QBIX-G – A Transcoding Multimedia Proxy *

Peter Schojer, Laszlo Böszörmenyi, Hermann Hellwagner
Dept. of Information Technology, University Klagenfurt, Austria
{pschojer,laszlo,hellwagn}@itec.uni-klu.ac.at

ABSTRACT

Due to the increasing availability of audio/visual data on the Internet, proxy caching is gaining on importance as a performance factor. This increase is accompanied by a diversification in the end terminals, which calls for media gateways and filters.

An *adaptive* proxy is presented which performs (1) caching, (2) filtering and (3) media gateway functionality in one. The proxy can perform media adaptation – using transcoding – on its own. A cost model is presented which incorporates user requirements, terminal capabilities and video variations in one formula. Based on this model, the proxy acts as a general *broker* of different user requirements and of different video variations. This is a first step towards *What You Need is What You Get* (WYNIWYG) video services, which deliver videos to users in exactly that quality what they need and what they want to pay for. The MPEG-7 and MPEG-21 standards enables this in an interoperable way. A detailed evaluation based on a series of measurements is provided.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software; H.5.1 [**Information Systems**]: Multimedia Information Systems

Keywords

Video proxy, video caching, media gateway, media adaptation, MPEG-4, MPEG-21, cache replacement strategies

1. INTRODUCTION

It is well-known that client-side proxies can give substantial support for video delivery over the Internet. Traditionally, such proxies provide two basic functionalities: they serve (1) as a *firewall* and (2) as a *cache*.

*This project was funded in part by FWF (Fonds zur Förderung der wissenschaftlichen Forschung) P14788 and by KWF (Kärntner Wirtschaftsförderungsfonds).

These basic functionalities can be considerably extended if we take into consideration that video delivery is getting ever more challenging, partly due to the heterogeneity in user requirements, partly also due to greatly diverse equipment, characterized by (1) different connectivity (ranging from high speed LANs over UMTS to slow connections over modem or GSM) and (2) different computational power (ranging from workstations over PDAs to cell phones). In such a heterogeneous environment, the proxy can take over a much more general role than usual by serving different user and terminal classes by different video quality classes. It can act as a kind of *media broker* that (1) understands the *preferences* and *capabilities* of the user and (2) can handle different *variants* of the same video. Based on these inputs, it can perform an optimal *match* between the needs of the user and the possibilities of the provider. It can further detect when a request cannot be fulfilled and acts in this case as a *request filter* that protects both server and client from videos that the client ought to abort due to a quality mismatch.

The *MPEG-7* standard provides tools to describe different variants of a video and the emerging *MPEG-21* standard provides tools to describe user preferences, terminal and network capabilities in an *interoperable* way. This enables us to build *What You Need is What You Get* (WYNIWYG) video services. The users do not get just the available quality nor the best possible quality, but exactly that quality which they need and which they are ready to pay for.

This paper introduces a novel concept for such a brokering proxy. The proxy may cache videos in different quality variants. If a video is getting popular for users both with high and with low capabilities then it caches both corresponding variants. Thus, it acts not only as a cache but also as a *media gateway*.

Lower quality variants may be available in a very efficient way due to layered encoding. In the MPEG standardization group, great efforts are in progress to define an efficient layered video coding scheme. Currently, however, layered coding (as defined in MPEG-4) is not supported by virtually any codec. If the video is not available in layered coding, then the proxy can perform *transcoding* on its own. Transcoding may be of course a time consuming process, therefore the proxy has to consider its costs.

We introduce a *cost model* that controls the decisions of the proxy. To our knowledge this is the first work on a quality

aware video cache that combines partial caching in the quality domain with a differentiated model of user preferences, of video variations and of the caching costs. A nice feature of this approach is that we get the gateway and a basic filtering functionality dynamically and for free as a "side effect".

The paper presents the simulation results of the brokering proxy. Based on these, the algorithms are currently integrated into our Quality Based Intelligent Proxy [5].

2. BASIC NOTIONS

2.1 Client-Side Proxy Cache

A proxy cache is a computer residing between client and server and caching data which the client is requesting from the server. A *client-side proxy* (from now just proxy) has normally a faster connection to its clients then to its servers. Clients send their requests to the proxy, which tries to fulfill the requests from its local cache, if possible. Otherwise it forwards the requests to the server and stores a local copy of the data on its disk. Ideally, a client-side proxy reduces load and network traffic on the server side and gives clients a reduced startup delay.

2.2 Media Gateway

A media gateway is similar to a proxy in inspecting the data flows between server and client but it may also modify them according to some transcoding rules. The transcoding itself can be hinted by meta-data, such as user preferences or terminal capabilities or it can be hard coded. A media gateway node transcodes videos to some specific quality characteristics, such as a given dimension, bitrate, color etc.

2.3 Media Adaptation

In the context of video transmission, media adaptation means the transformation of an already compressed video stream. Media adaptation can be classified into three major categories: (1) bitrate conversion or scaling (including frame dropping, i.e., temporal conversion), (2) resolution or spatial conversion, and (3) syntax conversion. Bitrate scaling can adapt to shortages in available bandwidth. Resolution conversion can adapt to bandwidth limitations, but it can also accommodate for known limitations in the user device in processing power, memory, or display size. Syntax conversion is used in hybrid networks to match server and client compression protocols. If we need to decompress and re-encode a video for adaptation, we speak about adaptation in the *decompressed domain*, a slow and resource-intensive task. More lightweight adaptation in the *compressed domain* is therefore preferable.

2.3.1 Adaptation and Cache Replacement

Cache replacement strategies in the area of video caching are divided into two categories: *full caching* and *partial caching*.

With *full caching* videos are handled like normal Web objects, with the disadvantage that videos are huge and only a small number of videos can be cached at one node; thus, hit rate is low. With partial caching, only a selected part of a video is cached, e.g., only a prefix [10], or bursty parts of a video [12], hotspot segments [2] etc.

In this paper we concentrate on *partial caching* in the quality domain only. Related work in this area mostly relies on layered coded videos, which reduces adaptation to the simple case of deleting the highest available enhancement layer. Examples are periodic caching of layered coded videos [3], combination of replacement strategies and layered coded videos [6], quality adjusted caching of GoPs (group of pictures) [9], adaptive caching of layered coded videos in combination with congestion control [8] or simple replacement strategies (patterns) for videos consisting of different quality steps [7]. Most of these proposals rely on simulation to evaluate the performance of the caching techniques.

None of these proposals considers user preferences or reload behavior due to quality mismatches.

2.3.2 Codecs and Adaptation

Most codecs do not support layered encoding, although this is a requirement for fast and efficient adaptation. One of the first widely used standards with rudimentary adaptation support was MPEG-2, which allowed the definition of one single enhancement layer. This feature was pretty much ignored by content providers.

MPEG-4 is actually the first standard that offers extensive adaptation options, i.e., temporal, spatial and bitrate scalability through the means of layered encoding. Most implementations of the standard in software/hardware do not (yet) support this feature, but restrict themselves to the *simple profile* part of MPEG-4, which does not even support B-frames.

2.3.3 Adaptation and Media Gateways

Using adaptation in media gateways on content that does not support layered encoding creates several problems. The first problem is the high burden on the CPU created by resource intensive decoding and encoding operations. For example a 1.4 GHz Pentium IV processor is capable of performing bitrate transcoding on only three CIF(352x288) streams in parallel (measured with XviD) [5].

Another problem is the reduced hit rate in the cache. It is a common assumption that request patterns follow the Zipf distribution. The higher the Zipf α value, the higher is the hit rate in the proxy[1]. Without layered encoding the proxy stores n variations of one and the same video in the cache. Due to differing user preferences, the requests now do not accumulate on one single object but are distributed over n variants. This "scattering" disturbs the original Zipf distribution and has the effect as if the α value were reduced. The number of one-timers (videos that are requested only once) increases and even a class of *zero-timers* is introduced. Zero-timers are videos that are used only as transcoding source but are never explicitly requested by any client. As we have to lock these videos when using them as a transcoding source, they may remain in the cache for a fairly long time.

Moreover, the size of these n variants is in total greater than the size of the stream in layered encoding format. Thus, a

[1]Higher α means more skewed popularity distribution.

media gateway can store more video objects than a Web proxy but less *different* videos.

The advantage of the gateway functionality is that reloading of a video due to quality mismatches happens significantly less frequently and that costumer satisfaction should be considerably higher. Costumer satisfaction is of course generally hard to measure – that is the reason why we take user preferences and costs into consideration.

2.4 User Preferences

There are currently two major standards available for communicating user preferences to a server. The first one is CC/PP (Composite Capability/Preference Profiles) [11] which is a standardized framework developed by the W3C as an extension to the HTTP 1.1 standard. It is a collection of the capabilities and preferences associated with a user and the configuration of hardware, software and applications used by the user to access the World Wide Web. The disadvantage of this protocol is that it fails to allow users to specify priorities for features, e.g. to prefer bitrate over dimension.

The other major standard is MPEG-21 [1], specifically the Digital Item Adaptation (DIA) part, which will become international standard in 2004. The advantage of MPEG-21 DIA is that it was designed with content adaptation in mind.

3. QBIX-G

QBIX-G (Quality Based Intelligent proXy Gateway) realizes the combined media gateway/proxy cache functionality. It features standard-compliant RTSP, with extensions that allow clients to transmit their user preferences to the proxy, it allows real-time transcoding of avi, MPEG-1/-2/-4 videos to the MPEG-4 format.

3.1 Scenarios

The client sends an RTSP DESCRIBE request, which contains the URL of the requested video and the user preferences of the client.

The proxy checks in its cache, whether it can find a version that matches the user preferences. Four different scenarios can now occur:

1. *Object miss with quality miss*: A full miss is given when the proxy either finds a version with too low a quality or it does not find any entry for the given URL. In this case, the proxy has to forward the request to the server. If the server does not support transcoding, the proxy has to remove the user preferences from the DESCRIBE field. The video stream received from the server is then adapted on the proxy according to the user preferences as specified by the client.
2. *Object miss with quality hit*: This situation is similar to the one above, except that the video coming from the server matches the user requirements, either because the server did the transcoding, or because the original version happens to have the required properties.
3. *Object hit with quality miss*: The proxy finds a cached version of the video but the quality of the object is too high and transcoding is needed.
4. *Object hit with quality hit*: The proxy finds a cached version where quality is within the ranges specified in the user preferences.

The proxy forwards the created/found version of the video to the client when receiving the RTSP SETUP and PLAY requests and depending on the hit/miss scenario it tries to cache the original/transcoded version (if not yet cached).

3.2 Cost Function

QBIX-G uses internally a cost function to decide which version of a video should be sent or created.

3.2.1 Requirements

The cost function should fulfill the following requirements:

- The higher the actual load of a resource, the more expensive it should be to use that resource.
- Cache hits should be preferred over cache misses.
- It should be possible to assign weights to resources, according to their importance.
- Clients requesting transcoding and not paying enough should be rejected (commercial scenario only).
- It should try to maximize quality if enough resources are available.

3.2.2 Input

Let V be the set of videos cached at the proxy, let S be the set of videos requested by all clients, $V \subseteq S$. Each video $v \in S$ is described by a feature set f_v. For visual streams, the feature set is declared as follows:

$$f_v = \{dimX, avgBitRate, color, frameRate\}.$$

Each video v is uniquely identified by a URL and the associated features f_v. The feature set specifies all parameters that a client is allowed to change. In addition, the *aspect ratio* of a video, which is used to calculate $dimY$, is known but is treated as an invariant, which is not changeable neither by the proxy nor the client. A single request r consists of the URL of the video v and, for each single feature $f \in f_v$, an *acceptance range* $[min, best, max]$ and an associated *importance value*. All importance values must sum up to 1.

The client also specifies the maximum delay she is willing to wait for the service in milliseconds. For commercial scenarios a client must also specify an upper limit of money she is willing to pay for the proxy service.

3.2.3 Quality

$Quality(val_f, range_f, importance_f)$ for a feature f is defined as a "distance" of the corresponding feature value val of the video from the *best* value.

As shown in figure 1, quality is zero if $val < min$ or $val > max$. If $val = best$, then we achieve the maximum possible quality for this feature, which is equal to *importance*. We defined the edge points of the acceptance range (min, max)

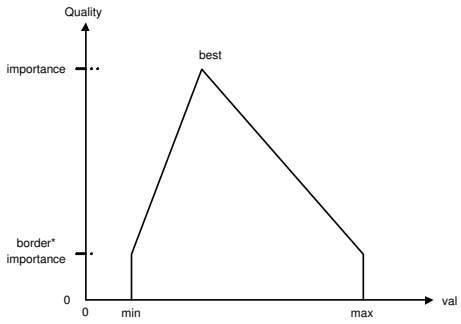


Figure 1: Simple Quality Function

to have some minimum *border* quality ($0 < border < 1$)**:** $border * importance$.

For simplicity, we assumed that quality degrades linearly in the ranges [min, best] and [best, max].

Overall quality of a video v for a request r is defined as

$$Quality(r) = \sum_{i=1}^{|f_v|} Quality(val_i, range_i, importance_i)$$

or zero, if only a single feature with an importance value greater zero returns 0. The maximum possible quality value is 1, which means that a perfect hit was found.

By specifying such ranges, the client explicitly states that she will not reject the video due to a quality mismatch[2]. If no user preferences are specified, it is assumed that the client requests the video with the highest quality.

3.2.4 Costs

We calculate costs for the resources network, CPU and hard disk. Each resource is described by an upper limit and a current load. For network, NET specifies the actual load, for hard disk HD, and for the processor CPU.

Costs depend directly on the amount of resources currently available and the price for each resource, namely p_{net}, p_{cpu} and p_{hd}. In the first step the proxy calculates for a request r the resource usage in percent of the maximum of each resource: net_r for network, cpu_r for CPU, and hd_r for disk access. For example, if we encounter a cache miss to a file with a source bitrate of 1 MBbit/sec, which should be sent out with 0.5 Mbit/sec – after corresponding transcoding – and the proxy has only one single network card with a connection speed to the Internet of 10 Mbit/sec, we calculate $net_r = (1+0.5)/10 = 0.15$. Calculating costs for disk access is similar.

CPU costs for transcoding are currently simplified to two cases. The first is temporal adaptation where B-frames are dropped in the compressed domain and thus transcoding costs are virtually zero. The second case is adaptation in the decompressed domain, which is simplified to decoding and encoding, which are the dominant factors. The costs for the down-scaling or grey-scaling operations proper can be neglected. We measured on MPEG-4 videos that at encoding and decoding, the costs depend directly on the amount of pixels processed per second, and that encoding is about four times more expensive than decoding [5]. By benchmarking a system, one can estimate how many pixels per second a system can decode, e.g., for a four-processor system (Pentium 4, 1.4 Ghz) a value of 150 Mpixels is realistic. Costs are calculated as $dimX * dimY * framerate$ for decoding; for an encoding operation, the value is multiplied by 4.

The formula for resource costs for a given request r is given as

$$ResourceCosts(r) = \frac{1}{1 - NET - net_r} * p_{net} + \frac{1}{1 - CPU - cpu_r} * p_{cpu} + \frac{1}{1 - HD - hd_r} * p_{hd} + \frac{startupDelay}{maxDelay}$$

Note that by including the current resource usage, the cost function also acts as an admission control, which assigns infinite costs to requests which violate one of the following constraints: $NET + net_r < 1.0$, $CPU + cpu_r < 1.0$ and $HD + hd_r < 1.0$

The inclusion of the startup delay increases the costs for a cache miss. The actual load of the current resources handicaps resource intensive operations if free resources are sparse.

Billing costs are calculated as

$$BillingCosts(r) = ResourceCosts(r) + ContentCosts(r, quality) + profit_r.$$

If the billing costs exceed the amount of money a client is willing to pay, the request is rejected. Note that the content costs function has to be provided by the content owner. If no function exists, the proxy must bill full costs even for a lower quality version of the video. $profit_r$ is the minimum profit the proxy makes by servicing a single request.

FinalCosts are calculated as

$$FinalCosts(r) = (1 - Quality(r)) * ResourceCosts(r)$$

If the resource costs are infinite or quality is 0, infinite final costs are returned.

The disadvantage of this formula is that it ignores visual quality. For instance, consider the case where the proxy has cached two variants of the same video, a high-resolution and a low-resolution version, and a client requests an even lower-resolution version. The proxy will calculate the resource costs, quality and the final costs for the following two possibilities: $highRes \rightarrow veryLowRes$ and $lowRes \rightarrow veryLowRes$. Our quality function will in both cases return the value how exactly the features of the $veryLowRes$ video

[2]She might still reject the video due to its content.

meet the user preferences of the client. In both possibilities the target features are the same, thus the returned quality value will be equal for both. Decoding a high-resolution video is naturally more expensive than a low-resolution video, which results in higher resource costs – and thus higher final costs – for the first scenario. However, starting from the source with the larger resolution, the first version may also result in better visual quality, which is currently not addressed by our approach.

3.2.5 Finding the Best Version

For a request r, the proxy searches all videos in V that have the same URL. For each video version found, the proxy calculates its quality value. If one version is found that returns a quality value greater than zero, we have a quality hit (no transcoding is necessary) and can start streaming this version.

In the other case, we have to calculate the final costs for each version that meets the user preferences. The $source \rightarrow target$ pair with the lowest final costs is chosen and the video is adapted and streamed to the user in real-time.

In case the proxy cannot find any cached version that allows the proxy to generate a quality hit, it contacts the server and repeats its calculation with the original video as source.

To reduce the number of possible variations stored in the proxy, we allow transcoding only to specific discrete points:

- Feature $dimX$ must be a multiple of 44, $dimY$ is calculated according to the aspect ratio of the original video.
- Average bitrate per second is defined as $dimX * dimY * b$ bits/second, $b \in \{1, 2, 4, 8\}$.
- Framerate should be a multiple of 5 or equal to the original framerate; if the video contains B-frames, the adaptation result can vary but a version created by dropping B-frames is never cached[3].
- Color can only be true or false.

If no pre-cached version is found that has a quality greater than zero, or which allows to create a version that has a quality higher than zero, a quality miss is encountered, and the original video has to be fetched from the server.

3.2.6 Example

Assume that a server offers the video *dummy.mp4* with the following features: dimX=352, bitrate=912384, color=true, framerate=25.00 fps, 12 of 25 frames are B-frames (all B-frames account for 20% of the video's bitrate), the content is freely available. Aspect ratio is a constant with a value of $\frac{11}{9}$, resulting in $dimY = 288$. A client is requesting this video with dimX: (176, 197, 219)/0.20; bitrate: (64000, 183999, 303999)/0.20; color: true/0.20; framerate: (15.00, 15.00, 30.00)/0.20; maximum delay: (0,0,1000)/0.20. The proxy charges $p_{net} = 10$, $p_{cpu} = 10$, $p_{hd} = 10$, current resource usage is zero.

According to the user preferences, the proxy calculates for each feature the following possible values: $dimx = \{176\}$ (thus $dimY$ is set to 144), $bitrate = \{101376, 202752\}$, $color = \{true\}$ and $framerate = \{15, 20, 25, 30\}$. The proxy calculates for all combinations ResourceCosts, Quality and FinalCosts:

Features	Costs	Quality	Final
(176,101376,true,15.0)	**8.969**	0.536	4.161
(176,101376,true,20.0)	9.068	0.476	4.751
(176,101376,true,25.0)	9.167	0.416	5.353
(176,101376,true,30.0)	9.269	0.356	5.969
(176,202752,true,15.0)	8.994	**0.632**	**3.311**
(176,202752,true,20.0)	9.092	0.572	3.893
(176,202752,true,25.0)	9.192	0.512	4.487
(176,202752,true,30.0)	9.294	0.452	5.094

Thus, the proxy will insert into the (initial empty) cache the following versions (most popular first):

- (176,202752,true,15.0) – required quality
- (352,912384,true,25.0) – original quality

4. EVALUATION

Before integrating the cost function into QBIX-G, experiments were performed in order to test the idea. We assumed $p_{net} = 10$, $p_{cpu} = 10$, $p_{hd} = 10$, a disk bandwidth of 10 Mbyte/sec and a network bandwidth of 80 Mbit/sec. For CPU speed we assumed a two-processor system, which is capable of transcoding approximately six streams in parallel (bitrate scaling of CIF stream).

We used WebTraff [4] to generate a list of 10000 requests[4], for the request pattern we assumed a Zipf distribution with $\alpha = 1.0$ and 0.3. We simulated 1000 videos each with a dimension of 352x288, a framerate of 30 and a constant bitrate of 912384 bits/sec. The total size (thus also the duration) of the videos followed a Pareto distribution with the tail index set to 1.2. On average, videos had a duration of 86 seconds, request interarrival time was set to 20 seconds.

The frame pattern was set to $IPBPB...$, which allowed a 30 fps video to be temporally adapted down to 16 fps. B-frames contributed to 20% of the total bitrate. The total size of all videos was approximately 9 GB, cache sizes were set in the range from 1% up to 10% of the total video size, the number of one-timers in the request sequence was set to 30%. For cache replacement we used standard LRU.

We ran several benchmarks with the number of users requiring transcoding (e.g., for mobile devices) varying between 0% and 100%. Such mobile users (as referred to in the remainder of the paper) were simplified to use one of four different devices, with the corresponding user preferences

[3]It would be counterproductive to cache a version which can be generated on the fly so easily.

[4]The number of requests generated by WebTraff was actually slightly lower.

dimX	bitrate	delay in msec	color	framerate
(144,172,200)/0.20	(48000,172999,297999)/0.20	(0,0,1000)/0.20	false/0.20	(15.00,15.00,30.00)/0.20
(201,229,257)/0.20	298000,422999,547999)/0.20	(0,0,1000)/0.20	true/0.20	(15.00,20.00,30.00)/0.20
(258,286,314)/0.20	(548000,672999,797999)/0.20	(0,0,1000)/0.20	true/0.20	(15.00,25.00,30.00)/0.20
(315,344,374)/0.20	(798000,923000,1048000)/0.20	(0,0,1000)/0.20	false/0.20	(15.00,30.00,30.00)/0.20

Table 1: User Preferences

shown in table 1. Mobile requests were pseudo-randomly distributed over the whole request sequence with every class being equally important.

We assume that the server does not do transcoding, all transcoding work is done by the proxy. None of the videos is present in a layered encoded format, thus transcoding is never used for cache replacement.

4.1 Transcoding Rules

In case a quality miss is encountered, the proxy tries to match the stored versions to the request according to the following guidelines:

- Bitrate miss: Drop B-frames until in bitrate range or until no B-frames left. If the result violates the acceptable frame/bitrate range of the client, do transcoding in the decompressed domain. Never cache a result generated by temporal adaptation, simply assign the hit to the original video.
- Spatial miss: Always transcode, put the generated video to the beginning of the LRU list, do not change the position of the original video.
- Framerate miss: Drop B-frames until in framerate range or no B-frames left. If the resulting fps value is still too high, transcode and additionally drop frames in the decompressed domain. Only insert transcoded versions into the proxy cache.

If the proxy cannot create a version that matches the request (because of an invalid request or due to its admission control), it rejects the request.

4.2 Measured Parameters

The following parameters are measured during benchmark execution:

- Rejected Requests: How many requests were rejected due to unsatisfiable requests, either because the admission control rejected the request or because the requested transcoding step is not supported in the proxy? A rejected request also counts as an object miss.
- Quality Hits: How many requests could be fulfilled directly from the cache without the need for further transcoding?
- Object Hits: How many requests could be fulfilled from the cache (including hits that needed adaptation)?
- Byte Miss Rate: How many of the requested bytes had to be fetched from the media server when an object miss was encountered or a request was rejected? We decided to include the latter to distinguish between our proxy which detects rejection caused by quality mismatch in advance and a traditional proxy which would try to service even absurd requests such as streaming an HDTV video to a cell phone.
- Not Cached Due To Locking: How many videos could not be inserted into the proxy cache because it could not free up enough space due to file locking?
- Not Cached Due To Size: How many videos were not inserted due to the video object exceeding a size threshold value?

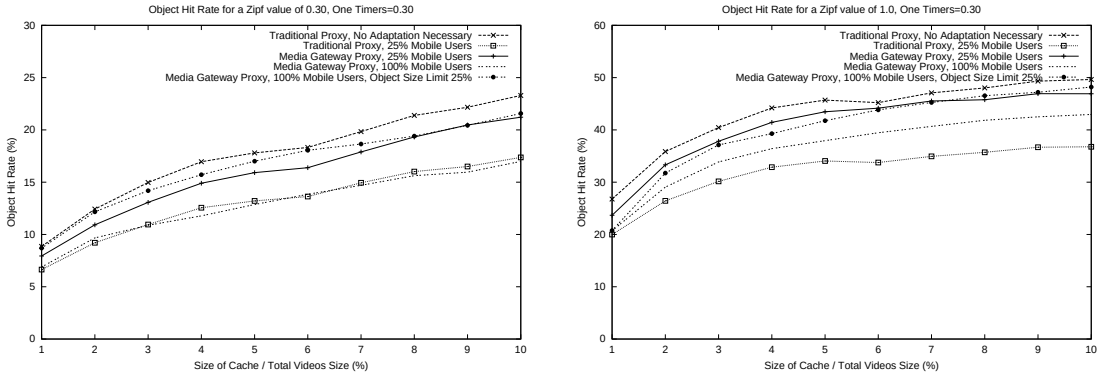
4.3 Results

We present the results from five different scenarios. The first is the typical business user scenario, where no mobile clients are present and our media gateway acts as a conventional Web proxy. A more realistic approach assumes that up to 25% of all requests are from mobile users. We measured this use-case with transcoding turned off, which resulted in a rejection rate of 25%, and with media gateway functionality enabled, with a rejection rate of 0%. The (unlikely) extreme case, where all clients require adaptation was also benchmarked, once with the cache size being the upper size threshold limit for video objects, and once with with limiting the size of the objects at 25% of the cache size. Note that request rejection due to admission control occurred only in the unlikely 100% mobile scenario, where at most 4% of the requests were rejected.

4.3.1 Object Hit Rate

As shown in figure 2, the proxy-only scenario without adaptation proves to be the best in object hit rate, but as soon as adaptation is a necessity, the picture changes. The traditional proxy with mobile users at 25% is now the worst solution.

The 100% mobile users scenario shows the price one pays for using media gateway functionality without layered encoding. In the less skewed case with $\alpha = 0.3$, object hit rate is nearly equal or worse to the traditional proxy case, where 25% of the requests were rejected! With a more skewed distribution ($\alpha = 1.0$), it shows a much better hit rate. There are several reasons for this behavior: first, none of the videos supported layered encoding, thus the sum of the size of all transcoded versions of one video is larger than the size of one single layered encoding version of the same video. Second reason is that if there is enough disk space in the proxy cache, we also store the original versions of the videos. This causes problems later when cache replacement is triggered and the videos the proxy wants to evict are locked. For example,



(a) Zipf $\alpha = 0.3$ (b) Zipf $\alpha = 1.0$

Figure 2: Object Hit Rate

when transcoding a two hour video in the decompressed domain, this video, and the generated transcoded version, will be locked for the whole play-out time, i.e. two hours. Thus, large files can remain for very long time in the cache, even if they are one- or zero-timers.

To solve this problem we introduced the use case with the small threshold value of 25% of the cache size. With such a small threshold value, for most videos the original version and large transcoded versions are never cached but instead smaller videos are. Due to the fact that the majority of requests goes to videos that are below this threshold, this results in an overall improvement of object hit rate by up to 6%. This result is very close to the hit rate a traditional proxy can achieve.

4.3.2 Quality Hit Rate

This measurement is used to detect how much transcoding work can be saved by the media gateway functionality. When comparing figure 2 with figure 3, one sees that in the non transcoding scenarios Quality Hit Rate and Object Hit Rate are equal. In the 25% mobile users case, our media gateway offers a higher quality hit rate than the traditional proxy. Again, the lack of layered encoding forces our gateway proxy to "waste" disk space on several self contained versions of one video. Thus, the higher the amount of mobile users is, the lower is the hit rate. Again, introducing the lower size threshold proves to be beneficial to the quality hit rate of the 100% mobile case. In the $\alpha = 1.0$ case, quality hit rate is as high as 38%, which means that approximately every third request can reuse an adaptation result from a prior request.

4.3.3 Byte Miss Rate

Again, the traditional proxy is best, with media gateway functionality increasing the miss rate by a few percent (see figure 4). With the small threshold limit set, byte miss rate is slightly worse than in the standard case.

4.3.4 Locking

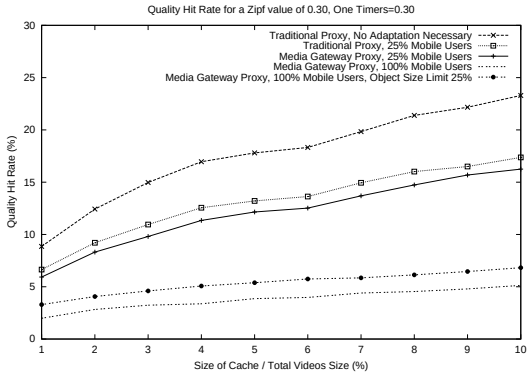
Generally speaking, locking has a worse effect in a proxy that does transcoding than otherwise. As shown in figure 5, 13% to 34% of all video insertions fail due to locking whereas in the non-transcoding scenarios this value is clearly lower. Interestingly, locking seems to be worse with highly skewed request distributions. The reason is that some large source videos remain in the cache for a very long time, e.g., consider a popular 100 second video being used as transcoding source. With $\alpha = 1.0$ it is very likely that during the time the original video is locked, another request will need the same video for a different transcoding step and extend the lock time for the original video. Thus, a constant (large) amount of the proxy cache is always locked, with the other less popular requests competing for the reduced space.

Introducing a threshold of 25% significantly reduces the locking problem. Due to the size limit, most original video versions are never cached and thus cannot block other videos. Naturally, the rejection rate of videos because of their size goes up as shown in figure 6. Up to 60% of all video insertions fail. Note that for a sequence of 10000 requests, where all requests requiring transcoding, the number of insertions is as high as 16000 for the less skewed case (cache size 10%). With $\alpha = 1.0$ this value is reduced to less than 10500.

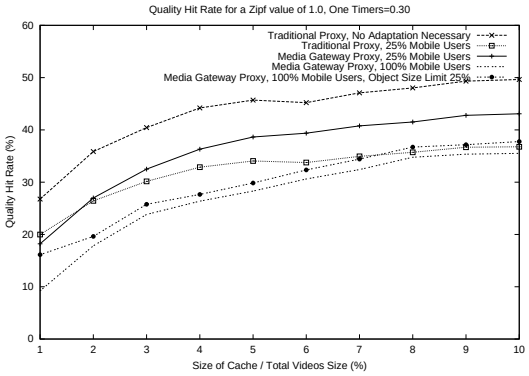
5. CONCLUSION AND FUTURE WORK

In this paper, we have presented a multimedia proxy gateway that makes a first step towards offering *What You Need is What You Get* services. By combining user preferences, resource usage and quality into one cost formula, we are able to determine which version will give the client a good enough quality with acceptable costs at the proxy. We have shown the effects of transcoding in the decompressed domain on the byte, object and quality hit rate and that locking is a major problem if the number of adaptation requests is high and one is forced to rely solely on transcoding. As long as the number of devices requiring adaptation remain a minority, transcoding is a feasible processing step in a media gateway proxy and a useful addition to layered encoding support. The minor loss of object hit rate is compensated by the gained functionality and will be further reduced when layered encoding is available.

For further work, we will extend our simulations to include layered encoding, and implement the cost function and user preferences support into our open-source RTSP proxy implementation QBIX, which in turn is part of our multimedia framework ViTooKi (Video Tool Kit, available at `http://vitooki.sourceforge.net`). The integration is conform to the MPEG-7 and MPEG-21 standards.

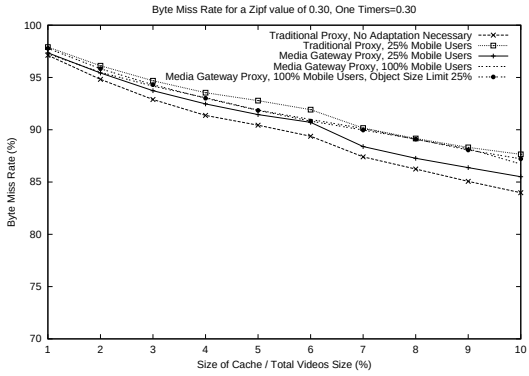


(a) Zipf $\alpha = 0.3$

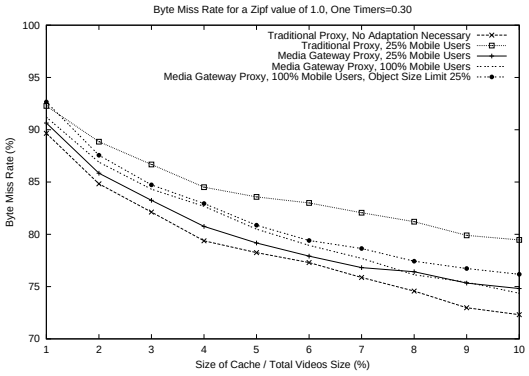


(b) Zipf $\alpha = 1.0$

Figure 3: Quality Hit Rate



(a) Zipf $\alpha = 0.3$



(b) Zipf $\alpha = 1.0$

Figure 4: Byte Miss Rate

6. REFERENCES

[1] J. Bormans and K. Hill. N5231 - MPEG-21 Overview v.5. *http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm*, Oct. 2002.

[2] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. H. Hsu. Proxy Servers for Scalable Interactive Video Support. *IEEE Computer*, 43(9):54–60, Sept. 2001.

[3] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross. Distributing Layered Encoded Video through Caches. In *Proceedings of IEEE INFOCOM*, pages 622–636, Apr. 2001.

[4] C. W. N. Markatchev. WebTraff: A GUI for Web Proxy Cache Workload Modeling and Analysis. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, number 10, pages 356–363, Oct. 2002.

[5] P. Schojer, L. Böszörmenyi, H. Hellwagner, B. Penz, St. Podlipnig. Architecture of a Quality Based Intelligent Proxy (QBIX) for MPEG-4 Videos. In *ACM World Wide Web Conference*, pages 394–402, May 2003.

[6] S. Paknikar, M. Kankanhalli, K. R. Ramakrishnan, S. H. Srinivasan, and L. H. Ngoh. A Caching and Streaming Framework for Multimedia. In *Proceedings of ACM Multimedia*, pages 13–20, Nov. 2000.

[7] S. Podlipnig and L. Böszörmenyi. Replacement Strategies for Quality Based Video Caching. In *IEEE International Conference on Multimedia and Expo (ICME)*, volume 2, pages 49–52, Aug. 2002.

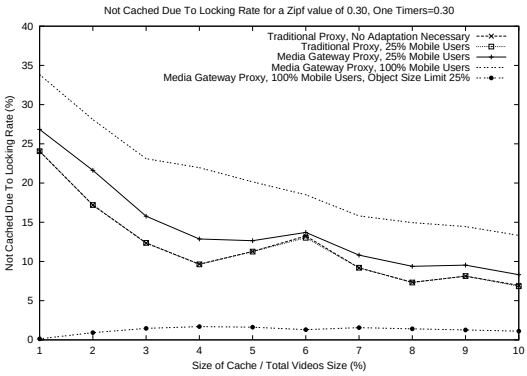
[8] R. Rejaie and J. Kangasharju. Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In *11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 3–10, June 2001.

[9] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Proxy Caching Mechanisms With Video Quality Adjustment. In *Proceedings of the SPIE Conference on Internet Multimedia Management Systems*, pages 276–284, Aug. 2001.

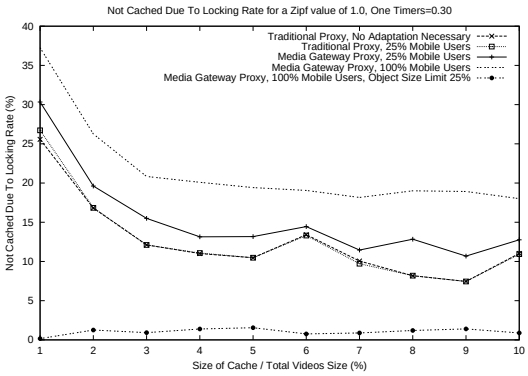
[10] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proceedings of IEEE INFOCOM'99*, pages 1310–1319, Mar. 1999.

[11] W3C. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 (W3C Recommendation). *http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/*, Jan. 2004.

[12] Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Shu. Video Staging: A Proxy-Server-Based Approach to End-to-End Video Delivery over Wide-Area Networks. *IEEE/ACM Transactions on Networking*, 8(4):429–442, 2000.

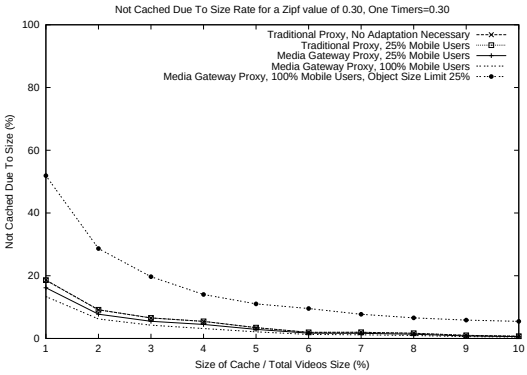


(a) Zipf $\alpha = 0.3$

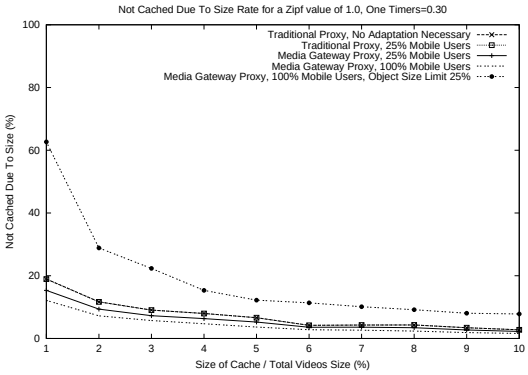


(b) Zipf $\alpha = 1.0$

Figure 5: Not Cached Due To Locking



(a) Zipf 0.3



(b) Zipf 1.0

Figure 6: Not Cached Due To Size