Design and Implementation of a Soft Caching Proxy

Jussi Kangasharju Young Gap Kwon Antonio Ortega
{kangasha,younggap,ortega}@sipi.usc.edu

Integrated Media Systems Center
Department of EE-Systems
University of Southern California
3740 McClintock Ave. #400
Los Angeles, CA 90089

Abstract

In this paper we address a set of modifications to classical proxy caching algorithms which allow the implementation of a soft caching proxy system. Changes to replacement algorithms are detailed and image size and recoding issues are discussed. We also present our working soft caching testbed based on the Squid proxy, detail the modifications we have made and present the experiences obtained.

Keywords: web caching, image recoding

1 Introduction

The explosive growth in Internet traffic has made it critical to look for ways of accommodating the increasing number of users while preventing excessive delays, congestion and widespread blackouts. This has led to a great deal of effort being devoted to studying web caching techniques [2]. While caching is useful both at the server (for example some pages might be kept in RAM memory) and the client (where recently accessed files are saved to disk), we concentrate here on proxy based caching. In this environment clients can designate a host to serve as a proxy for all or some of their requests. The proxy acts as a cache by temporarily storing on local disks, or memory, objects which were requested by the clients.

Current design efforts are based on the assumption that object integrity has to be preserved, i.e. objects cannot be modified. Thus an object is either present or not in the cache, but cannot be available in part. Here we propose that caching proxies should be able to perform recoding of the images in the cache so that lower resolution versions of the images can be stored and made available to the clients. We call this strategy *Soft Caching* [8, 17], since we now allow a lower resolution version of an image object to be stored; a soft decision is made (how many bits to use for a certain image) instead of a hard one (is the image left in the cache or not).

In our framework, specific caching strategies are derived for images and other media objects, for which preserving the object integrity is not completely necessary. While incomplete delivery of text data may render it useless, for images it may be sufficient to provide the end user with a lower resolution version of the image stored at the remote server, especially if this can be done in significantly less time than it would take to download the full resolution image. This is one of the reasons for the continuing popularity of progressive image formats such as Progressive JPEG [10] nowadays, or pyramids [1] and wavelets [12] in the near future.

When access occurs over a fast link progressive transmission may not offer significant advantages, but over a slow link a progressive format allows a usable, albeit reduced resolution, picture to be available even if the transfer is terminated early. This allows users to stop the transfer when a sufficient quality image has been downloaded. A study of such a multi-resolution imaging system under simple assumptions can be found in [9, 7] and motivates the advantages in terms of average access delay of using a progressive transmission.

If the proxy has a lower resolution version of the requested image but the image quality is too low, the user can always request the original image from the server by using the standard method of the browser for by-passing caches (e.g. Shift-Reload in Netscape). Currently this means transferring all of the original image from the server to the client, so the price to pay for the better resolution is an increased download time.

As progressive image formats and transmission methods that make it possible to request partial objects, such as the range-request method of HTTP/1.1 [5], become more widespread, the reloading of the whole image can easily be avoided. The benefits on the user-proxy link are still the same and even when the user wants to see all of the image, only the bits that are needed to complete the image need to be requested from the origin server, thus reducing the load on the backbone.

Real-time distillation [3, 4] has been proposed to allow proxies to extract (distill) a low resolution image to serve it to slow clients (e.g. clients connected to the network via dial-up). Recently, a commercial implementation of the real-time distillation framework [14] has come to the market.

While the distillation approach is similar in philosophy, soft caching addresses a more general set of problems. The recoding operation is similar to distillation which gives a faster download time on a slow client-to-proxy link thanks to the smaller size of objects. However, in our approach, recoding in the cache is done both to satisfy the lower bandwidth requirements of the clients and to allow more efficient memory usage. By using progressive formats we can serve a range of image qualities to the clients, while the number of levels of resolution to be kept in the cache will depend on image popularity and the available memory.

This paper is organized as follows. Section 2 addresses the modifications to existing systems that are required by soft caching. In section 3 we present our soft caching testbed. Section 4 sets directions for future work. Finally, section 5 concludes the paper.

2 Soft Caching Algorithms

While most of the general principles of proxy caching are directly applicable to soft caching, an efficient soft caching system must also consider the effects of image recoding in its algorithms.

Given the advantages (minimization of average access time) of progressive transmission [9, 7], our goal is to find simple, yet efficient ways to improve existing caching algorithms to make them benefit from the advantages of soft caching.

We will first discuss issues related to recoding and image size and then we will present the modifications to existing cache replacement algorithms.

2.1 Recoding Levels

An important factor in a soft caching system is the number of recoding levels or the number of different representation levels an object can have in the proxy.

Having many recoding levels reduces the potential benefits from soft caching since an image can remain in the cache for a long time even though it will not be used. Having fewer recoding levels would get rid of the unneeded images quicker but could also reduce the quality of the useful images faster. Also, the smaller the number of levels, the less we use CPU-time to recode the image during its stay in the cache.

To get more insight into the number of levels needed, we took 800 unrecoded JPEG-images and recoded them through all the 10 levels used by the Independent JPEG Group's library [6]. The images were mostly small photographic images, resolution about 300x200 pixels and average size around 12 KB. Some of the images were much larger (1500x1000, 400 KB or even larger) and a significant part of them were non-photographic graphical images such as buttons and toolbars. Table 1 presents the savings obtained in going from a higher recoding level to a lower one.

From the results in table 1 we clearly see that having 10 recoding levels is an overkill. Going from level 8 to level 7 and on to level 6 does not offer much new savings in file size and the degradation in visual quality is almost non-perceptible. So a user who finds level 8 acceptable will also find level 6 acceptable with a very high probability. A similar argument holds for levels 4, 3 and 2.

For most photographic images the level where coding artifacts start to appear is level 5 or 6. Non-photographic images usually degrade faster, quality becoming objectionable around level 7 and degrading completely at level 5. This

Level	Percentage from previous level	Percentage from original image
Full	100 %	100 %
9	64.7 %	64.7 %
8	92.6 %	59.9 %
7	92.7 %	55.5 %
6	95.5 %	53.0 %
5	71.0 %	37.6 %
4	71.3 %	26.8 %
3	86.0 %	23.1 %
2	85.4 %	19.7 %
1	55.2 %	10.9 %

Table 1: JPEG-image size at different recoding levels

is because JPEG has not been designed to compress such images. Figure 1 shows the Lena-image at different recoding levels.

Full Level 6

Level 3 Level 1

Figure 1: Lena-image at different recoding levels

These findings indicate that a good number of recoding levels would be 4 or 5. These would be situated at the full level and approximately at levels 8, 6, 3 and possibly level 1. Although at the first level no details of the image can be seen, it still gives an overall view of the contents of the image. In some situations, such as browsing an image database, this might be enough information to make the decision whether to load more or not.

2.2 Image Size

Since a significant part of the images consists of navigation buttons and other such non-photographic images which do not take well to recoding, something should be done about them.

A simple way is to look only at the image size and make decisions based on that. Recoding an image of size 2 KB at full resolution to level 6 would save us about 1 KB of disk space but would also use CPU-power doing that. It is therefore not efficient to recode small images for two reasons:

1. They are likely to be non-photographical, i.e. recoding them is not useful

2. The savings in transmission time and disk space are negligible compared to the CPU-time used.

Where to set the threshold of small image? Our sample of 800 images had an average size of 12 KB and we have observed that most non-photographic images tend to be under 5 KB in size. Also, photographic images that require a small storage are likely to be small on screen, and therefore are useful with high quality (e.g. thumbnail images). Therefore a threshold of around 10 – 15 KB would be a good choice. It would not waste time and effort on small images but would still give all the benefits for larger images.

2.3 Replacement Strategy

Cache replacement strategies in the context of WWW proxies have been extensively studied [18, 11, 19]. The popular Least Recently Used (LRU) [16] algorithm has been found to yield suboptimal performance and many better algorithms have been proposed, although LRU still remains attractive due to its simplicity. An algorithm that takes into account the number of references, object size and download speed has been found to yield good and robust performance [19].

The main difference between a soft and a hard caching system is the existence of partial objects in the soft caching system. Initial simulations would seem to indicate that a strategy that takes into account the fact that the image has been recoded yields better performance (shorter average transmission time, higher hit-rate) than a strategy which ignores the state of the object.

How to account for recoding in the replacement algorithm? In an algorithm based on access times, such as LRU, we will need to adjust the object's access time or look also at the recoding time. Otherwise, an object that was chosen for removal would get chosen again after recoding as it would still be the least recently used.

A simple strategy would be to set the access time of an image which has just been recoded to the current time, in this way we will increase its priority in terms of the LRU algorithm. This tends to favor the lower resolutions too much compared to their utility, since images that have been recoded several times may potentially remain in the cache for a long period even if they have been accessed just once. A better strategy would set the access time to the old access time plus some fraction of the time elapsed between the last real access and the recoding instant based on the current recoding level.

Instead of modifying the access time, we could use the time when the object was recoded as a secondary key. We mark the time when an object is recoded and when the same object is selected for eviction the next time, we check if it has been accessed since it was recoded. If not, then it is released but if it has been accessed, we recode it again and set the recoding time accordingly. This is also known as the "second chance" algorithm [16].

For algorithms which are based on the size of the object and throw out the largest ones, using the actual size of the object would favor recoded objects over untouched ones. Since the quality of recoded images is inferior to that of the unrecoded ones this is not desirable. We should therefore use in the algorithm an estimated size which is calculated as the actual size plus a fraction of the difference between the original and actual sizes.

2.4 Summary

No matter what strategy is used to choose the objects to be evicted, we must decide whether the object should be released or recoded. The correct decision depends on the current state of the object as well as on general parameters such as number of recoding levels. This decision should carefully weigh the potential benefits gained (shorter transmission time, smaller size) if the image is recoded against the CPU-cost of doing so. If these benefits seem too small as is often the case for very small images, then the image should be released from the cache instead of wasting resources in recoding it.

Figure 2 depicts this decision based on image size and access frequency. We could define small images as those smaller than 15 KB and frequently referenced images as those that have been referenced at least twice.

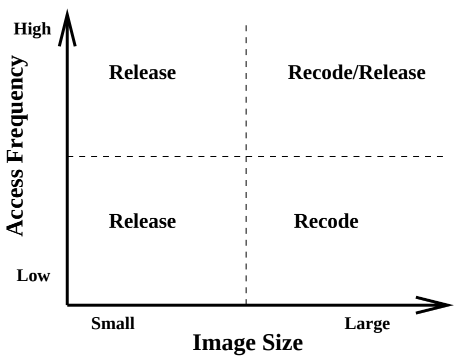

Figure 2: Release/recode-decision based on image size and access frequency

The decision whether to recode the image or release it in the top right quarter of Figure 2 depends on the availability of progressive transmission. Assuming that users are using browser caches, a frequently referenced object in a proxy cache is referenced by different users who will possibly have very different opinions about image quality. If we do not have progressive transmission, we should recode the frequently referenced large images only a few times in order to minimize the additional waiting time due to the retrieval of the whole object when the quality is not sufficient. With progressive transmission, however, we can recode them more since no unnecessary bits need to be transmitted even when the user requests a higher resolution.

After the object has been recoded we will need to update the changes (new size, recoding time) into the internal data structures. When the object is released, no special procedures are needed.

All in all, the modifications needed to convert a classical proxy to a soft caching proxy are small and limited mostly to the replacement algorithm. The actual recoding can and should be done in a separate program which is called by the proxy when recoding services are needed.

3 Programs

As our testbed for studying soft caching in practice, we have chosen the freely available Squid proxy [15]. We are currently using version 1.1.20.

We use recoding only on JPEG-images and as the format for recoded images we have chosen Progressive JPEG. It has the advantage of being supported by the browser and the freely available Independent JPEG Group's library [6] provides us with almost all the functionality we need to recode JPEGs to a coarser representation. Some JPEG-images on the Web are already in the progressive format, but most of them are sequential JPEGs. Therefore when the image is recoded for the first time it must also be transformed from the sequential format to the progressive format.

3.1 Modifications to Squid

We have implemented the main recoder as an external process (recoder daemon), like the `dnsserver`-process of Squid. The recoders are run on the same machine as the main proxy. It is possible to run multiple recoders. In this case, recoding requests are given to the recoder with the smallest load.

We have chosen to do recoding in two situations. In both of the situations, the recoding is performed on images that the LRU-algorithm has chosen for removal.

One of them is in function `storeGetSwapSpace` where Squid cleans up a large amount of space because the swap size has hit the high-water mark. In this case, we recode a JPEG-image by removing two layers.

The other recoding situation is in function `storeMaintainSwapSpace` which is called regularly to try to keep the swap size between the high- and low-water marks. In this case we recode by removing only one layer.

In all cases, if the image was chosen for removal because its explicit expiration time has passed, the image is released.

To make matters simpler, we recode only images that are on the disk, i.e. the "hot" objects that Squid keeps in memory are not touched. Also, should it happen so, that an image that has been handed to the recoder becomes "hot" and gets swapped into memory then we simply discard the recoded result when it is available.

The recoding function forwards the recoding request to the recoder daemon which then handles the details of recoding.

Because the recoding is not instantaneous, we will need to estimate the amount of disk space freed. Otherwise, Squid would keep on sending images to the recoder, since the replacement algorithm tries to free enough space to make the swap size go under a configured threshold.

As the estimate we have chosen 10% of the current image size, a value which is rather conservative in most cases and therefore should guarantee that after the recoding the swap size is below the configured maximum. Also it has the added benefit of recoding a bit more images than necessary which increases the probability of a user seeing a recoded image thus giving us more data on the visual quality of recoded images.

After the image has been recoded, the above estimate is corrected to the actual value of disk space freed. We also update the time when the image was last referenced. We have chosen to update the reference time to current time, i.e. make it look like the object was just accessed. This tends to keep the images in the cache and thus increases the chances of a user seeing a recoded image. In a production version this could be done in a more optimal way.

3.2 Experiences

We have been running our modified proxy in our lab and are thus far satisfied with the results. The modifications are stable, reasonably fast and the users are happy with the system.

Images where the recoding effects are clearly visible have occasionally been encountered but mostly the image quality has been satisfactory. This may in part be due to the equipment used, mostly Sun workstations with 8 bit graphics cards. In this setup the dithering algorithm of Netscape actually hides some of the recoding artifacts so that recoding cannot be seen before the last few levels.

With a 24 bit graphics card and a good monitor the recoding effects start to appear after a couple of recodings and become clearly visible after five recodings (at least to someone who knows what sort of artifacts to look for).

We have also tested how much recoding affects the performance of the proxy. For this experiment we preloaded a trace of 65000 URLs into a cache on our local network and ran the same trace through our modified Squid and a normal Squid using this preloaded cache as a parent. Using this setup we can easily see the performance of the proxy since the network has been taken out of play.

The trace had 65000 URLs, 700 MB of data, 60 % hit-rate and JPEGs represented 16 % of the bytes. The test proxies were quite small with 50 MB swap and the high-water mark at 95 % and low-water mark at 75 %. This setting makes it harder for the soft proxy to clear the space from the high-water mark to the low mark, since recoding does not free up space as quickly as a simple removal would. The requests were given at two speeds, 1 request per second and 3 requests per second on the average.

The metrics used were the average service time per request and the speed at which data was served in KB/sec as reported by the calamaris.pl analysis tool. Table 2 shows the results at 1 request per second and using 2 recoder-processes for soft proxy. Table 3 shows the results at 3 requests per second and using 1 and 4 recoders.

	Service Time (sec)	KB/sec
Squid	0.10	43.93
Soft-2	0.09	46.54

Table 2: Results with 1 request/second, 2 recoders

	Service Time (sec)	KB/sec
Squid	0.26	17.85
Soft-1	1.12	3.89
Soft-4	1.69	2.60

Table 3: Results with 3 requests/second, 1 and 4 recoders

From Table 2 we see that under a reasonable load the recoding process does not affect the performance, but from Table 3 we see that as the proxy is placed under an unreasonably high load, the processing time used by the recoders has a big impact on the performance of the proxy. It should be noted that in the second experiment the recoders are almost constantly running due to the proxy parameters. Even the normal proxy suffers a considerable performance hit under these extreme conditions.

Since recoded images are smaller than normal images, a soft caching proxy can keep more objects than a normal proxy. Figure 3 shows the number of objects in the two proxies during the experiment in table 2.

Figure 3 shows that the soft caching proxy (marked Soft-2) has sometimes considerably more objects than the normal proxy. These objects are recoded images, and figure 4 shows the distribution of the recoding levels during the experiment.

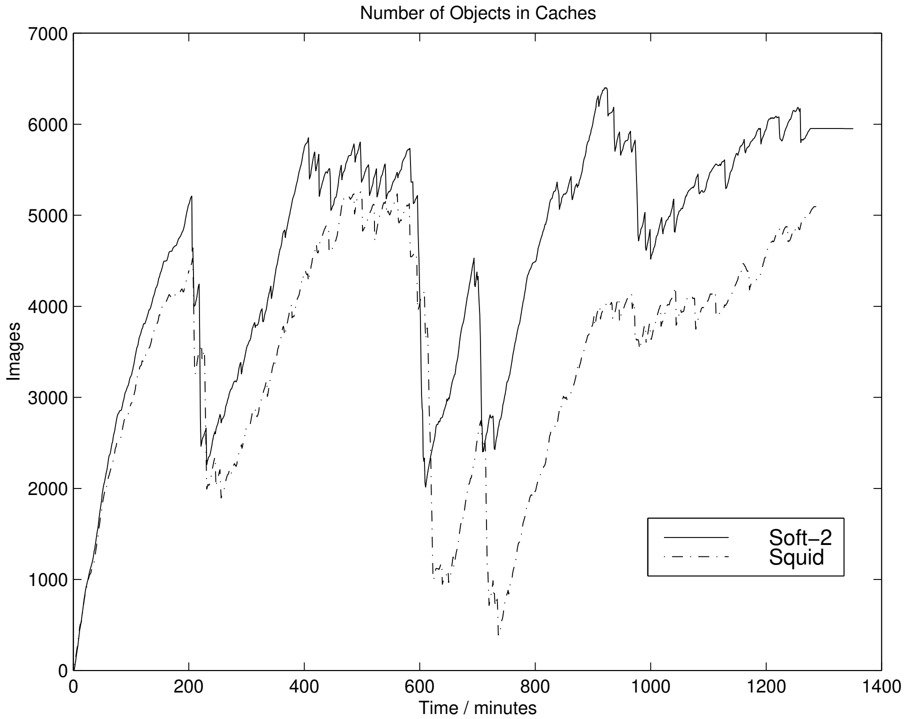

Figure 3: Number of objects in two proxies

4 Future Work

Currently when a user requests a reload on a recoded image, the whole image is transferred from the origin server. For normal JPEG-images this is the only way of doing it but if the original image is in the progressive JPEG -format then it would be enough to transfer only the missing bits. As Squid already implements HTTP/1.1-protocol, we can use it to study the advantages progressive transmission in a real-life setup by implementing range-requests for objects already in the progressive format.

When we have more experimental data on the visual quality of recoded images we will formulate a probabilistic model of user behavior that we will integrate into the replacement strategy. We will also do more detailed studies on how to include recoding levels into the different existing replacement policies. We also plan to study the implementation issues related to the optimal soft caching replacement policy presented in [20].

We have made our modifications available as a patch to squid 1.1.20. The patch is accessible through our project web page [13].

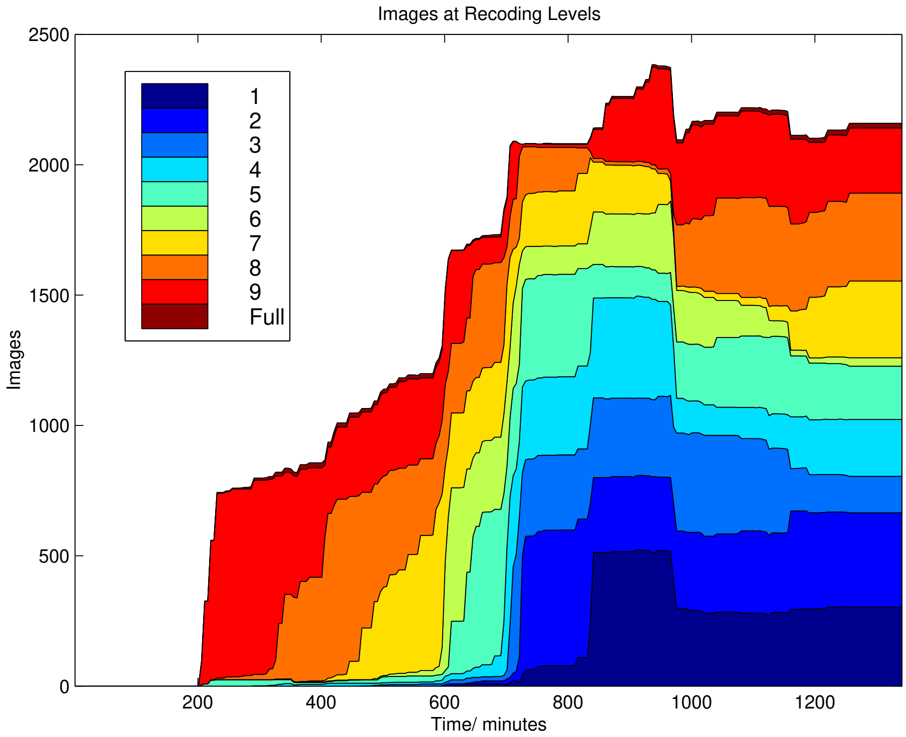

Figure 4: Distribution of recoded images

5 Conclusion

To accommodate soft caching, existing cache replacement algorithms have to be modified to take into account the fact that only a part of the object may be available. A decision whether to recode the object or release it must be made based on cache parameters. Based on experimental data we have studied both the number of recoding levels and image sizes and have found good values to be used in a real soft caching system.

We have presented our working soft caching system based on the freely available Squid proxy. We described the modifications necessary for achieving the desired functionality and presented our experiences obtained in using the modified proxy.

References

[1] P. J. Burt, E. H. Adelson, The laplacian pyramid as a compact image code, IEEE Trans. on Commun., vol. 31, pp. 532-540, Apr. 1983.

[2] A. Chankhunthod, P. B. Danzig, C. Neerdals, M. F. Schwartz, K. J. Worrell, A hierarchical internet object cache, in USENIX Tech. Conf., 1996.

[3] A. Fox, E. A. Brewer, Reducing www latency and bandwidth requirements by real-time distillation, in Proc. Intl. WWW Conf., (Paris, France), May 1996.

[4] A. Fox, S. D. Gribble, E. A. Brewer, E. Amir , Adapting to network and client variability via on-demand dynamic distillation, in Proc. Proc. Seventh Intl. Conf. on Arch. Support for Prog. Lang. and Oper. Sys. (ASPLOS-VII), (Cambridge, MA), Oct. 1996.

[5] Hypertext Transfer Protocol - HTTP/1.1 (RFC 2068).

[6] Independent JPEG Group's Software.
<URL:ftp://ftp.uu.net/graphics/jpeg/>.

[7] A. Ortega, Optimization Techniques for Adaptive Quantization of Image and Video under Delay Constraints. PhD thesis, Dept. of Electrical Engineering, Columbia University, New York, NY, 1994.

[8] A. Ortega, F. Carignano, S. Ayer, M. Vetterli, Soft Caching: Web Cache Management Techniques for Images, IEEE Signal Proc. Society Workshop on Multimedia Signal Processing, (Princeton, NJ), June 1997.

[9] A. Ortega, Z. Zhang, M. Vetterli, A framework for optimization of a multiresolution remote image retrieval system, in Infocom'94, (Toronto, Canada), pp. 672-679, June 1994.

[10] W. Pennebaker, J. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, 1994.

[11] P. Scheuermann, J. Shim, R. Vingralek, A case for delay-conscious caching of web documents, in Proc. Intl. WWW Conf., (Santa Clara, CA), Apr. 1997.

[12] J. M. Shapiro, Embedded image coding using zerotrees of wavelet coefficients, IEEE Trans. on Signal Proc., vol. 41, pp. 3445-3462, Dec. 1993.

[13] Soft Caching Project Page.
<URL:http://sipi.usc.edu/~ortega/SoftCaching/>.

[14] Spyglass Prism.
<URL:http://www.spyglass.com/solutions/technologies/prism/>.

[15] Squid internet object cache. <URL:http://squid.nlanr.net/>.

[16] A. S. Tanenbaum, Modern Operating Systems, Prentice Hall, 1992.

[17] C. Weidmann, M. Vetterli, A. Ortega, F. Carignano, Soft Caching: Image Caching in a Rate-Distortion Framework, in Proc. of ICIP, (Santa Barbara, CA), Oct. 1997.

[18] S. Williams, M. Abrams, G. Abdulla, S. Patel, R. Ribler, E. A. Fox, Removal policies in network caches for world-wide web documents, in Proc. of ACM SIGCOMM'96, (Stanford, CA), pp. 293-305, Aug. 1996.

[19] R. P. Wooster, M. Abrams, Proxy caching that estimates page load delays, in Proc. Intl. WWW Conf., (Santa Clara, CA), Apr. 1997.

[20] X. Yang, K. Ramchandran, An optimal and efficient soft caching algorithm for network image retrieval, in Proc. of ICIP, (Chicago, IL), Oct. 1998.

Vitae

Jussi Kangasharju was born in Helsinki, Finland, in 1972. He is pursuing the M.S. degree in the Department of Computer Science at the Helsinki University of Technology. Currently he is working as a visiting researcher at the Department of Electrical Engineering-Systems at the University of Southern California, Los Angeles.

His research interests include web caching, multimedia networking and Internet protocols.

Young Gap Kwon was born in Seoul, Korea, in 1968. He received the B.S. degree in Electrical Engineering from the Han-Yang University ,Korea, in 1993 , the M.S degree in the department Electrical Engineering from Electrical Engineering-Systems at the university of Southern California, Los Angeles, in 1996, where he currently is pursuing the Ph.D. degree.

His research interests include natural and synthetic image compression, video coding and web caching.

Antonio Ortega was born in Madrid, Spain, in 1965. He received the Telecommunications Engineering degree from the Universidad Politecnica de Madrid (UPM), Madrid, Spain in 1989 and the Ph.D. in Electrical Engineering from Columbia University, New York, NY in 1994. At Columbia he was a graduate research assistant at the Center for Telecommunications Research (1991-94) and was supported by a scholarship from the Fulbright commission and the Ministry of Education of Spain.

Since September 1994 he has been an Assistant Professor in the Electrical Engineering-Systems department at the University of Southern California. At USC he is also a member of the Integrated Media Systems Center, an NSF Engineering Research Center, and the Signal and Image Processing Institute. In 1995 he received the NSF Faculty Early Career Development (CAREER) award. In 1997 he received the USC School of Engineering Northrop-Grumman Junior Research Award. He received the 1997 IEEE Communications Society Leonard G. Abraham Prize Paper Award for a paper co-authored with Chi-Yuan Hsu and Amy R. Reibman. He is an Associate Editor for the IEEE Transactions on Image Processing and a member of the IEEE, ACM and SPIE.

His research interests are in the areas of image and video compression and communications. They include topics such as joint source-channel coding for robust video transmission, rate control and video transmission over packet wired or wireless networks. He has over 60 publications in international conferences and journals on these topics.