

Squid-based proxy server for content adaptation*

Claudia Canali

Dip. di Ingegneria dell'Informazione

Università di Parma

claudia@weblab.ing.unimo.it

Valeria Cardellini

Dip. di Informatica, Sistemi e Produzione

Università di Roma "Tor Vergata"

cardellini@ing.uniroma2.it

Riccardo Lancellotti

Dip. di Informatica, Sistemi e Produzione

Università di Roma "Tor Vergata"

riccardo@weblab.ing.unimo.it

Università di Roma "Tor Vergata"

Dept. of Computer Engineering

TR-2003-03

January 2003

Abstract

The growing popularity of Internet and technology advancements have led to the appearance of many different Web-connected devices. In such an heterogeneous client environment, the resource adaptation to a broad range of device capabilities is becoming a major requirement for the future Web. In this paper we describe the architecture of Squid-based prototypes that carry out the adaptation of images of several formats operating on various parameters, such as spatial geometry, color depth, quality factor, and MIME subtype. The realized prototypes combine the content adaptation functionality to the caching of multiple versions of the same resource. We also investigate how to realize cooperative architectures of proxy servers organized in hierarchical and distributed topologies to evaluate to what extent the proxy collaboration in discovering, transcoding, and delivering Web content can furtherly improve the user-perceived performance.

*This Report has been submitted for publication and will be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. No part of its text nor any illustration can be reproduced without written permission of the Authors.

1 Introduction

In the last years Internet has enriched more and more its functionalities thus becoming the support to many different communication services and penetrating deeply in the daily life of a huge number of users. This increasing popularity, combined with the technology development, allows users to be connected to the Web through many devices, which differ from traditional PCs, such as laptops, handheld computers, TV browsers, Personal Digital Assistants (PDAs), and mobile phones. These emerging Web-connected devices differ considerably in network connectivity, processing power, storage, display, and format handling capabilities. Hence, there is a growing demand for solutions that enable the transformation of Web content for adapting and delivering it to diverse destination devices.

The process of converting a multimedia resource from one form to another to match the device characteristics is called *content adaptation* or *transcoding*. It can be applied to transformations within media types (e.g., reducing the image size, transforming from high-fidelity JPEG to low-fidelity GIF format), across media types (e.g., speech to text, video item to image set) or to both of them. The transcoding process should preserve the semantic value of the original resource and produce a version which may be consumed by the client device. The use of XML and XSL, that allow the same content to be presented on a variety of Web-connected devices through structured XML documents (with the appropriate XSL style-sheets) facilitates the presentation of multiple versions of a Web site content. However, it does not eliminate the need for content transcoding.

The existing approaches to deploy Web content adaptation fall into three broad categories depending on the entity that performs the adaptation process [2, 18, 20]: *client-based*, *proxy-based*, and *server-based* adaptation.

In the *client-based* adaptation, the required transcoding is performed by the client devices. Carrying out the transcoding task on the client device allows to keep unchanged the current communication protocols; the client does not have to communicate any information about its characteristics. Client's computing power and connection bandwidth are continuously improving. However, since the content adaptation process generally involves computationally intensive operations [5, 13], the limited capabilities and low bandwidth of devices make content adaptation at the client very time consuming, if not impossible at all.

A better solution is to assume a thin client, thus not relying at all on the destination device to produce the required content version. In the *server-based* approach [14, 22], the functionalities of a traditional Web server are enhanced with content adaption. Therefore, transmission times are reduced by delivering an already adapted version of the object. The content transformation is typically generated off-line at content creation time, often involving a human designer to hand-tailor the content for some specific requirements. The multiple variants of the same resources are stored in the server and selected to match the client specification [22]. XSL can be also used to generate appropriate delivery markup from a common XML representation [14].

In the *proxy-based* approach [5, 11, 12, 13, 15, 21], a proxy server, located in an intermediate position in the delivery chain between the client device and the content server, can also analyze and transcode the desired content on-the-fly, before delivering the result to the client, as shown in Figure 1. The proxy server can cache the result of the transcoding, thus avoiding round-trips to the content server and subsequent transcoding operations when transcodable content can be served from the cache [12, 21]. Intermediate adaptation can shift computational load from content-providing servers and simplify their design, as it completely offloads the transcoding task to the intermediary infrastructure. A potential problem of the proxy-based transcoding is that traditional transcoding proxies operate autonomously, without being guided by the content server in the transcoding decisions; the server-directed transcoding proposal aims at solving this problem,

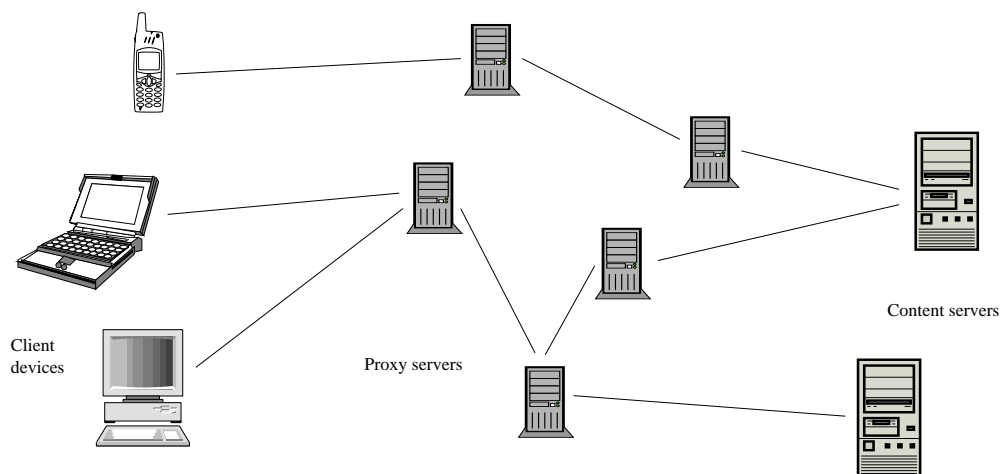


Figure 1: Proxy-based approach.

thus preserving the end-to-end content semantics [19].

In this paper, we present the architecture and implementation of Squid-based prototypes that extend the traditional caching systems to the heterogeneous client environment both at the single node level and at the cooperation level. The first extension enhances the traditional cache server, and transforms it into an active smart intermediary server that not only cache Web objects but also transcode them and store the result [21]. The second novel extension allows a single transcoding node to cooperate with other proxies in discovering, transcoding, and delivering Web content. Specifically, we investigate different architectures including active proxy servers organized in hierarchical and distributed topologies.

The rest of this paper is organized as follows. Section 2 provides a general description of the main functionalities that distinguish a traditional proxy server from a proxy that integrates transcoding operations and multi-version caching of Web resources. Sections 3 and 4 detail the modifications at the single node level and at the cooperation level, respectively. Section 5 describes the workload model that we have used to exercise the prototypes. Section 6 presents some experimental results that demonstrate the different functionalities of our prototypes. Section 7 discusses related work in the area of content adaptation carried out by intermediate proxy servers. Section 8 concludes the paper with some final remarks.

2 Functionalities of the caching and transcoding proxies

The proposed prototype of enhanced proxy server performs on-the-fly transcoding and caching of Web objects. In this section we describe the main functionalities which transform a proxy from an object repository along the delivery chain between the client and the Web server into an active element providing content adaptation.

The features of client devices vary widely in screen size and colors, processing power, storage, user interface, and software. Client's access links to Internet also range from wired networks, such as LAN, xDSL, ISDN, and telephone modems, to wireless networks such as GSM, CDPD, and future UMTS. When the proxy receives a client request, its first task is to identify the client type to decide which version of the object fits the client needs as much as possible. The client may include the resource data type it can consume as meta-informations into the HTTP request header, taking advantage of the potentialities of the protocols already in use. Recently, the WAP Forum

and the W3C have also proposed the compatible standards CC/PP and UAProf for describing the client capabilities [2]. The proxy can obtain the information regarding client capabilities by accessing a table storing the characteristics of the various client devices that may be served by that proxy. For example, a table entry for a particular client device can be created/stored when the device first registers with the edge server. Then, this entry can be transmitted by the edge server to the other proxies, for example by including it in the client request. Therefore, we can suppose that the proxy server is aware of the characteristics of all client devices. Hereafter, we will refer to the information describing the capabilities of the requesting client as *requester-specific capability information* (RCI).

The transcoding process implies the presence of multiple versions of the same object. An object which was already transcoded may be further transcoded to yield a lower quality object. In particular, each version may be transcoded from a subset of the higher quality versions. Different versions of the same object (and the allowed transcoding operations among them) can be represented through a *transcoding relation graph* [4]. In this paper, we consider a complete transcoding relation graph, assuming that each version can be obtained from any more detailed version.

The main modifications to the traditional behavior of a proxy server introduced by content adaptation are outlined below.

- Management of the Requester-specific Capability Information to determine the capabilities of the client device.
- Discovery phase: the lookup mechanism takes into account the existence of multiple versions of objects to find more detailed and transcodable versions that can be transformed to match the device characteristics.
- Content adaptation: each proxy provide transcoding capabilities to adapt resources to the client characteristics.
- Caching of multiple versions of objects: the caching mechanism has been modified to handle the presence of multiple versions of objects.

Our prototype is implemented as a modification of the freely available Squid 2.4 software [26]. Squid is an open source, high-performance proxy caching server. We chose Squid as a basic platform for its robustness and wide spread usage. Moreover, as Squid can support many cooperation mechanisms, such as Internet Cache Protocol [29] and Cache Digests [24], multi-version cooperative lookup can be implemented by modifying the existing code instead of writing it from the scratch.

Our caching and transcoding proxy can operate not only singularly but also in a cooperative way, following a hierarchical or a distributed scheme. In such a way we join the benefits of the cooperative discovery with transcoding functionality. Therefore, the proxy behavior has been modified both at the single node level and at the cooperation level. Modifications, which will be described in the following sections, have been brought so as to leave unchanged the normal proxy functionality if content adaptation is not required.

3 Interventions at the single proxy level

In this section we describe the main modifications to the basic original proxy in order to supply the multi-version caching and transcoding functionalities. Our prototypes present some substantial differences with respect to the original Squid architecture [28]. As regards the Squid code, the most important changes regard the *Client_side* and *Storage_manager* modules, which handle client request processing and caching operations, respectively.

3.1 Identification of the client device

The realized prototype aims at tailoring Web content to match the device characteristics as much as possible using all possible information contained in the client request. We choose not to implement a novel protocol for describing the client capabilities, but to use the HTTP protocol for communicating the resource data type that a client can consume.

The *client-side* Squid module has been modified in order to handle the RCI. To this purpose, we use the weak consistency of the entity tag (ETag), which is an identifier used to compare two or more versions of the same resource [10]. A numerical code, contained in this field, indicates the required version of a resource. By including this information into the HTTP request header, we can avoid altering the current communication mechanism between clients and proxies. Each client request contains the request modifier header *If-Match* with a weak ETag indicating the required version of a resource. When the request header is processed by the proxy, the proxy checks the presence of the field *If-Match* from which the weak ETag is extracted. A code value different from zero means that the required version is not the original one: in this case, the code is stored in the data structure `clientHttpRequest`, which keeps information regarding each client request. We choose this approach to simplify the testing of our prototype. However, it is possible to use any information included in the client request, such as another HTTP header or the client IP address, since the proxy identifies the requested version by analyzing the client HTTP request.

3.2 Identification of multiple versions

Squid identifies every Web object using the relative URL, through which it elaborates the hash key for finding the relative `StoreEntry`. This is the data structure that keeps track of the objects saved in the cache.

The process of transcoding implies the presence of multiple versions of the same resource. Therefore, the URL is no more sufficient to identify a resource, because it would be identical for any version. To manage multiple variants of the same object, we have decided to modify the URL by adding the string `"/X"` to it, where X represents the numerical identifier of one particular version. This modification has some impact on the *storage-manager* module, which is responsible for handling the cache content.

As shown in Figure 2, the proxy identifies the requested version by parsing the HTTP request header. If the *If-Match* field is present, the numerical code contained in the *Etag* is extracted and added to the original URL. A code value of zero, which identifies the original resource, leaves unchanged the URL.

3.3 Local multi-version lookup

Once the proxy has determined the version required by the client, it looks for the requested object in the local cache. If the required version is not present in the cache, the proxy searches for a more detailed and transcodable version that can be transformed to match the client characteristics. Hence, the lookup phase differs from the corresponding one of a traditional caching mechanism, because it is necessary a multi-version lookup which may generate one of the following three events.

- *Exact local hit*: the cache contains the exact version required by the client. A copy can be immediately delivered to the client.
- *Useful local hit*: the proxy cache contains a more detailed and transcodable version of the requested object that can be transformed to obtain a less detailed version that meets the client request.

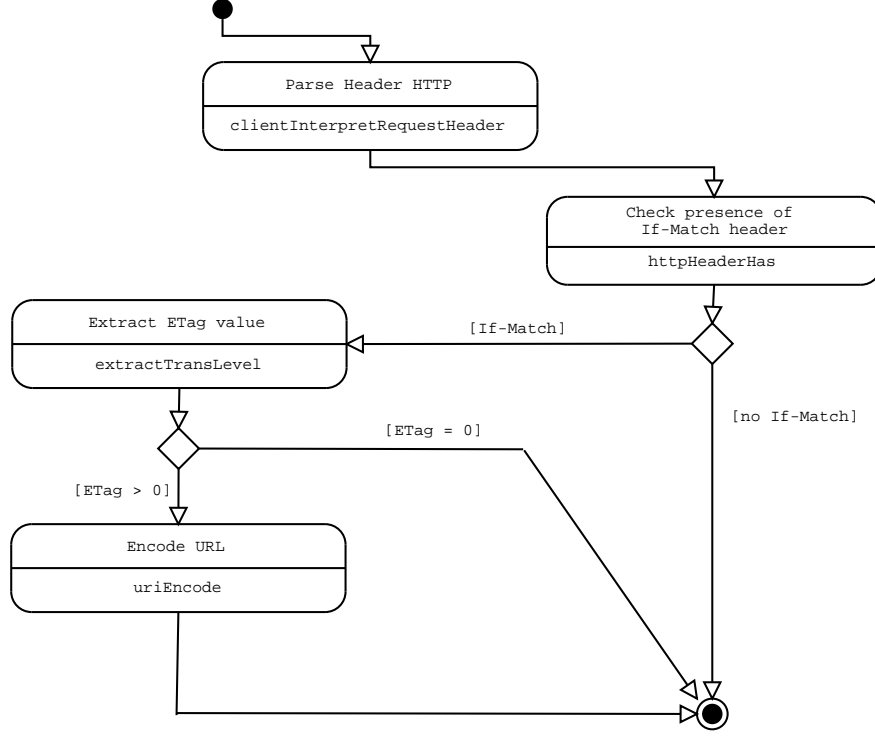


Figure 2: Parsing of a HTTP request.

- *Local miss*: the cache of the proxy does not contain any valid copy of the requested object. This means that no version of the object is found or a less detailed or untranscodable version is found, but it does not satisfy the RCI associated with the client request.

3.4 Caching of multiple versions

After the transcoding task, the proxy has to decide which version of the object is more useful to cache. To avoid repeating the onerous transcoding operations at the same proxy, every transcoded object may be cached. On the other hand, the object version on which the transcoding process has been applied, may be useful to satisfy a greater number of devices being more detailed.

There are three simple alternative caching policies that have been described in [4] to manage the caching of multi-version content. In the *demand-based* policy, the proxy caches the object version resulting from transcoding; in the *coverage-based* policy, the proxy caches only the transcodable version retrieved from the local cache of the transcoding proxy or from another peer; finally, in the *anticipatory* policy, the proxy caches both the transcoded and the retrieved versions.

Our prototype adopts the anticipatory policy. Hence, the Squid caching mechanism has been modified to cache both versions of the same resource, before and after transcoding. Figure 3 shows the sequence of operations performed when the proxy receives a HTTP reply from either a peer or the content server. Squid standard behavior, which carries out caching of the resource contextually to the delivery phase, has been modified in the following way. The transcodable version is both cached and written on a temporary file (whose function will be explained in Section 3.5) at the same time. After the transcoding process, a new **StoreEntry** structure is created to save the adapted version in the proxy cache. As shown in the bottom of Figure 3, caching of the transcoded version happens contextually to the delivery phase. As the main goal of this paper is

not to study new cache replacement policies, we adopt standard LRU as the replacement algorithm; more sophisticated policies have been discussed in [7, 25]. It is worth to observe that the caching mechanism of Squid is preserved; hence, every caching policy already implemented on Squid can be used in our prototype.

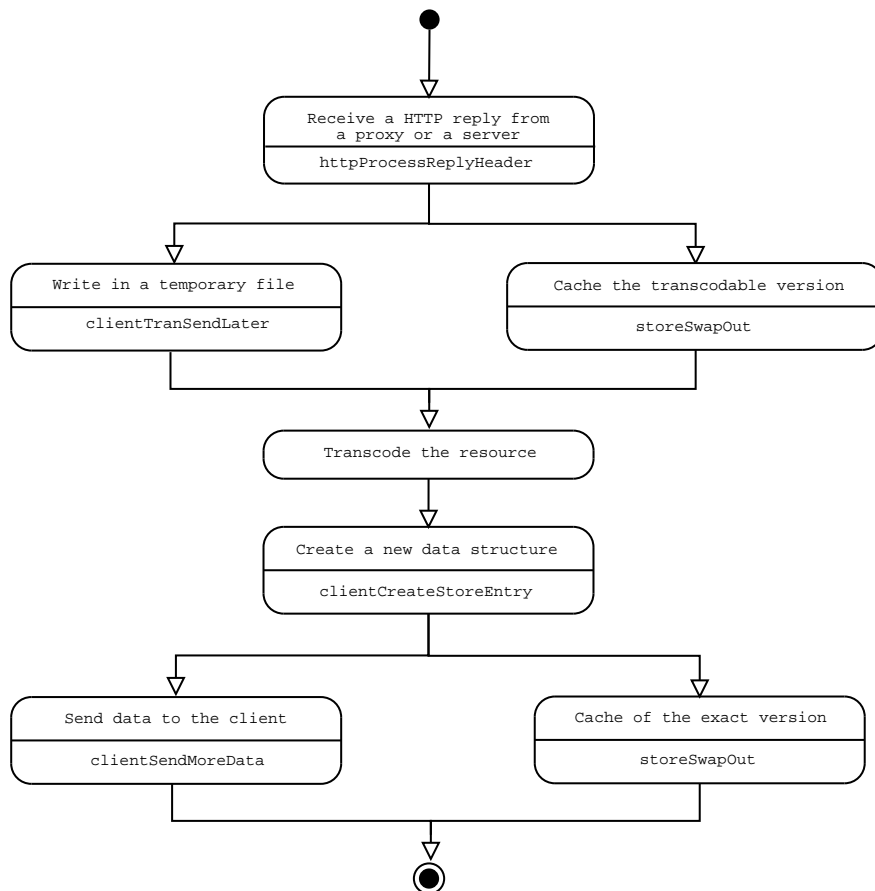


Figure 3: Caching in presence of transcoding.

3.5 The *Transcoder* extern process

To handle transcoding operations, our prototype uses an external process, called *transcoder*, which is created when necessary. It exchanges information with Squid through a pipe. The motivation for this choice is that, unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process. Since transcoding operations are generally longer than any other typical Squid task, it would not be efficient to include them in the main() program. This solution allows multiple transcoding operations to be performed in parallel. Moreover, the use of multiple instances of the transcoder process can increase the performance in SMP systems (our experience is on bi-processor machines). For example, we found that on a bi-processor SMP system this approach reduces the 90-percentile of the transcoding time up to 110% with respect to a uniprocessor system.

Squid and the transcoder process use also temporary files to exchange data. At first, the transcodable object is written on a temporary file, then an extern process is created and a communication pipe is opened through the Squid InterProcessCommunication functions. Squid uses other extern processes, like pinger or unlinkd, so that our solution is completely consistent with the Squid

code. At the end of the adaptation task, the transcoder process through the pipe notifies Squid that the adapted resource is available on the temporary file to be sent to the client. Figure 4 shows the communication mechanism between processes: the continuous lines represent exchanges of data, the dashed ones the execution flow.

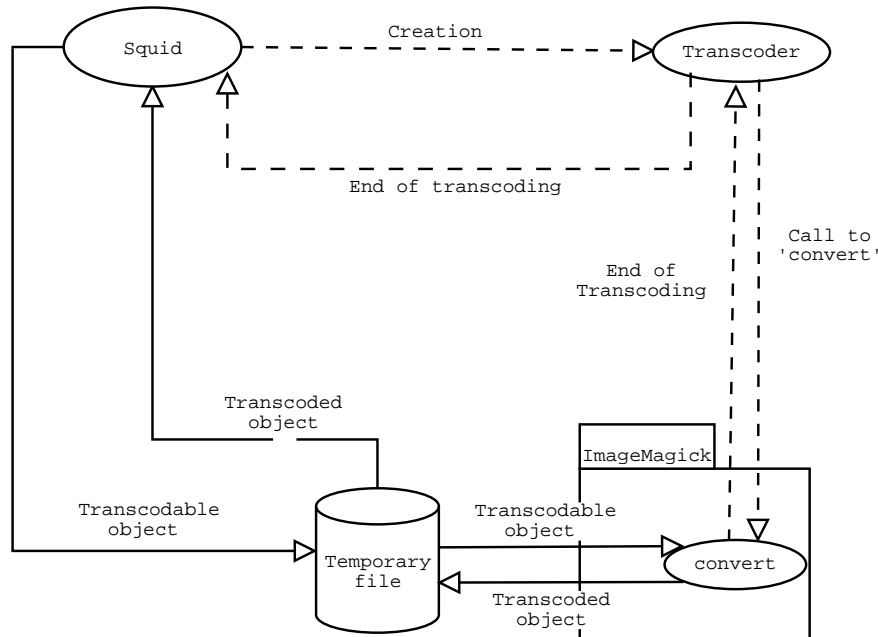


Figure 4: Interaction among processes.

In this version the transcoder process uses the freely available ImageMagick library version 5.4.5 [16] to adapt the resources to the client specifications. This library allows to manipulate many image formats, supplying numerous transcoding algorithms. However, other transcoders can be used with simple software integration. Indeed, the transcoding module is implemented as an external process with the specific purpose of maximizing flexibility and making the prototype independent of the transcoding functionality. The manipulation methods for adapting media objects are a function of their current type (HTML, image, video, and audio) and the characteristics of the client device.

4 Interventions for cooperation

We are also interested in considering and implementing a distributed system for cooperative discovery, transcoding, and delivery of Web content. Cooperation among proxies can be established along a vertical direction, namely hierarchical Web caching, descending from the Harvest project [8], or along an horizontal direction, namely flat or distributed Web caching [23]. In a *hierarchical* architecture, the proxies are organized in a hierarchy of levels of nodes, where only the bottom level proxies, called *edge* proxies, serve client requests directly. In a *flat* architecture, all the proxies are placed on the same level and all the nodes are edge proxies.

In this section we describe the main Squid modifications to integrate the transcoding process with a cooperation scheme.

4.1 Cooperative transcoding in hierarchical topologies

In a *hierarchical* caching architecture, the edge proxies forward the client request up to the hierarchy until an hit occurs. If no hit occurs at any level, the root proxy retrieves the original version of the requested object from the content server.

As transcoding may involve computationally expensive operations, there is a great risk of poor performance due to overload in the upper level nodes. For this reason, we propose a cooperation algorithm for discovery where transcoding task is performed only by edge proxies, while upper levels nodes act as traditional cache servers.

The behavior of an edge proxy when it receives an HTTP request can be described as follows:

- In the case of local exact hit, the edge proxy sends the resource to the client.
- The edge proxy that experiences a useful hit performs the transcoding task locally before sending the resource to the client.
- In the case of local miss, the edge proxy requires the original version of the requested object to the parent proxy and transcodes it locally.

When a resource passes down the hierarchy, a copy is left at each intermediate cache, until it reaches the client.

The hierarchical cooperation does not require further modifications to the Squid code in addition to the ones described in Section 3.

4.2 Cooperative transcoding in flat topologies

In a *flat* architecture all the proxies are *peer*; many different protocols may be used to carry out the cooperation among proxies. Apart from the used lookup protocol, the existence of multiple versions of the same resource implies modifications to the cooperative discovery. Indeed, the lookup phase may provide one of the following results.

- *Remote exact hit*: the exact version of the required object is found in a peer cache.
- *Remote useful hit*: a transcodable and more detailed version of the required object is found in a peer cache.
- *Remote miss*: no version of the requested object is found in peer caches; in this case a *global miss* occurs and the original version is retrieved from the content server and transcoded, if necessary.

Figure 5 shows the behavior of a peer when it receives an HTTP request. The proxy searches for exact and useful hits simultaneously, but exact hits are preferred to the useful ones as they allow to avoid transcoding operations.

In the following two sections we consider two classical cooperation policies for distributed systems, namely *query-based* and *directory-based* cooperation.

4.2.1 Directory-based protocols

In directory-based protocols, as Cache Digests [24], each proxy periodically informs the others about changes in its set of cached documents. When a local miss occurs, the proxy checks the summary of the documents cached at its peers to discover potential remote exact or useful hits. For the experiments, we choose Cache Digests as a representative of the directory-based architectures,

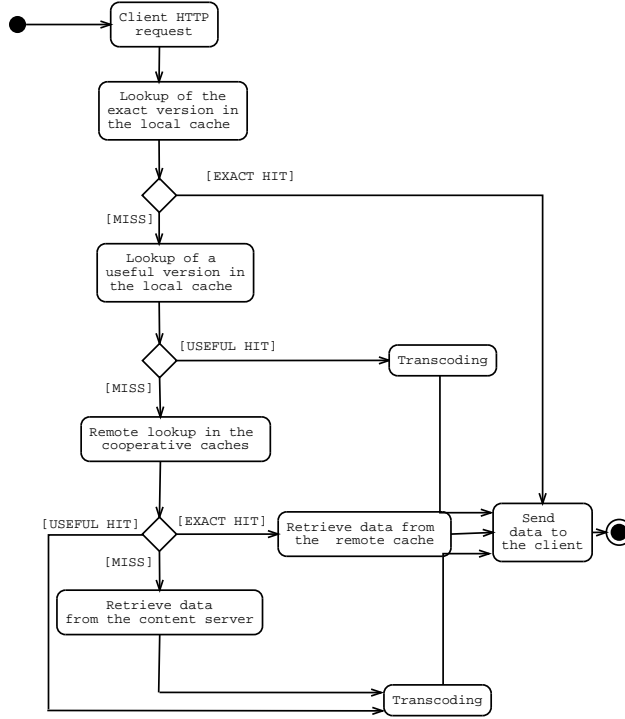


Figure 5: Multi-version lookup in a flat architecture.

because of its popularity and its implementation in the Squid software. By adding the resource version identifier to the URL, we include in our prototype the support for multiple versions into the summary-based lookup process in a transparent way. Therefore, the basic mechanism of Cache Digests cooperation is preserved. However, the lookup process becomes more expensive because a lookup for every possible useful version must be performed.

4.2.2 Query-based protocols

Query-based cooperation activates message exchanges only at lookup time: when a proxy experiences a local miss, it sends a query message to every peer in order to locate the exact or a useful version of the requested object. The most important query-based protocol is ICP, defined in [29], and used in NetCache [9] and Squid [28]. For this reason we used ICP as the basic protocol for query-based cooperation.

More essential modifications are necessary to add the support for multiple-version lookup into the Squid version of ICP (defined in the module *ICPv2*). ICP does not supply support for HTTP semantics: an ICP query contains only the URL of the resource, but none of HTTP headers. Hence, we included the encoded URL, added of the resource version identifier, in the payload of the ICP message to indicate which version of the object is required. To distinguish the multiple-version lookup from the regular lookup of the ICP protocol, we use a new flag, namely *ICP_MULT_VERSION*. We also introduce a new response code to indicate a useful hit instead of an exact hit. In the *ICP_OP_USEFUL_HIT* response, the URL field of the ICP message contains the transcodable version identifier into the URL.

In particular, the enhanced ICP lookup is activated by setting the flag *ICP_MULT_VERSION* in the request message. Then, we have three possible response codes: *ICP_OP_HIT* in the case of exact hit in the peer, *ICP_OP_USEFUL_HIT* for useful hits, and *ICP_OP_MISS* when neither exact nor useful

hits are detected.

Figure 6 shows the ICP reply mechanism of a transcoding proxy upon receiving an ICP_QUERY message with the ICP_MULT_VERSION flag.

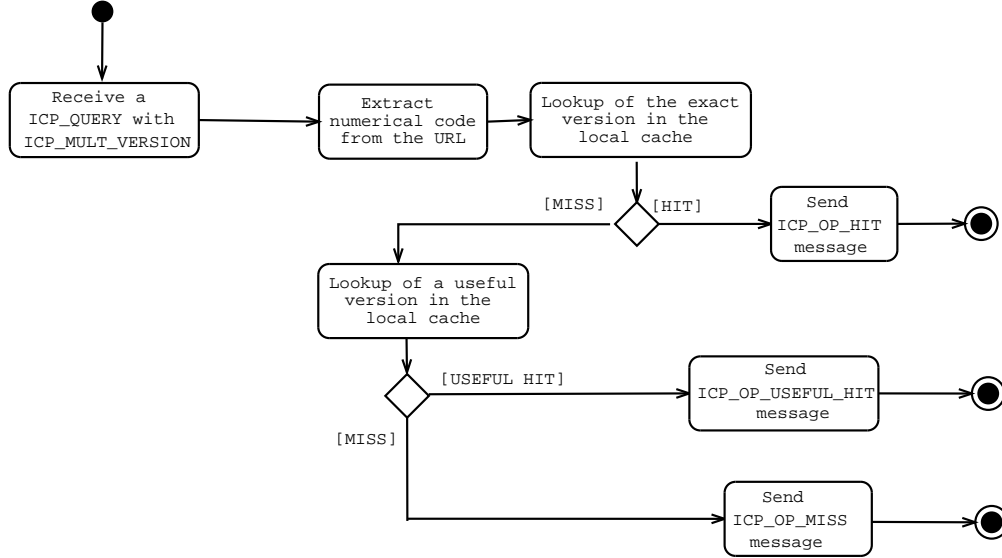


Figure 6: Receiving a ICP query with ICP_MULT_VERSION flag set.

If different useful versions are detected in the peer caches, the proxy tries to retrieve the least detailed version to reduce the transmission time and the cost of content adaptation.

5 Workload model

In this section, we describe the client workload models used to exercise the proxy prototypes. We consider a classification of the client devices on the basis of their capabilities of displaying different objects and connecting to the assigned edge proxy [4, 7]. We have identified the following six classes of devices:

- **HighPC**: a high-end workstation/PC with a network link ranging from Ethernet to DSL; this device can consume every object in its original form.
- **MedPC**: a midrange PC or a laptop connected through a fast/medium wire-connected modem; to reduce network-resource usage, the image size and the quality factor (for JPEG images) are reduced.
- **TVBrowser**: a set-top box that can turn a TV into a Web browser; as the maximum resolution is limited by the TV screen, the image size should be trimmed down not to waste network and CPU appliance resources.
- **HPC**: a hand-held PC connected through a modem; it can display color images with different resolutions, but the screen is typically small.
- **PDA**: a personal digital assistant using a wireless connection; it is not capable of displaying colorful and large images.

- **Phone:** a cellular phone using a wireless GSM connection; the screen is small and can only display black and white (1 bit) graphics.

Table 1 shows the bandwidth of the network link connecting the device to the edge proxy and the different device capabilities for handling images.

Table 1: Client device types.

Device	Bandwidth	Color	Maximum resolution (pixels)	Percentage of diffusion
HighPC	10 Mbps	24-bit color	original	25%
MedPC	64 Kbps	24-bit color	800×600	25%
TVBrowser	56 Kbps	8-bit color	640×480	12.5%
HPC	56 Kbps	8-bit color	120×120	12.5%
PDA	28.8 Kbps	greyscale	120×120	12.5%
Phone	9.6 Kbps	b&w	120×120	12.5%

To characterize the workload, it is worth making prediction about the future diffusion of different Web-connected devices. In this paper, we assume that traditional devices, such as PCs and laptops, are still more popular than other client devices. The percentages of diffusion of client devices are reported in the last column of Table 1. It is difficult to predict the trend of diffusion of future Web-connected devices. However experiments with percentages within 10% of those reported here did not affect the main conclusions of this paper.

In this paper, we consider that transcoding operations are applied only to image objects (both GIF and JPEG formats), as more than 70% of the files requested in the Web still belongs to this class [1, 6].

It is worth to determine which characteristics of Web images may be critical for displaying on limited devices, in order to choose efficient and appropriate transcoding algorithms. In a recent study, Chandra et al. [6] have analyzed the nature of typical images in Web workload and their transcoding characteristics. They found that 74,81% were GIF and 24,41% were JPEG images. GIF and JPEG images represent almost all the image bytes transferred, but the percentages are evenly distributed between the two formats. With regard to the image size, most of the GIF images accessed on the Web (about 80%) are smaller than 6KB, while about 40% of the JPEG images are larger than 6KB. Transcoding can be performed along a number of different axes. To obtain a reduction in the file size, the transcoding algorithms seem to perform better if operating on spatial geometry, color depth, quality, animation, and MIME subtype.

For our experiments, we use a workload model based on proxy traces collected on August 2002 and belonging to the nodes of the IRCache infrastructure [17]. We downloaded the resources from their content servers and placed them on Web servers. We performed some characterization on the images of such workload such as file size, JPEG quality factor, and number of colors of GIF images, and found that they are very close to the results reported in [6]. Hence, we can assume that our workload captures a realistic scenario of present Web requests. The measured costs of transcoding operations required by the workload on machines used for our experiments gave the following results: 0.12 and 0.22 seconds for the average and the 90-percentile service time, respectively.

6 Experimental results

The main goal of the experimental results described in this section is twofold. First, we aim to evaluate the effectiveness of the transcoding process in reducing object size and, consequently,

transmission times on the link between the client and its edge proxy. Second, we focus on cooperative resource discovery, analyzing the performance of different cooperative proxy architectures organized in hierarchical and distributed topologies, and comparing them to the case of a system of non-cooperative proxies.

As our main target is to enable heterogeneous devices to access Web content, the servers transcode the object to best fit the client capabilities, while we do not explore object compression to reduce transmission time (e.g., [13]). We also consider only complete transcoding relation graphs, where each version can be created starting from any higher quality version [4]. In our experiments we set up a system of 16 servers. Each node is a PC with a Pentium III 933 MHz CPU and 256 MB of RAM running Linux. The servers are equipped with the prototypes described in this paper and configured to cooperate through different architectures and discovery protocols.

Figures 7 and 8 show the size reduction obtained by performing transcoding on GIF and JPEG images, respectively. The metric used is the cumulative distribution of the *stretching factor*, defined as the ratio of the size of a transcoded image over the size of its original version.

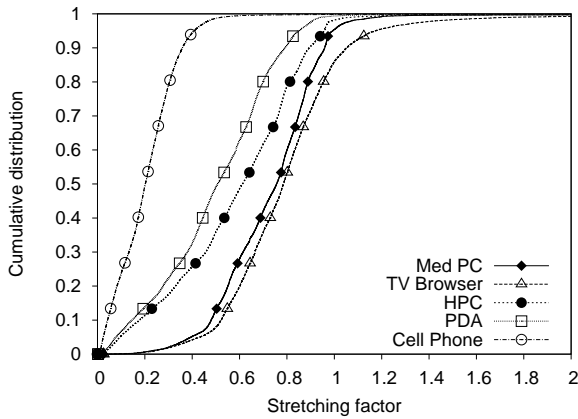


Figure 7: Size reduction of JPEG images.

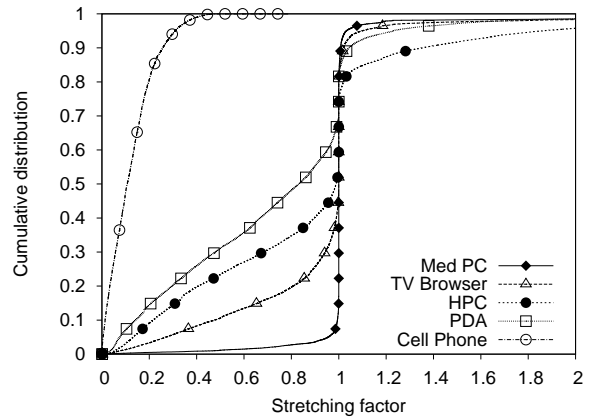


Figure 8: Size reduction of GIF images.

For JPEG images adapted to TV browser devices, we have obtained a stretching factor value lower than 1% for 85% of images. For JPEG images adapted to other devices this percentage reaches 95%, as shown in Figure 7. The adaptation for cellular phones reduces all the JPEG images at least of 50%: for this class of devices, we convert JPEG images to the GIF format because it is more suitable for black and white images. In this case, it is necessary to dynamically change the MIME subtype included in the HTTP reply header. The size of GIF images is dropped by a little percentage (5%) after adaptation to Med PC, while we can see better results for the other classes of devices (40% for TV Browser, 50% for HPC, 70% for PDA and over 90% for Cellphone), as shown in Figure 8.

The size reduction obtained by performing transcoding can reduce the transmission time of the adapted resources on the link between proxy and client. Tables 2 and 3 show the median value of transmission times for the different client types, depending on the image format. Table 2 refers to the transfer of original resources, while Table 3 reports results about images that require transcoding operations.

It is interesting to note that the median value of transmission times is comparable for all the client types, except for High PC, as shown in Table 3. In the case of GIF images, the transfer time for devices with slow connections, as Cell Phones, is even lower than that of better connected devices as MedPC.

	HighPC	MedPC	TVBrowser	HPC	PDA	Phone
GIF	0.002	0.369	0.421	0.421	0.819	2.458
JPEG	0.005	0.944	1.079	1.079	2.097	6.292
GIF & JPEG	0.003	0.540	0.617	0.617	1.200	3.600

Table 2: Median of transmission times for original resources [sec].

	HighPC	MedPC	TVBrowser	HPC	PDA	Phone
GIF	0.002	0.378	0.354	0.2	0.261	0.241
JPEG	0.005	0.667	0.897	0.393	0.672	0.675
GIF & JPEG	0.003	0.462	0.508	0.256	0.383	0.367

Table 3: Median of transmission times for transcoded resources [sec].

We carried out a second set of experiments to compare the performance of different cooperative architectures and to understand if cooperation among proxies can improve the system response time with respect to the case of no cooperation. We set up a scenario where all servers are well connected among them and with the clients. The content servers are placed in a remote location, connected through a geographic link with 14 hops in between, a mean round-trip time of 60 ms, and a maximum bandwidth of 2Mb/sec. We verified that in this scenario the network path to the content servers (reached in case of global miss) was one of the possible system bottleneck. Hence, the global cache hit rate may impact on the response time.

We consider a **Hierarchical** architecture, a flat peer-to-peer architecture using a query-based discovery protocol (**ICP**), and a flat architecture adopting a summary-based protocol (**Cache Digests**). The configuration for ICP and Cache Digests is based on a flat peer-to-peer cooperation scheme, where all proxies have sibling relationships among them. The hierarchical architecture is configured on the basis of a three-level hierarchy with 12 edge proxies, 3 intermediate proxies (with a nodal out-degree of 4), and one root proxy. The client requests reach only the edge proxies. For performance comparison, we consider also the non-cooperative scheme (**No_Coop**), in which proxy servers do not cooperate with the other nodes. In this scheme, exact and useful hits can only be local, while local misses cause a request of the resource to the content server.

Table 4 shows the achieved cache hit rates: the first four columns report the local and remote hit rates (exact and useful), while the last column summarizes the global hit rate. For the hierarchical scheme, we do not report the remote useful hits, because the requests to the parent nodes refer only to the original version of the resources.

Table 4: Cache hit rates.

	Local exact HR	Local useful HR	Remote exact HR	Remote useful HR	Global HR
No_Coop	28.4%	7.6%	n/a	n/a	36.0%
Hierarchical	10.2%	8.2%	19.6%	n/a	38.2%
Cache Digests	21.2%	11.9%	11.5%	11.5%	56.4%
ICP	19.4%	16.9%	13.8%	19.3%	69.5%

From the last column of Table 4 we can observe that there are some significant differences in the global hit rates, depending on the used cooperation mechanism. Flat architectures offer the best results, in particular ICP achieves higher values about remote hit rates. Indeed, when the cooperation occurs among many nodes, Cache Digests becomes imprecise (the accuracy of the

exchanged cache digests decrease) and its remote hit rates diminish. We have conducted a brief performance comparison of flat cooperative architectures between ICP and Cache Digests in [3]. The hierarchical model shows hit rates lower than those of other architectures. As transcoding is performed only by edge servers, when a local miss occurs at these nodes both the original and the transcoded version of the requested resource are stored in the cache. Due to the limitation of the cache size, this can notably reduce hit rates.

7 Related work

In this section we review some recent proposals regarding content adaptation when the latter is carried out by intermediate proxy servers enhanced with functionalities to transcode the content on-the-fly and possibly cache it. Most research efforts have focused on handling the variations in client bandwidth and display capabilities [5, 11, 12, 13, 15, 30] without focusing on caching aspects, while considering object compression techniques [13, 30]. In these proposals, the edge proxy server that directly connects to the clients typically reduces the resource size, thus reducing bandwidth consumption, apart from providing an object version that the client device can consume.

A limited number of recent proposals have also exploited techniques to combine both transcoding and caching to reduce the resource usage at the proxy [7, 25, 27]. Singh *et al.* [25] have examined how to improve the effectiveness of transcoding by integrating it with caching mechanisms and taking into account the transcoding utility of any cached object to decide about transcoding. The image transcoding is carried out using the convert utility provided by the ImageMagick library, as done in our prototype. With respect to the proxy architecture discussed in [25], which is based on the Java-based proxy Rabbit, we propose to enhance with transcoding and caching functionalities the Squid proxy server, which is characterized by robustness and wide spread usage. Moreover, we introduce a new challenge in proxy caching by enhancing some cooperation mechanisms provided by Squid in order to provide multi-version resource discovery.

A Squid-based transcoding proxy has been first presented by Maheshwari *et al.* [21], which have also discuss how to match client requests to cached transcoded objects. As already pointed out, in this paper we present Squid modifications not only at the single proxy level but also at the cooperation level.

More sophisticated cache replacement policies designed for transcoding proxies have been proposed in [7, 27]. Chang *et al.* [27] have investigated various caching policies in the context of HTML and image files and have compared their performance through simulation. Tang *et al.* [27] have focused on adaptive caching policies for video objects, also proposing a technique to partition a video object into sections and handle them individually for proxy caching. The performance evaluation of the proposed caching algorithms is carried out through simulation. In future work we plan to investigate more sophisticated caching policies for multi-version content, besides the simple anticipatory one (that caches both the original and the transcoded objects) which is currently implemented in our prototype. It is worth to point out that our Squid-based prototype provides an extensible framework that allows us to integrate new caching policies and evaluate their performance in a real Web environment.

The main motivation that leaded us to study cooperative architectures of caching and transcoding proxies is the limited scalability to which the existing proxy-based approach may be subject, since transcoding operations can be computationally expensive [21]. Fox *et al.* [12] address this scalability issue by proposing a cluster of locally distributed proxies. This approach may solve the CPU-resource constraint, but it tends to move the system bottleneck from the proxy CPU to the interconnection of the cluster. On the other hand, in our proposal the system of cooperative

proxies can be distributed over a wide area network thus avoiding network bottlenecks. Finally, in [4] the authors have obtained some preliminary results through simulation that demonstrated the importance of distributing the computational load of transcoding in a proxy hierarchy to improve user-perceived response time.

8 Conclusions

In this paper, we have proposed operative Squid-based prototypes that adapt Web images by operating on geometry, color depth, quality, and MIME subtype depending on the client device capabilities. We have modified most of the traditional phases of the standard Squid to support the transcoding process. In particular, we have introduced a multi-version discovery process to manage the existence of different versions of the same resource. We have used an external transcoder process to maximize flexibility and make the prototype independent of the particular transcoding algorithms applied to adapt Web content. We have also investigate how to enable cooperation for proxy servers organized in hierarchical and distributed topologies, with the objective of demonstrating that cooperative schemes perform better when compared to the case of no collaboration among the proxies.

Integrating the transcoding process with traditional cooperative caching raises many important and unexplored research issues. Interesting issues that require further investigation are the use of different transformation methods, novel cache replacement algorithms that take into account multiple versions of a resource, and the integration of server and proxy operations. It is also worth to emphasize that the computational cost of transcoding may be notably reduced by using cooperation schemes that involve some other proxies to perform the possibly expensive adaptation task.

References

- [1] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a Web proxy in a cable modem environment. *ACM Performance Evaluation Rev.*, 27(2):25–36, Aug. 1999.
- [2] M. Butler, F. Giannetti, R. Gimson, and T. Wiley. Device independence and the Web. *IEEE Internet Computing*, 6(5):81–86, Sept./Oct. 2002.
- [3] V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu. A distributed architecture of edge proxy servers for cooperative transcoding. In *Proc. of 3rd IEEE Workshop on Internet Applications*, June 2003.
- [4] V. Cardellini, P. S. Yu, and Y. W. Huang. Collaborative proxy system for distributed Web content transcoding. In *Proc. of 9th ACM Int'l Conf. on Information and Knowledge Management*, pages 520–527, Nov. 2000.
- [5] C. S. Chandra, S. Ellis and A. Vahdat. Application-level differentiated multimedia Web services using quality aware transcoding. *IEEE J. on Selected Areas in Communication*, 18(12):2544–2465, Dec. 2000.
- [6] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat. Transcoding characteristics of Web images. In *Proc. of Multimedia Computing and Networking Conf.*, Jan. 2001.
- [7] C.-Y. Chang and M.-S. Chen. Exploring aggregate effect with weighed transcoding graphs for efficient cache replacement in transcoding proxies. In *Proc. of IEEE 18th Int'l Conf. on Data Eng.*, Feb. 2002.
- [8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A hierarchical Internet object cache. In *Proc. of USENIX 1996*, pages 153–163, Jan. 1996.
- [9] P. Danzig. NetCache architecture and deployment. *Computer Networks*, 30(22-23):2081–2091, 1998.

- [10] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, June 1999.
- [11] R. Floyd and B. Housel. Mobile web access using eNetwork Web Express. *IEEE Personal Communications*, 5(5):47–52, Oct. 1998.
- [12] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proc. of 16th ACM Symp. on Operating Systems Principles*, pages 78–91, Oct. 1997.
- [13] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. Dynamic adaptation in an image transcoding proxy for mobile Web browsing. *IEEE Personal Communications*, 5(6):8–17, Dec. 1998.
- [14] IBM. IBM WebSphere Transcoding Publisher, 2003. http://www.ibm.com/software/pervasive/products/mobile_sols/transcoding_%publisher.shtml.
- [15] S. Ihde, P. P. Maglio, J. Meyer, and R. Barrett. Intermediary-based transcoding framework. *IBM System Journal*, 40(1):179–192, 2001.
- [16] ImageMagick, 2003. <http://www.imagemagick.org/>.
- [17] IRCache project, 2003. <http://www.ircache.net>.
- [18] A. Joshi. On proxy agents, mobility, and Web access. *Mobile Networks and Applications*, 5(4):233–241, 2000.
- [19] B. Knutsson, H. Lu, and J. Mogul. Architectures and pragmatics of server-directed transcoding. In *Proc. of 7th Int’l Workshop on Web Content Caching and Distribution*, Aug. 2002.
- [20] W. Y. Lum and F. C. M. Lau. On balancing between transcoding overhead and spatial consumption in content adaptation. In *Proc. of ACM Mobicom 2002*, pages 239–250, Sept. 2002.
- [21] A. Maheshwari, A. Sharma, K. Ramamritham, and P. Shenoy. TransSquid: Transcoding and caching proxy for heterogeneous e-commerce environments. In *Proc. of 12th IEEE Int’l Workshop on Research Issues in Data Engineering*, pages 50–59, Feb. 2002.
- [22] R. Mohan, J. R. Smith, and C.-S. Li. Adapting multimedia Internet content for universal access. *IEEE Trans. on Multimedia*, 1(1):104–114, Mar. 1999.
- [23] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002.
- [24] A. Rousskov and D. Wessels. Cache Digests. *Computer Networks*, 30(22-23):2155–2168, 1998.
- [25] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. PTC: Proxies that transcode and cache in heterogeneous Web client environments. In *Proc. of 3rd Int’l Conf. on Web Information Systems Engineering*, Dec. 2002.
- [26] Squid Internet Object Cache. <http://www.squid-cache.org>.
- [27] X. Tang, F. Zhang, and S. T. Chanson. Streaming media caching algorithms for transcoding proxies. In *Proc. of Int’l Conf. on Parallel Processing*, pages 287–295, Aug. 2002.
- [28] D. Wessels. *Squid Programmers Guide*, 2002.
- [29] D. Wessels and K. Claffy. *Internet Cache Protocol (ICP), version 2*. RFC 2186, Sept. 1997.
- [30] B. Zenel. A general purpose proxy filtering mechanism applied to the mobile environment. *Wireless Networks*, 5(5):391–409, 1999.