Adapting to Network and Client Variability via On-Demand Dynamic Distillation

Armando Fox Steven D. Gribble Eric A. Brewer Elan Amir
University of California at Berkeley

The explosive growth of the Internet and the proliferation of smart cellular phones and handheld wireless devices is widening an already large gap between Internet clients. Clients vary in their hardware resources, software sophistication, and quality of connectivity, yet server support for client variation ranges from relatively poor to none at all. In this paper we introduce some design principles that we believe are fundamental to providing "meaningful" Internet access for the entire range of clients. In particular, we show how to perform on-demand datatype-specific lossy compression on semantically typed data, tailoring content to the specific constraints of the client. We instantiate our design principles in a proxy architecture that further exploits typed data to enable application-level management of scarce network resources. Our proxy architecture generalizes previous work addressing all three aspects of client variation by applying well-understood techniques in a novel way, resulting in quantitatively better end-to-end performance, higher quality display output, and new capabilities for low-end clients.

1 Introduction

The current Internet infrastructure includes an extensive range and number of clients and servers. Clients vary along many axes, including screen size, color depth, effective bandwidth, processing power, and ability to handle specific data encodings, e.g., GIF, PostScript, or MPEG. High-volume devices such as smart phones [30] and smart two-way pagers will soon constitute an increasing fraction of Internet clients, making the variation even more pronounced.

These conditions make it difficult for servers to provide a level of service that is appropriate for every client. For example, PDA's with wireless LAN adapters enjoy reasonable bandwidth but have low-quality displays. Full-featured laptops may connect to the Internet via low bandwidth wide-area wireless or wireline modems. Network computers and set-top boxes may not be able to keep up with the entire proliferation of data formats found on the Web.

In this paper we propose three design principles that we believe are fundamental to enabling meaningful access to Internet content for a wide range of clients spanning all these areas of variation. We describe experiments using prototype software as well as a uniform system architecture embodying these principles. The principal underlying idea of the architecture is that *on-demand distillation* (datatype-specific lossy compression) both increases quality of service for the client and reduces end-to-end latency perceived by the client. By performing on-demand distillation in the network infrastructure rather than at clients or servers, we achieve a separation of concerns that confers both technical and economic benefits; by making it datatype-specific, we enable intelligent management of a scarce resource—a slow network link—at the application level.

Platform	SPECint92/ Memory	Screen Size	Bits/pixel
High-end PC	200/16-48M	1280x1024	24, color
Midrange PC	160/8M	1024x768	16, color
High-end laptop	110/16M	800x600	8, color
Midrange laptop	50/8M	640x480	4, gray
Typical PDA	low/2M	320x200	2, gray

Table 1: Physical Variation Among Clients

Network	Bandwidth (bits/s)	Round-Trip Time
Local Ethernet	10 M	0.5 - 2.0 ms
ISDN	128 K	10 -20 ms
Wireline Modem	14.4 – 28.8 K	350 ms (PPP gateway)
Cellular or CDPD	9.6 - 19.2 K	0.1 - 0.5 s

Table 2: Typical Network Variation

In the remainder of this section we outline our design principles and the adaptation mechanisms derived from them. Section 2 describes an architecture that instantiates these mechanisms in the network infrastructure, away from clients or servers. Section 3 presents quantitative evidence that our approach improves end-to-end latency for a large number of cases. Section 4 describes the current implementation status of our architecture and what we have learned from it so far. Section 5 describes related work in various areas, and we present our conclusions in Section 6.

1.1 Client Variation

Clients vary along three important dimensions: network, hardware, and software.

Network Variations include the bandwidth, latency, and error behavior of the network. Much work has been done to optimize network-level protocols for wireless and similar media [5]. In contrast, we focus on reducing bandwidth requirements at the application level. Table 2 shows typical variations in bandwidth seen today.

Hardware Variations include screen size and resolution, color or grayscale bit depth, memory, and CPU power. For example, according to proponents of the Network Computer, "NC's will ride the trailing edge of the price/ performance curve, where they can reap the benefits of plunging prices" [19], which implies that their hardware capabilities will be more modest than those of a typical desktop PC. As Table 1 illustrates, physical variation across current clients spans an order of magnitude.

Software Variations include the application-level data encodings that a client can handle (e.g., JPEG, PostScript,

nonstandard HTML extensions), and also protocol extensions such as IP multicast support [12].

Although we expect clients to improve over time, there will always be older systems still in use that represent relatively obsolete clients, and the high end will advance roughly in parallel with the low end, effectively maintaining a gap between the two. There will always be a large difference between the very best laptop and the very best smart phone.

1.2 Design Principles for Adapting to Variation

We have identified three design principles that we believe are fundamental for addressing client variation most effectively.

1. Datatype-specific lossy compression mechanisms, which we introduce as *distillation* and *refinement,* can achieve much better compression than "generic" compressors. Their intelligent decisions about what information to throw away are based on the *semantic type* of the data. They are therefore more effective adaptation mechanisms than are typeless compressors.
2. It is computationally feasible on today's hardware to perform *"on the fly" adaptation* by computing a desired representation of a typed object on demand rather than relying on a set of precomputed representations. Although there is some latency associated with generating the representation, the *end-to-end* latency from the server to the client can be reduced due to the smaller size of the distilled representation.
3. There are technical and economic reasons to push complexity away from both clients and servers; therefore, on-demand distillation and refinement should be done at an *intermediate proxy* that has access to substantial computing resources and is well-connected to the rest of the Internet.

1.3 High-Level Semantic Types Allow Datatype-Specific Operations

As we describe in Section 3, we have found that datatype-specific lossy compression is an effective adaptation mechanism that can achieved by providing well-defined operations over semantically typed data. For example, lossy compression of an image requires discarding color information, high-frequency components, or pixel resolution. Lossy compression of video can additionally include frame rate reduction. Less obviously, lossy compression of formatted text requires discarding some formatting information but preserving the actual prose. In all cases, the goal is to preserve information that has the highest semantic value. The user can always explicitly ask for a higher-quality representation later if she decides that the data is valuable enough to be worth the additional latency.

Knowledge about datatypes also allows us to reorder traffic to minimize perceived latency. If the user is retrieving a technical paper, prioritizing text content ahead of image content will usually result in the early delivery of the information with the highest semantic value. We can only do this if we can first decompose an object into smaller pieces of different semantic types.

1.3.1 Datatype-Specific Distillation

We define *distillation* as highly lossy, datatype-specific compression that preserves most of the semantic content of a data object while adhering to a particular set of constraints. Table 3 lists the "axes" of compression corresponding to three important datatypes: formatted text, images, and video streams. Of course there are limits to how severe a degradation of quality is possible before the source object becomes unrecognizable, but we have found that order-of-magnitude size reductions are often possible without significantly compromising semantic usefulness.

Semantic type	Some specific encodings	Distillation axes or quality levels
Image	GIF, JPEG, PPM, PostScript figure	Resolution, color depth, color palette
Text	Plain, HTML, PostScript, PDF	Heavy formatting, simple markup, plain text
Video	NV, H.261, VQ, MPEG	Resolution, frame rate, color depth, progression limit (for progressive encodings)

Table 3: Three important types and the distillation axes corresponding to each.

An object's semantic type and its encoding are logically independent, e.g., PostScript can be used to encode either formatted text or a picture. Although the abstract *technique* for distilling a particular object is a function of the object's high-level semantic type, the *implementation* of the distillation technique requires intimate knowledge of the encoding. In practice, constraints may be imposed by specific encodings that admit efficient implementation of specific operations; for example, in the JPEG [20] image encoding, scaling the image by a power of 2 is exceptionally inexpensive. Nonetheless, in general the distillation technique depends on the data type and not the encoding.

Distillation can be thought of as optimizing the object content with respect to representation constraints.

1.3.2 Datatype-Specific Refinement

The primary purpose of a distilled object is to allow the user to evaluate the value of downloading the original, or some part of the original; for instance, zooming in on a section of a graphic or video frame, or rendering a particular page containing PostScript text and figures without having to render the preceding pages. We define *refinement* as the process of fetching some part (possibly all) of a source object at increased quality, possibly the original representation.

As with distillation, the refinement technique is a function of the semantic type, and the implementation of the technique requires intimate knowledge of the encoding. For example, "zooming in" is a useful operation for all images regardless of encoding, but encoding-specific tools are necessary to extract and magnify the desired zoom region.

1.4 Distillation and Refinement On Demand

To reap the maximum benefit from distillation and refinement, a distilled representation must target specific attributes of the client. It is likely that very few clients will impose exactly the same constraints on distillation, especially if the user can decide how much to constrain each distillation axis.

To provide the best possible service to all clients, we should compute each desired representation on demand, rather than attempting to pre-compute a set of representations. Our (non-intuitive) observation, resulting from simulations and implementation of prototypes as described in Section 3, is that distillation time is small and more than made up for by the resulting reduction in transmission time over low-bandwidth links. We have successfully implemented useful prototypes that serve clients spanning an order of magnitude in each area of variation, and we believe our approaches

can be generalized into a common framework, which we discuss in Section 4.2.

1.5 Pushing Complexity Into the Infrastructure

There is growing sentiment that the next wave of successful Internet client devices will be inexpensive and simple [19]. On the other hand, the amount of variation even among existing clients has led to substantial complexity in servers. Fortunately, we can push the complexity away from both clients *and* servers by relocating it into the network infrastructure. Services such as distillation and refinement should be provided by a *proxy*, a source of bountiful cycles that is well connected to the rest of the Internet. For example, an Internet Service Provider connection point or wireless basestation. This arrangement confers technical as well as economic advantages:

- Servers concentrate on serving high quality content, rather than having to maintain multiple versions.
- Servers do not pay the costs required to do on-demand distillation.
- Legacy servers remain unchanged.
- Simple and inexpensive clients can rely on the proxy to optimize content from servers designed for higher-end clients.
- Rather than communicating with many servers per session, the client communicates with a single logical entity, the proxy. This allows fine control over both endpoints of the slow network connecting the client to the proxy. In effect, it allows the client to manage bandwidth at the application level, just as databases manage memory at the application level [23].
- Distillation and refinement can be offered as a value-added service by a service provider. The result is an economic model favorable to service providers, clients, and servers.

1.6 Existing Approaches to Dealing with Variation

Today's Internet servers deal poorly with client variation:

- Some servers ignore client variation, but this can prevent or dissuade lower end clients from accessing the servers' content.
- Other servers use only the most basic data types and minimal graphics to reduce bandwidth and client rendering requirements; sophisticated clients lose their advantages when accessing these.
- Some servers provide multiple formats for downloading a document (e.g., abstract-only, text-only, full PostScript), or multiple versions of a Web page (text-only, moderate graphics, graphics-intensive). This requires additional administration at the server and leaves out a wide middle ground of representations.
- Progressive encodings and the Netscape IMG LOWSRC extension provide ways of retrieving some content at lower quality first, and incrementally refining the quality later. These encodings typically assume that all parts of the encoded document are equally important, so they cannot reorder traffic to present the highest-value content first. In addition, it is often impossible for clients to prevent subsequent refinements from being delivered after the initial request, so in effect the bandwidth for the entire object has already been committed.

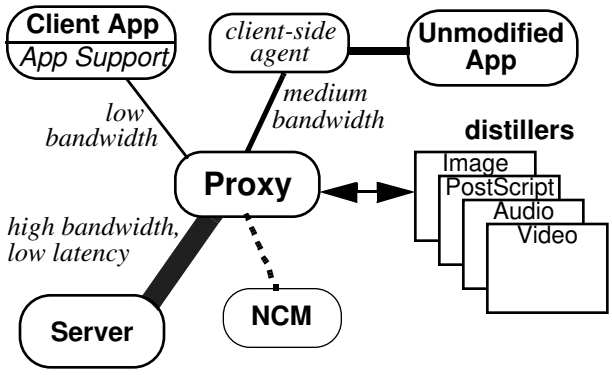

Figure 1: Basic proxy architecture. The proxy uses the distillers to optimize the quality of service for the client in real time. The Network Connection Monitor (NCM) monitors end-to-end bandwidth and connectivity to the proxy's client(s) and notifies the proxy of any changes, which may affect the proxy's transcoding decisions. Client applications are linked with an Application Support library that provides a standard API for communicating with the proxy, or in the case of unmodified (proxy unaware) applications, communicate with the proxy via a "client-side agent".

2 A Proxy Architecture for On-Demand Distillation

To bring our design observations to bear on today's Internet, we propose the proxy architecture in Figure 1. The components are the *proxy*, one or more *datatype-specific distillers*, an optional *network connection monitor*, and the *application support library.*

2.1 Proxy Control Point

A client communicates exclusively with the *proxy*, a controller process located logically between the client and the server. In a heterogeneous network environment, the proxy should be placed near the boundary between strong and weak connectivity, e.g., at the basestation of a wireless mobile network. The proxy's role is to retrieve content from Internet servers on the client's behalf, determine the high-level types of the various components (e.g., images, text runs), and determine which distillation engines must be employed. When the proxy calls a distiller, it passes information such as the hardware characteristics of the client, acceptable encodings, and available network bandwidth. We envision that communication between the client and proxy will use a lightweight transport optimized for slow links, but we offer some support for legacy protocols as well, as we describe in Section 2.4.

2.2 Datatype-Specific Distillers

The datatype-specific distillers are long-lived processes that are controlled by proxies and perform distillation and refinement on behalf of one or more clients. Typically, a distiller will be operating on a single data object at a time, using the constraint information supplied by the proxy. For example, if the client has an 8-bit grayscale display, the target bandwidth is 9600 bps, and delivery is desired in 4 seconds, the image distiller must predict how much resolution loss is necessary to achieve a final representation size of 37.5Kbits. The distiller must account for the reduction achieved by converting from indexed color to 8-bit grayscale as well as the efficiency of that target encoding. In Section 4.4 we describe our first efforts at building distillers that can do this.

2.3 Network Connection Monitor

There are three methods of determining the characteristics of the client's network connection:

1. **User advice.** Via an appropriate user interface, the user notifies the proxy of her expected bandwidth.
2. **Network profile.** The Daedalus project [25,24] is exploring *vertical handoff* (VHO): the seamless migration of a client among heterogeneous wireless networks. VHO allows the client to always switch to the best-available network, which often results in an order-of-magnitude bandwidth change. If the proxy is informed of the VHO, it can activate a new client profile corresponding to the average characteristics of the new network.
3. **Automatic**. A separate process tracks the values of effective bandwidth, roundtrip latency, and probability of packet error, and posts events when some value leaves a "window" of interest.

Clearly, method (3) is the most difficult to implement but has the potential to provide the most responsive behavior to the user, provided the information delivered in the events can be put to good use. Specific API's for delivery of such events are described in [4,36].

Because distilled object representations are generated on demand in our architecture, and because the distillers are assumed to be able to optimize to a particular size of the output, our distillation-based proxy architecture can dynamically react to changing network conditions. For example, a control panel might allow the user to specify a maximum allowable end-to-end latency for delivery of a particular image. The proxy can ask the image distiller to construct a representation that can be delivered within the desired time bound, given the current effective bandwidth estimate.

Since the adaptation mechanism is cleanly separated from the detection of network changes, our architecture can exploit an automated network connection monitor as an optimization, but will still deliver improved service without it.

2.4 Client-side Architecture

Rather than dealing directly with transport and data format issues, we would like the client application to deal with datatypes at the application level, in whatever encodings it finds convenient, and to be able to specify distillation and refinement preferences in an abstract way. One way to do this is via an *application support library* that provides an API with suitable abstractions for manipulating data and interacting with the proxy. The application support library is centered around the process of retrieving distilled documents, and includes abstractions for asynchronous document delivery, specifying which data encodings are acceptable to the client, and affecting the proxy's distillation decisions by specifying weighted constraints on the distillation axes.

Because of the library's rich API, the task of implementing client applications that support our refinement and distillation mechanisms is greatly simplified. Since all communication with the proxy is handled by the support library, applications automatically reap the benefits of an efficient, tokenized transport protocol. We describe a Tcl/Tk [31] implementation of the library in Section 4.3.

Unfortunately, legacy applications must be modified if they are to take advantage of our support library. Our architecture does support unmodified legacy applications with the help of an *client-side agent*. As shown in Figure 1, the client-side agent is a process that runs locally on the client device. It communicates with the remote proxy using the application support library (and therefore benefits from the library's high-level abstractions and optimized transport protocol), but communicates with local applications using whatever protocols and data formats the applications require, such as HTTP.

Original Image (GIF)		Reduce to <8KB		Reduce, +map to 16 grays		Reduce, map, +convert to PICT	
size, KB	colors	size (%)	time	size (%)	time	size (%)	time
48	87	15.0	3.27	7.7	2.18	27.3	2.66
153	254	5.0	6.72	1.9	3.26	5.8	3.73
329	215	1.8	6.17	1.0	5.18	2.1	5.70
492	249	1.5	8.31	<1.0	6.25	1.4	6.75

Table 4: Distillation latency (seconds of wall clock time) and new sizes (as percent of original), for three sets of distillation parameters and four images. Each column is independent, i.e. the third column gives the results for performing all three operations of reduction, gray mapping, and format conversion to PICT. There is an implicit requantization back to the original color palette in the first column, accounting for its higher times.

From the perspective of legacy applications, the client-side agent is functionally equivalent to the remote proxy, but in reality acts as a "protocol filter", efficiently communicating with the remote proxy on behalf of the application. There are limits to the flexibility of this approach; for example, it is awkward to provide a user interface for refinement in most existing Web browsers. Nonetheless, the client-side agent mechanism allows legacy applications to reap many of the benefits of our proxy architecture.

3 Distillation Performance

In this section we describe and evaluate three prototype distillers: images, text, and network video streams. The goal of this section is to support our claim that in the majority of cases, end-to-end latency is reduced by distillation. We do this by demonstrating that distillation performance on today's desktop workstations is sufficiently fast that the time to produce a useful distilled object is small enough to be more than made up for by the savings in transmission time for the distilled object relative to the original.

3.1 Images

We have implemented an image distiller called *gifmunch*, which implements distillation and refinement for GIF [18] images, and consists largely of source code from the NetPBM Toolkit [33]. As described in Section 4.4, *gifmunch* makes simple predictions about the size of its output by measuring the achieved bits per pixel compression of the original image relative to a "raw" bitmap. Figure 2 shows the result of running *gifmunch* on a large color GIF image of the Berkeley Computer Science Division's home building, Soda Hall. The image of Figure 2a measures 320x200 pixels—about 1/8 the total area of the original 880x610—and uses 16 grays, making it suitable for display on a low-end notebook computer. Due to the degradation of quality, the writing on the building is unreadable, but the user can request a refinement of the subregion containing the writing, which can then be viewed at full resolution.

Table 4 shows the latencies of distilling a number of GIF images with three different sets of distillation parameters, and the resulting size reductions. The measurements were taken on a lightly loaded SPARCstation 20/71 running Solaris 2.4. The three sets of distillation parameters were chosen as representative values for addressing the three categories of variation: size reduction to under 8K bytes, color quantizing to 16 grays, and format conversion to Macintosh PICT. The table data reveals three effects of interest:

Figure 2: Left (a) is a distilled image of Soda Hall, and above (b) illustrates refinement. (a) occupies 17K bytes at 320x200 pixels in 16 grays, compared with 492K bytes, 880x600 pixels and 249 colors in the original (not shown). The refinement (b) of the circled area occupies 12K bytes (using 16 grays). Distillation took about 6 seconds and refinement less than 1 second on a lightly loaded SPARCstation 20/71.

- More aggressive color quantization actually takes *less* time than less aggressive quantization. In the first column, the number of colors in the distilled representation was chosen to match the number of colors in the original, e.g., 87 for the first test image. Note that this usually requires the distiller to perform color quantization, since scaling down may introduce a large number of new colors. In the second column, the number of colors in the distilled representation was fixed at 8, chosen from a fixed gray palette. For all four images, the time required to scale and quantize aggressively was less than the time required to scale and quantize less aggressively.
- Format conversion may cause image representation to increase in size, in this case because of PICT's inefficient encoding of bitmaps compared to GIF. This shows how choice of representation impacts bandwidth requirements: if the distiller is informed that the client requires PICT, it must "budget" fewer pixels for the distilled representation.
- The additional work of transcoding to PICT adds virtually no latency to the overall distillation process (compare columns 2 and 3). Adding this step makes sense, e.g., for a PDA client such as the Newton, on which PICT is supported by the native GUI but GIF-to-PICT conversion is computationally expensive.

Image distillation can be used to address all three areas of client variation:

Network variation: The graphs in Figure 3 depict end-to-end client latency for retrieving the original and each of four distilled versions of a selection of GIF images: the top set of bars is for a cartoon found on a popular Web page, and the bottom two sets correspond to the images in the last and first rows of Table 4. The images were fetched using a 14.4Kb/s modem with standard compression (V.42bis and MNP-5) through the UC Berkeley PPP gateway, via a process that runs each image through *gifmunch*. Each bar is segmented to show the distillation latency and transmission latency separately. Clearly, even though distillation adds latency at the proxy, it can result in greatly reduced end-to-end latency. This supports design principle #2.

Hardware variation: The "map to 16 grays" operation in Table 4 is appropriate for PDA-class clients with shallow grayscale displays. We can identify this operation as an effective lossy compression technique precisely because we know we are operating on an image, regardless of the particular encoding, and the compression achieved is significantly better than the 2x-4x typically achieved by "generic" lossless compression (design principle #1).

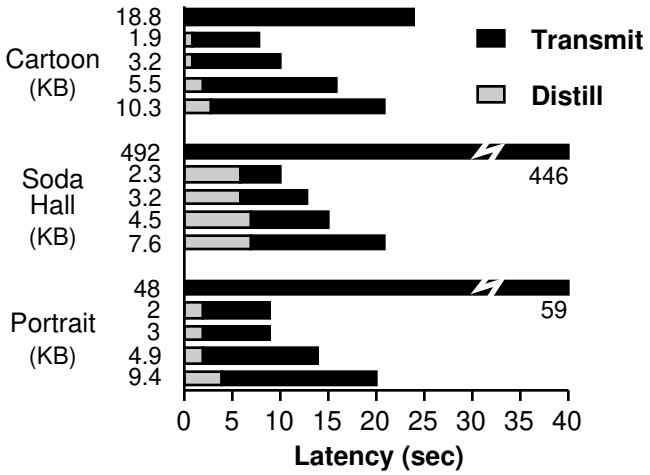

Figure 3: End-to-end latency for images with and without distillation. Each group of bars represents one image with 5 levels of distillation; the top bar represents no distillation at all. The y-axis number is the distilled size in kilobytes (so the top bar gives the original size). Note that two of the undistilled images are off the scale; the Soda Hall image is off by an order of magnitude.

Software variation: Even though PICT offers less efficient bitmap encoding than GIF and is not supported by most servers, conversion to PICT is useful for clients such as the Newton, where PICT is the only graphic format that can be rendered efficiently (design principle #3).

3.2 Rich-Text

We have also implemented a rich-text distiller [27], which performs lossy compression of PostScript-encoded text. The distiller replaces PostScript formatting information with HTML markup tags or with a custom rich-text format that preserves the position information of the words. PostScript is an excellent target for a distiller because of its complexity and verbosity: both rendering and transmission are potentially resource intensive. Table 5 compares the features available in each format. Figure 4 shows the advantage of rich-text over PostScript for screen viewing.

As with image distillation, PostScript distillation yields advantages in all three categories of client variation:

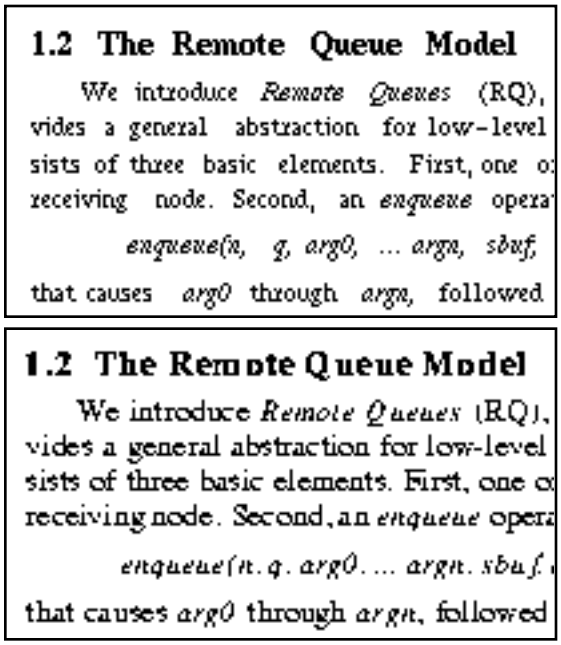

Figure 4: Screen snapshots of our rich-text (top) versus *ghostview* (bottom). The rich-text is easier to read because it uses screen fonts.

Feature	HTML	Rich Text	Post-Script
Different Fonts	Y	Y	Y
Bold and Italics	Y	Y	Y
Preserves Font Size	Headings	Y	Y
Preserves Paragraphs	Y	Y	Y
Preserves Layout	N	Y	Y
Handles Equations	N	some	Y
Preserves Tables	N	Y	Y
Preserves Graphics	N	N	Y

Table 5: Features for PostScript Distillation

Metric	HTML	Rich-Text	Post-Script
Percent Size	18.3%	52.0%	100%
Percent Size after Gzip compression	5.8%	15.4%	26.3%

Table 6: PostScript Distillation Comparison. The size percentages are the geometric mean over the five files listed in Figure 5.

Network Variation: First, distillation greatly reduces the required bandwidth and thus the end-to-end latency, as shown in Table 6. We achieved an average size reduction of 5x when going from compressed PostScript to compressed HTML. Second, the pages of a PostScript document are pipelined through the distiller, so that the second page is distilled while the user views the first page. In practice, users only experience the latency of the first page, so the difference in perceived latency is about 8x for a 28.8K modem, as shown in Figure 5. Distillation typically took about 5 seconds for the first page and about 2 seconds for subsequent pages.

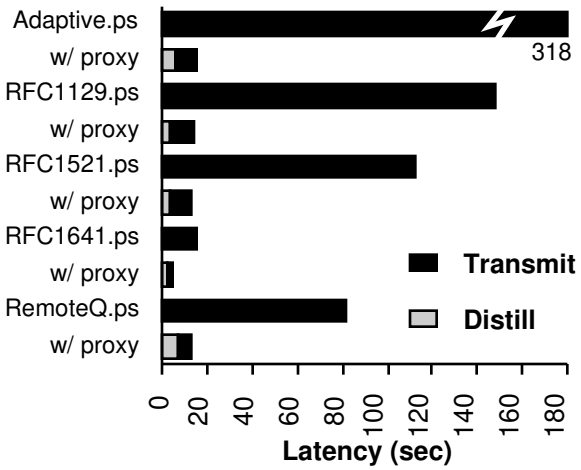

Figure 5: Comparison of end-to-end latency for the first page, with and without a proxy. Source files are uncompressed PostScript; distilled files are uncompressed HTML. The geometric mean of the speedups is 8.1x. (SparcStation20 proxy, PowerPC Macintosh client, 28.8K modem)

Hardware Variation: Distillation reduces decoding time by delivering data in an easy-to-parse format, and results in better looking documents on clients with lower quality displays.

Software Variation: PostScript distillation allows clients that do not directly support PostScript, such as Macintoshes or PDA devices, to view these documents in HTML or our rich-text format. The rich-text viewer could be an external viewer similar to *ghostscript*, a module for a Java-capable browser, or a browser plug-in rendering module.

The PostScript distiller will support two kinds of refinement. First, users can request a particular page at higher quality. Second, if the users are in rich-text mode (which preserves layout), they can refine a region by marking it with a rectangle. This is particularly useful for viewing figures and equations; the rich-text format tends to have blank regions where the figures go, so it is easy to know what to refine.

Overall, rich-text distillation reduces end-to-end latency, results in more readable presentation, and adds new abilities to low-end clients, such as PostScript viewing. The latency for the appearance of the first page was reduced an average of 8x using the proxy and PostScript distiller. Both HTML and our rich-text format are significantly easier to read on screen than rendered PostScript, although they sacrifice some layout and graphics accuracy compared to the original PostScript.

3.3 Real-time Video Streams

Distillation of real-time traffic introduces an additional degree of freedom with respect to non-real-time traffic types, namely, the temporal dimension. This affects the distillation process of real-time traffic in two ways. First, it allows the distiller to perform distillation in this dimension (commonly called *temporal decimation*), such as limiting the frame rate to meet a target bit rate. Second, temporal dependencies impose tight timing constraints on the distillation process, which affects the architecture of the distiller.

In the spatial domain, we can still perform operations similar to those used in the image distiller: resolution scaling, requantization,

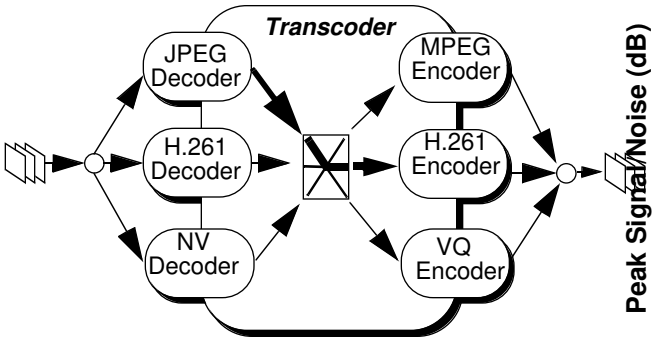

Figure 6: The design of the video distiller. Any of several supported input formats can be converted into any supported output format. As an example, the figure shows a JPEG/H.261 configuration. The intermediate stage performs a transformation on the output of the decoder when necessary, to match the conventions of the decoder and encoder and hands it to the appropriate encoder. In addition to any bandwidth reduction inherent in format conversion, the output can be rate-controlled by decoupling the generation of output frames from the arrival of input frames.

color decimation, or recoding in a format that is more compressible or better matches the characteristics of the end client.

As with all distillation operations, a central challenge in the distillation process is to optimize the perceptual quality of the resulting stream as a function of the possible distillation operations and external timing constraints. The video distiller follows the model of pushing complexity away from the client and trading bandwidth for quality. The additional challenge introduced in the real-time case is that temporal efficiency (the analogue of reducing end-to-end latency for non-real-time distillers) is not an optimization, but a requirement.

The video stream distiller used in our architecture is the video gateway, *vgw*. Figure 6 presents a high level schematic of its design. At the core of the gateway we have a distiller that can transcode between arbitrary input and output formats. The gateway controls the output rate of the resulting stream by applying any of the above mentioned techniques. Section 4.5 gives some implementation details; for a complete description, the reader is referred to [2]. Figure 7 illustrates the relationship between the output frame rate and output quality, as measured by the peak signal-to-noise ration (PSNR) of the resulting frames. The figure plots several sizes of images at a given bit-rate, showing a four way trade-off between quality, frame-rate, spatial decimation, and bit-rate.

Although *vgw*'s distillation techniques are different from those of the image and rich text distillers, they clearly address the three areas of client variation:

Network Variation: By throttling the frame rate, we can meet any target bit rate. We can also affect the output bit rate by altering encoding parameters such as quantization factors.

Hardware Variation: Since *vgw* can decode the incoming video stream to pixel domain, its output encoding can be tailored to meet client screen limitations. For example, *vgw* can provide halftoned output to a device with an 8-bit grayscale display, trading off compression performance for rendering complexity. *Vgw* can also accommodate devices with high bandwidth but limited CPU rendering speed by throttling the frame rate.

Software Variation: *Vgw* converts among NV, MJPEG, H.261, and an error-tolerant form of VQ used by the Berkeley InfoPad [8], and it appears to clients as a video source. This enables the InfoPad to participate in MBONE broadcasts, even though it employs a proprietary video format, and it does not support IP multicast. Thus several aspects of software variation are addressed without modifying or even notifying the server.

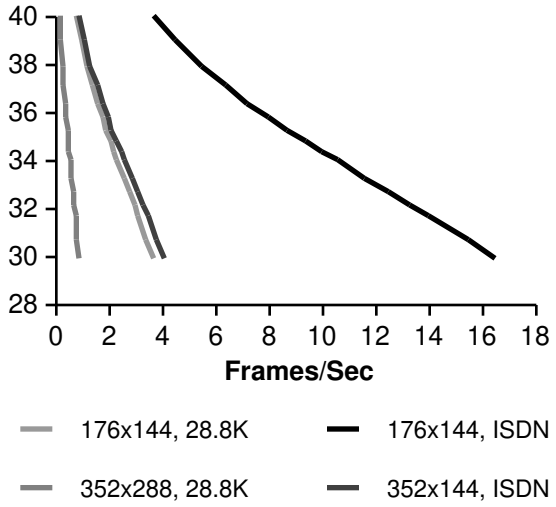

Figure 7: Four-way trade-off among frame rate, quality (PSNR), frame size, and bandwidth. This data is for H.261 encoding to 8-bit grayscale, and assumes there is *no temporal compression,* such as conditional replenishment. Color frame rate would be about 1.5x slower (using 4:1:1 YUV).

Motion	JPEG to H.261	Nv to H.261
Low (talking head)	30 frames/s	30 frames/s
High (panning camera)	27 frames/s	4.5 frames/s

Table 7: *vgw* performance on low- and high-motion streams, on a SPARCstation 20/71 with Solaris 2.4. Conversion is 320x240 to 352x288. High motion Nv to H.261 transcoding is constrained by the Nv encoder output rate in our experiment.

We are also adding refinement capabilities to *vgw*. In particular, clients will be able to zoom into a part of a stream and view that subset at higher quality. We can also dynamically control the trade-off among frame rate, color depth and resolution. For example, when the content is relatively static, such as overhead slides, we can reduce the frame rate and increase the resolution. As with images and rich-text, these techniques exploit datatype-specific knowledge to maximize the semantic content of the stream for each client, and enable video delivery to clients for which it would otherwise be impossible.

3.4 Scalability Concerns

The previous sections have demonstrated that modern workstations are sufficiently fast that distillation time is often small compared to time saved in transmission of a distilled object. In our architecture, multiple distillation operations can be serviced in parallel by a single pool of computing resources [3] on behalf of a potentially large set of clients.

To explore how well our architecture scales to large numbers of clients, we simulated the load placed on a WWW proxy by a parameterizable number of users. For the purposes of this simulation, our web proxy was modeled as a single 80-MHz HP PA-RISC workstation that serviced all image distillation requests but was otherwise unloaded. The simulator used performance characteristics of our image distiller (as measured on the same workstation) to model the distillation latency perceived by the user as a function of the image size and the number of simultaneous image distillations cur-

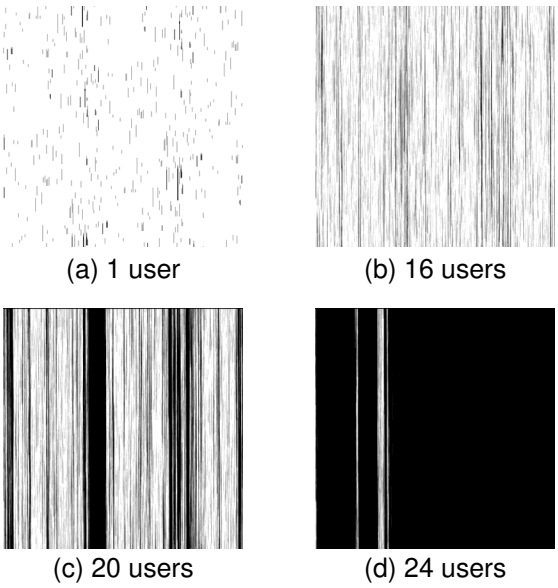

Figure 8: Each square represents the utilization of the proxy over time for a different number of simultaneous users. Time flows vertically up a column and then up its neighboring column. Each column represents 15 seconds; the entire figure represents 4500 seconds of activity. Black vertical stripes imply that clients are waiting for the distiller. At 16 users the proxy works reasonably well, while at 24 users the system is unusable.

rently in progress on the same machine. Input to the simulator came from the UC Berkeley Computer Science Division's HTTPd logs.

Image distillation is a CPU-bound task, since the process of image reduction and requantization requires a distiller to "touch" all of the pixels in an image many times. We observed that the latency of distillation was a linearly increasing function of the number of simultaneous operations, with a slope approximately proportional to the size of the original GIF (in bytes). Because *N* distillers shared the workstation's CPU equally, each distillation operation took *N* times as long to complete.

Recent work [32,11] strongly suggests that access to WWW documents is bursty. For example, an access to a new page causes a flurry of distillation operations. Since a user tends to digest the document for a period of time before moving on, there are variable-length periods of inactivity between distillations performed on behalf of that user. Burstiness results in good utilization of our compute resources, as the bursts of activity from multiple users tend to not overlap, as long as the compute server is not overloaded. Figure 8a shows an example of bursty distillation activity generated by a single user from our HTTPd logs; black represents many distillations occurring concurrently and white represents inactivity.

Adding more users to a proxy increases the "blackness" of the figure. If two distillation bursts collide, each will take twice as long to complete; this greatly increases the probability of further coincident bursts if more users are added. As the number of users increases, the black regions in the figure tend to smear out and merge (Figure 8b and c), until finally the entire figure is uniform black (Figure 8d). At this point, the proxy becomes overloaded, and distillation requests arrive faster than they can be serviced.

Figure 9 shows the simulated average latency of image distillation perceived by a user as a function of the number of users supported by a single workstation, on a logarithmic scale. At approximately 20 users, requests arrive faster than they are serviced, and beyond this point, distillation latency is unbounded for the single workstation. Nonetheless, this result suggests that even using today's desktop hardware, document access patterns (at least

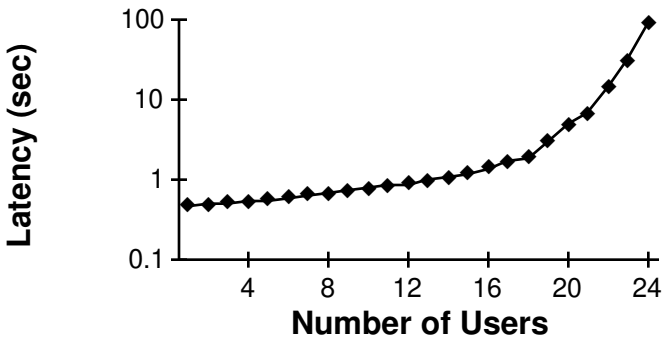

Figure 9: Image distillation latency versus the number of simultaneous users on one proxy.

for the Web) allow multiple users to be served by one compute server in a proxy installation. We have begun investigating how to balance a distillation workload across multiple distillation "servers" in a network of workstations.

4 Implementation and Experience To Date

In this section we describe our experiences to date in implementing various pieces of our architecture, what lessons we have already learned along the way, and what we see as areas of continuing work.

4.1 Pythia HTTP Proxy

We began with a distillation World Wide Web proxy, which we reported on in [14]. It demonstrated the feasibility of our design principles, and is useful despite the fact that much of the code is naive. Pythia is minimally scalable in that it can exploit a small additional number of workstations to share the distillation workload.

4.2 GloMop: A Modular Proxy

We have partially implemented a modular proxy server called *GloMop*[1], which implements the API alluded to in Section 2.4. GloMop's document abstraction is a partially ordered collection of *chunks*, each of which consists entirely of a single datatype and encoding. For example, a Web page is a document typically consisting of one or more *text/html* and *image/** chunks. The modular proxy can convert each semantic type (only text and images so far) to a common intermediate representation (a subset of HTML for text, and PPM [33] for images), distill the intermediate representation, and convert it to a different target representation for the client if desired.

Distillation of images adheres to user-specified constraints to the extent they are supported by our image muncher (Section 4.4). GloMop is modular in that the document-centric framework makes it easy to add new transcoding modules and new "client profiles" describing specific hardware properties (e.g., screen size, aspect ratio, color palette) of client devices.

GloMop is also designed to be scalable to large numbers of users by exploiting the observations that we reported in Section 3.4. The actual work of distillation can be off-loaded to other nodes in a network of workstations [3]. We are actively investigating how to do load balancing for such a system. GloMop also provides authentication and secure channels using Charon, a lightweight indirect authentication protocol based on Kerberos IV. In [15] we show that Charon requires little trust to be surrendered to the proxy and is amenable to implementation even on very small PDA-class devices.

1: From *Global Mobile Computing By Proxy.*

4.3 Client-side Implementation

We have implemented a client-side application support library in C, on top of which is layered language-specific "glue" to the programming environment of your choice (currently Tcl/Tk, via a shell called *gmwish*). We are using *gmwish* as a research testbed for exploring the effectiveness of our API, and for testing the implementation and performance of our core C library and modular proxy. To this end, we have implemented an image browser application that can retrieve and refine WWW images. We chose this application because it was easy to implement using Tk, and because it exercises the entire proxy API. The major functions provided by the API include requesting document delivery, providing asynchronous notification of document arrival, specifying which data encodings are acceptable to the client, and indirectly controlling distillation by specifying weighted constraints on the distillation axes.

It is difficult to design an effective user interface for controlling many semantically orthogonal distillation axes. In our image browser, the user can specify a desired exact value for any one of the download time, resolution of the image, and color depth of the image, and specify the relative importance of the remaining two. However, the conversion of these constraints to distillation parameters at the proxy is not as complete as it should be, and we expect this characterization to lead to significant future work.

Perhaps unsurprisingly, we observed that asynchronous data delivery is the largest source of complexity in the library and in applications that use it. For extremely impoverished clients, the implementation overhead of this complexity is likely to outweigh the benefits of asynchronous delivery notification. We are designing a lightweight version of the application support library that has the smallest useful subset of the full functionality for small clients.

4.4 Image Distiller

The image distiller we use is constructed largely from source code in NetPBM [33]. Currently the distiller picks a color palette based on the known capabilities of the client (which identifies itself when it first connects to the proxy and establishes a session), and optimizes for a particular target size in bytes of the distilled representation by predicting compression. Prediction is done by observing the expansion when converting the original image to the PPM intermediate form, and multiplying this by an encoding-specific "expansion ratio" based on the effective bits per pixel achieved in past runs using the same target encoding. The distiller is typically able to meet this output constraint within a margin of about 10-15%.

There are several motivations for choosing output representation size as the optimization target, including targeting a particular latency bound based on known network bandwidth or observing a maximum buffer size in the client (the latter is particularly important for PDA's). In practice, however, the user might want to optimize for a different constraint, e.g., sacrificing color to achieve better resolution while maintaining a roughly constant representation size. We recognize that in general, mapping a set of weighted constraints to a set of input parameters for a distiller is a nontrivial optimization problem. Our goal is to provide a general self-training mechanism, whereby a statistical model of the distillation process could be used to predict achieved compression or latency of performing the distillation, given a particular input document and set of distillation parameters. We do not currently have such a general mechanism, but it is a major subject of continuing work. We are exploring both automated statistical modeling [7] and neural networks as tools for tackling this problem.

4.5 *vgw* Video Stream Distiller

As initially described in Section 3.3, the basic *vgw* model involves transcoding from some set of supported input formats to a (possibly different) set of output formats. Each input format is handled by a module that decodes the incoming bit stream into an intermediate representation, which is transformed and delivered to an encoder that produces a new bit stream in a potentially new format. *Vgw* joins two separate stateless real-time protocol (RTP) sessions and properly transforms the RTP data and control streams.

Requiring every transcoder configuration to decompress all the way to the pixel domain, and then re-encode the stream from scratch, would impose a large performance penalty. Instead, multiple intermediate formats are allowed. The choice of intermediate format is a function of the encoder/decoder pair, determined by the lowest complexity decoder/encoder data path. In this way, encoder/decoder pairs can optimize their interaction by choosing an appropriate intermediate format. For example, DCT-based coding schemes like JPEG [20] and H.261 [35] can be more efficiently transcoded using DCT coefficients, which avoids the slow conversion to the pixel domain.

The intermediate format also allows us to perform transformations on the canonical image data, such as temporal and spatial decimation and frame geometry conversion (e.g., to handle different resolutions) and color decimation conversion (to handle different chrominance plane downsampling schemes).

Vgw has been in service in "production mode" at CERN, as their transatlantic MBONE gateway. It is also in use by the Berkeley InfoPad [8], transcoding MBONE video to InfoPad VQ format, thus allowing the MBONE video broadcasts to be viewed on the InfoPad.

4.6 Other Applications

The proxy architecture may be appealing for service providers that would like to enable their subscribers to use low-cost clients. It has been said that "…the ultimate fate of Network Computers may depend on the adoption rate of technologies such as... ATM and cable data modems" [19]. A "proxied" NC architecture mitigates this limitation, however, since the existing telephone and wireless infrastructure can be used to provide much better service via a proxy than could be obtained from the "raw" network.

Because the proxy architecture is designed to address the limitations of an extremely wide range of clients, it is a good candidate for a system to deliver Internet content via cable TV converter boxes, even if those clients enjoy connectivity at cable-modem speeds. The appeal of Internet-from-your-TV will be significant if the client computer can be integrated into the cable converter box, which cable subscribers do not need to purchase. This design requirement leads to severe hardware and software constraints, which our proxy architecture is uniquely positioned to address. We are working with Wink Communications, Inc. [38] to build and deploy such a system, allowing users of properly equipped cable converter boxes to access Internet content using their TV and remote control.

5 Related Work

None of the techniques discussed in this paper is fundamentally new on its own. We view our contribution as the generalization of existing techniques into a uniform architecture, in which distillation on demand is used is used to adapt to ever-increasing client variability along three distinct axes, all in a medium still undergoing explosive growth. In this section we discuss work in the three areas our architecture spans: content transcoding, adapting to poor networks, and shifting application complexity away from small clients.

5.1 Transcoding Proxies

The idea of placing an intermediary between a client and a server is not new. The original HTTP specification [6] explicitly

provides a proxy mechanism. Though it was originally intended for users behind security firewalls, it has been put to a number of novel uses, including Kanji format transcoding [34], Kanji-to-GIF conversion [39], and rendering equations from markup [40]. The Distributed Clients Project [13] is also exploiting application-level stream transducers [9] as one of several mechanisms for facilitating Web browsing with intermittent connectivity.

5.2 Shielding Clients From Effects of Slow Networks

On-the-fly compression, especially for protocol metadata at the network level, has long been used to mitigate the effects of slow networks [21,16]. Various network- and transport-level optimizations have also been used to address the wireless case [5], which has propagation and error characteristics quite different from those of most wired networks.

The Odyssey system [29], on the other hand, supports a form of end-to-end bandwidth management by providing a fixed number of representations of data objects on a server, and specifying an API by which clients track their "environment" (including, e.g., network characteristics) and negotiate for a representation that is appropriate for their current connectivity. The approach as described requires substantial changes (content, filesystem organization, control logic, and kernel modifications) to the server, and does not accommodate clients whose configurations suggest a data representation somewhere in between those available at the server. Nonetheless, a distillation proxy could negotiate with an Odyssey server for a representation that would minimize the additional work the proxy would need to do on behalf of its client.

5.3 Hybrid Network- and Application-Level Approaches

We know of at least two projects that combine network-level optimizations with at least some application-level content filtering. MOWGLI [26] provides both a proxy and a client-side agent, which cooperate to manage the wireless link using an efficient datagram protocol, hide disconnection from higher network layers, and tokenize application level protocols and data formats such as HTTP and HTML to reduce bandwidth requirements. However, MOWGLI's protocol-level lossless compression stands in contrast to our document model's semantic lossy compression, and MOWGLI cannot dynamically adapt its behavior to changing network conditions.

Bruce Zenel's "dual proxy" architecture [41] also features separate low-level and high-level filters, which can be demand-loaded by applications. The low-level filters operate at the socket API level and require modifications to the mobile device's network stack. The high-level filters can use application-specific semantics to filter data before it is sent to a client, but the filter is part of the application rather than a middleware component, which complicates its reuse by other applications and makes it awkward to support legacy applications.

5.4 Partitioning of Application Complexity

Rover [22] provides a rich distributed-object system that gives a uniform view of objects at the OS level, and a queued RPC system that provides the substrate for invoking on objects. Together these abstractions allow disconnection and object migration (including code) to be handled largely implicitly by the OS. For example, simple GUI code can be migrated to the mobile, where it will use queued RPC to communicate with the rest of the application running on a server. Rover's goals and functionality are complementary to our own, and nothing precludes the composition of queued RPC and RDO's with the functionality of our proxy architecture.

Wit [36] partitions mobile applications between a client running threaded Tcl on an HP palmtop, and a workstation-based proxy process. However, Wit 1 did not emphasize bandwidth management (though nothing in the Wit architecture precludes its use on a per-application basis), nor did it specify a uniform architecture for application partitioning. Wit version 2 [37] adds a uniform architecture in which client data is treated as a graph of objects constructed by the proxy, where graph edges connect "related" data objects (e.g., sequential or threaded messages in a mail queue). Bandwidth management can be achieved by explicitly pruning the graph, e.g., lazily fetching subsequent messages in a mail thread, rather than prefetching them in the initial communication with the proxy.

5.5 Cooperative Caching and Prefetching

Cooperative caching relays [17,28,1,10] and prefetching can reduce the latency seen by the client and server-to-cache bandwidth requirements, but do not address cache-to-client bandwidth (e.g., the client is connected to the caching relay via a slow link) or client hardware/software variation. We believe that distributed cooperative caching will ultimately be necessary for managing the explosive growth of the Internet, but not for reducing overall latency and bandwidth to the client. The proxy does, however, provides a natural location for cooperative caching of distilled and undistilled data, exploiting locality among a group of institutional users connected to the same proxy.

6 Conclusions

High client variability is an area of increasing concern that existing servers do not handle well. We have proposed three design principles we believe to be fundamental to addressing variation:

- Datatype-specific distillation and refinement achieve better compression than does lossless compression while retaining useful semantic content, and allow network resources to be managed at the application level.
- On-demand distillation and refinement reduce end-to-end latency perceived by the client and are more flexible than reliance on precomputed static representations.
- Performing distillation and refinement in the network infrastructure rather than at the endpoints separates technical as well as economic concerns of clients and servers.

We have also described a proxy architecture based on these design principles that has the ability to adapt dynamically to changing network conditions. Our architecture provides a generalized framework for simultaneously addressing three independent categories of client variation by applying well-understood techniques in a novel way.

Our preliminary results, based on both implemented prototypes and trace-driven simulations, confirm the efficacy of our approach. In particular, on-demand distillation leads to *better performance*, often including an order of magnitude latency reduction, *better looking output* (targeted to the particular client screen), and *new abilities* such as video access or PostScript viewing for low-end devices.

As new and varied Internet clients become available in volume, we expect that value-added proxy services based on this architecture will play an increasingly important role in the network infrastructure.

7 Acknowledgments

This paper was tremendously improved by the suggestions from our anonymous reviewers and the guidance of Larry Peterson. We received valuable comments from our UC Berkeley colleagues, especially Eric Anderson,Trevor Pering, Hari Balakrishnan, and Randy Katz. This research is supported by DARPA contracts

#DAAB07-95-C-D154 and #J-FBI-93-153, the California MICRO program, the UC Berkeley Chancellor's Opportunity Fellowship, the NSERC PGS-A fellowship, Hughes Aircraft Corp., and Metricom Corp.

8 Bibliography

[1] M. Abrams *et al.* Caching Proxies: Limitations and Potentials. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, USA, Dec 1995.

[2] E. Amir, S. McCanne, and H. Zhang. An Application Level Video Gateway. *Proceedings of ACM Multimedia 1995*, San Francisco, CA, USA, Nov 1995.

[3] T. E. Anderson *et al.* The Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, Feb 1995.

[4] B. R. Badrinath and G. Welling. Event Delivery Abstractions for Mobile Computing. *Rutgers Univ. TR #LCSR-TR-242*.

[5] H. Balakrishnan *et al.* A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. To appear, *Proceedings of the 1996 ACM SIGCOMM Conference*, Stanford, CA, USA, Aug 1996.

[6] T. Berners-Lee *et al.* Hypertext Transfer Protocol - HTTP/1.0. RFC 1945, May 1996.

[7] E. A. Brewer. High-Level Optimization via Automated Statistical Modeling. *Proceedings of Principles and Practice of Parallel Processing (PPoPP '95)*, Santa Barbara, CA, USA, July 1995.

[8] B. Brodersen *et al.* Infopad: A System Design for Portable Multimedia Access. *Proceedings of Calgary Wireless '94 Conference*, Calgary, Canada, 1994.

[9] C. Brooks *et al.* Application-specific Proxy Servers as HTTP Stream Transducers. *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, USA, Dec 1995.

[10] C. M. Bowman *et al*. The Harvest Information Discovery and Access System. *Proceedings of the Second International World Wide Web Conference*, Chicago, IL, USA, Oct 1994.

[11] M. E. Crovella and A. Bestavros. Explaining World Wide Web Traffic Self-Similarity. *Boston University Technical Report TR-95-015*.

[12] S. Deering. Host Extensions for IP Multicasting. RFC 1112, Aug 1989.

[13] The Distributed Clients Project. http://www.osf.org/www/dist_client.

[14] A. Fox and E. A. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-time Distillation. *Proceedings of the Sixth International World Wide Web Conference*, Paris, France, 1996.

[15] A. Fox and S. Gribble. Security on the Move: Indirect Authentication Using Kerberos. To appear, *Proceedings of the ACM MobiCom 96 Conference*, White Plains, NY, November 1996.

[16] J. Fulton and C. Kantarjiev. An Update on Low Bandwidth X (LBX). *The X Resource*, 1(5):251-266, Jan 1993.

[17] S. Glassman. A Caching Relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27(2), Nov 1994.

[18] Graphics Interchange Format Version 89a (GIF). CompuServe Incorporated, Columbus, Ohio, July 1990.

[19] T. R. Halfhill. Inside the Web PC. *Byte Magazine*, pp. 44–56, March 1996.

[20] ISO DIS 10918-1 Digital Compression and Coding of Continuous-Tone Still Images (JPEG). CCITT Recommendation T.81.

[21] V. Jacobson. Compressing TCP/IP Headers for Low-Speed Serial Links. RFC 1144, Feb 1990.

[22] A. D. Joseph *et al.* Rover: A Toolkit for Mobile Information Access. *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, Dec 1995.

[23] J. Kaplan. Buffer Management Policies in a Database System. M.S. Thesis, The Univ. of Calif., Berkeley, Calif., 1989.

[24] R. H. Katz and E. A. Brewer. The Case For Wireless Overlay Networks. *SPIE Multimedia and Networking Conference (MMNC'96)*, San Jose, CA, USA, Jan 1996.

[25] R. H. Katz and E. A. Brewer *et al.* The Bay Area Research Wireless Access Network (BARWAN). *Proceedings Spring COMPCON Conference 1996*, 1996.

[26] M. Liljeberg *et al.* Enhanced Services for World Wide Web in Mobile WAN Environment. *University of Helsinki CS Technical Report No. C-1996-28*, April 1996.

[27] P. MacJones, Dec SRC. Personal Communication.

[28] R. Malpani, J. Lorch, and D. Berger. Making World Wide Web Caching Servers Cooperate. *Proceedings of the Fourth International World Wide Web Conference*, Dec 1995.

[29] B. D. Noble, M. Price, and M. Satyanarayanan. A Programming Interface for Application-aware Adaptation in Mobile Computing. *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, MI, USA, Apr 1995.

[30] Nokia Communicator 9000 Press Release. Available at http://www.club.nokia.com/support/9000/press.html.

[31] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[32] V. Paxson and S. Floyd. Wide-area Traffic: The Failure of Poisson Modeling. *ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, London, UK, aug 1994.

[33] J. Poskanzer. NetPBM Release 7. ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM, 1993.

[34] Y. Sato. DeleGate Server. Documentation available at http://www.aubg.edu:8080/cii/src/delegate3.0.17/doc/Manual.txt.

[35] Video Codec for Audiovisual Services at p*64kb/s, 1993. ITU-T Recommendation H.261.

[36] T. Watson. Wit: An Infrastructure for Wireless Palmtop Computing. *Technical Report CSE-94-11-08*, University of Washington, Nov 1994.

[37] T. Watson, B. Bershad, and H. Levy. Using Application Data Semantics to Guide System Network Policies. SOSP '95 WIP Session, 1995.

[38] Wink Communications, Inc. http://www.wink.com.

[39] K.-P. Yee. Shoduoka Mediator Service. http://www.lfw.org/shodouka.

[40] K.-P. Yee. MINSE. http://www.lfw.org/math.

[41] B. Zenel. A Proxy Based Filtering Mechanism for the Mobile Environment. *Ph.D. Thesis Proposal*, Department of Computer Science, Columbia University, March 1996.