, and CPSR, and was Program Committee Chair for the Winter 1994 USENIX Technical Conference, and for the IEEE TCOS Sixth Workshop on Hot Topics in Operating Systems.