Multimed Tools Appl (2006) 31: 221–245
DOI 10.1007/s11042-006-0037-z

Caching video objects: layers vs versions?

Felix Hartanto · Jussi Kangasharju · Martin Reisslein · Keith Ross

Published online: 23 September 2006

Abstract Because Internet access rates are highly heterogeneous, many video content providers today make available different versions of the videos, with each version encoded at a different rate. Multiple video versions, however, require more server storage and may also dramatically impact cache performance in a traditional cache or in a CDN server. An alternative to versions is layered encoding, which can also provide multiple quality levels. Layered encoding requires less server storage capacity and may be more suitable for caching; but it typically increases transmission bandwidth due to encoding overhead. In this paper we compare video streaming of multiple versions with that of multiple layers in a caching environment. We examine caching and distribution strategies that use both versions and layers. We consider two cases: the request distribution for the videos is known a priori; and adaptive caching, for which the request distribution is unknown. Our analytical and simulation

A shorter version of this work has appeared in *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*, Vol. 2, pages 45–48, Lausanne, Switzerland, August 2002.

F. Hartanto
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
e-mail: hartanto@cse.cuhk.edu.hk

J. Kangasharju
Department of Computer Science, Darmstadt University of Technology, Germany
e-mail: jussi@tk.informatik.tu-darmstadt.de

M. Reisslein(✉)
Department of Electrical Engineering, WINTech Center, Arizona State University, Goldwater Center, MC 5706, Tempe, AZ 85287–5706, USA
e-mail: reisslein@asu.edu

K. Ross
Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201, USA
e-mail: ross@poly.edu

results indicate that mixed distribution/caching strategies provide the best overall performance.

Keywords Proxy caching · Streaming video · Layered video · Multi-version video

1 Introduction

Many analysts expect streaming stored video to be the dominant traffic type in the Internet in the upcoming years, dwarfing the bandwidth usage of other Internet applications. Driving this demand is the current deployment of residential broadband access technologies, such as cable modem and xDSL technologies.

Given that the Internet will soon be transporting vast quantities of video traffic, a major concern becomes the efficient distribution of the video data. As with Web objects, video data can be transported to the client in many different ways, including (a) directly from origin server to client; (b) through intermediate ISP caches; and (c) through content distribution networks (CDNs) such as the Akamai network.

In designing new strategies for distributing stored video over the Internet, we also must take into account that access to the Internet is highly heterogeneous [1, 10]. Today, Internet access includes 56 Kbps modem connections, 128 Kbps ISDN connections, shared-bandwidth cable modem connections, xDSL connections with downstream rates in the 100 Kbps to 6 Mbps range, and high-speed switched Ethernet connections at 10–100 Mbps. Because Internet access is heterogeneous, video content providers typically provide multiple quality levels, with each quality level having a different encoding rate.

Multiple quality levels can be created by encoding video into multiple versions, each version encoded at a different rate. However, multiple versions of the same video can cause large increases in the amount of storage. For example, for storing 1,000 videos, each video having an average length of 1 h and having an average encoded bit rate of 4 Mbps, the required storage is 1,800 GB of storage. If for each video there is a second lower-quality version at half the bit rate of the high-quality version, then we need an additional 900 GB of storage. When there are many versions, the additional required bandwidth can be yet much more.

Layered encoding (also known as hierarchical encoding) can also be used to create multiple quality levels. The storage requirements at a server for maintaining multiple layers is typically much less than maintaining the same number of versions. However, creating video layers generates additional bandwidth overhead [2, 9]. In particular, for the same quality level, layered encoding typically requires more transmission bandwidth than does a video version.

Given the presence of a caching and/or content distribution network infrastructure, and the need for multiple video quality levels, in this paper we compare distributing video versions to distributing video layers. We also examine mixed strategies consisting of both versions and layers.

Specifically, we consider a model in which all video content is encoded into two versions: a low-quality version and a high-quality version. All videos are also hierarchical encoded into a base layer and an enhancement layer. A proxy, representing an institutional cache or a server in a CDN, sits between the origin servers and the clients. Bandwidth between the proxy and the clients is assumed to be abundant.

However, bandwidth between the origin server and the proxy is a constrained resource, as well as is the storage capacity at the proxy. When the proxy receives a request for a video at a specific quality level, the proxy will directly satisfy the request if it has cached the appropriate version or layers; otherwise, if sufficient bandwidth is available between origin server and proxy, the origin server will stream the needed version or layer(s) to the client through the proxy.

We first consider the case when the request distribution for the videos is known. We consider three natural distribution strategies and develop an analytical performance methodology. We then consider the case when the request distribution is unknown. We propose three natural adaptive caching strategies and use simulation to compare their performance. Broadly speaking, we find that mixed strategies that use both versions and layers provide the most robust performance. Our model and methodology brings out a number of subtle issues that shed important insights on the distribution of multi-quality video in the Internet.

This paper is organized as follows. We end this section with an overview of related work. In Section 2 we present our model and establish some basic properties of optimal caching strategies. In Section 3 we consider the case when the request distribution is known. We develop an analytical methodology, which we use to study the performance of three natural caching strategies. In Section 4 we consider adaptive caching and again study three natural distribution schemes. We discuss video transcoding in the context of our distribution framework in Section 5 and summarize our findings in Section 6.

1.1 Related work

Decuetos et al. [4] also compared streaming of video versions to streaming of video layers. In particular, in a TCP-friendly context, they proposed prefetching and quality-level switching schemes for both pure versions and pure layers. The paper [4] focused on time-dependent streaming of a single video from origin server to client; it did not take into account an intermediate cache sitting between origin servers and clients.

Kim and Ammar [8] compared streaming layers and versions in a multicast context. Their results show that while layering may have a slight advantage in general, there are also situations where versions are to be preferred. However, their work is based on multicast and does not take into account any caching.

Kangasharju et al. [7] considered caching strategies for layered video. In particular, they formulated the problem as an optimization problem, showed that the optimization problem was intractable, and proposed and studied several natural heuristics. The paper [7] did not take into account multiple versions, and therefore did not compare caching layers, caching versions, and mixed strategies.

2 Model and notation

Figure 1 illustrates our architecture for video caching. Suppose there are M videos available; and all of them are stored on the origin servers. Popular videos are cached in a proxy server, which is located close to its client community.

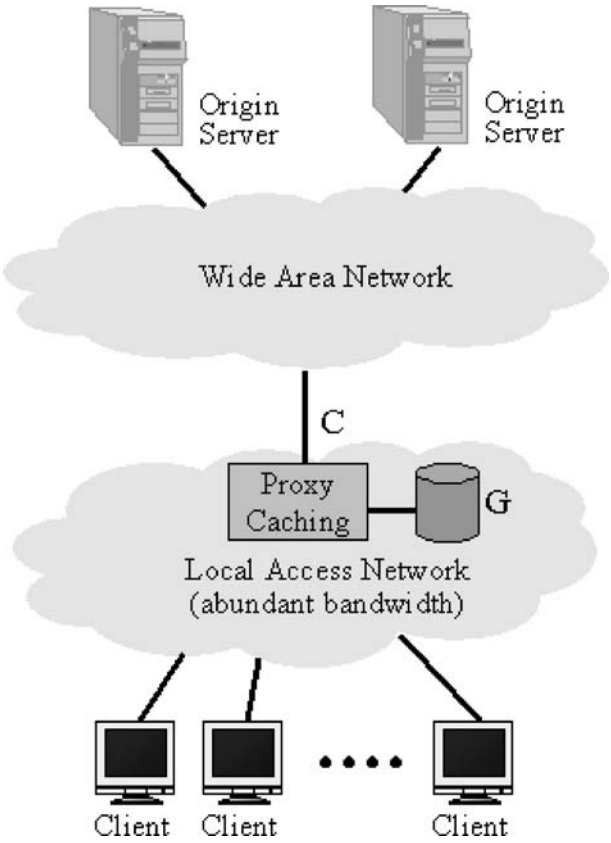


Fig. 1 Architecture for caching and streaming of adaptive video

2.1 Proxy server

The clients direct their requests to the proxy server; if the requested video (defined by type and quality) is in the proxy, then the video is streamed from the proxy to the client; if it is not in the proxy, the video is streamed from the origin server to the proxy, and then from the proxy to the client.

The proxy server is connected to the origin servers via a wide area network (e.g., the Internet). We model the bandwidth available for streaming from the origin servers to the proxy server as a bottleneck link of fixed capacity C (bit/s). The proxy is connected to the clients via a local access network, which could be a LAN running over Ethernet, or a residential access network using xDSL or HFC technologies. For the purposes of this study, we assume that there is abundant bandwidth for streaming from the proxy to the clients. We model the proxy server as having a storage capacity of G (bytes) and having infinite storage bandwidth (for reading from storage). Our focus in this study is on caching strategies that cache complete layers or versions of videos in the proxy. Our goal is to cache video layers or versions so as to maximize the number of supported streams.

2.2 Versions

Real Networks [12] and other video streaming technology companies today allow content providers to encode video into multiple quality versions. Video versions allow service and content providers to offer flexible streaming services to clients with vastly different access bandwidths and decoding capabilities. Clients with low-speed access will only be interested in the low-quality stream. Clients with LAN, cable-modem or ADSL access will be interested in high-quality streams.

Many content providers today store multiple versions of the same video on the origin server and stream the video version that is most appropriate on a user-to-user basis. This approach allows for flexible pricing structures. A content or service provider may offer the low-quality version for a standard charge and charge a premium for the high-quality version. Throughout this paper we shall assume that two quality levels are available for each video.

Although the approach of multiple versions offers greater service and pricing flexibility, it has major drawbacks. First, it requires more storage at the origin server than does the approach that makes only one quality level available. Second, if one quality version is cached in a proxy server, and there is a request for a different quality version, then the requested version must be fully streamed from the origin server, i.e., the cached version is of no use. And third, if both quality levels are cached in the proxy, then more storage is necessary than when only one version is used.

2.3 Layers

An alternative to using versions is to use layered (also known as hierarchical encoded) video. With layered encoding, each video object is encoded into a base layer and one or more enhancement layers. The base layer contains the most essential information, and the enhancement layers provide quality enhancements. A particular enhancement layer can be decoded only if all lower quality layers are available. Throughout this paper, we will assume that each video has been coded into two layers, a base layer and a single enhancement layer.

The storage requirements for the base and enhancement layer together are typically less than the requirements for the low-quality and high-quality versions together, for both the origin server and the proxy. Furthermore, if the base layer is cached in the proxy, and a client requests a high-quality version, then only the enhancement layer needs to be streamed from the origin server. Nevertheless, layered encoding has one major drawback, namely, encoding overhead. Typically, for the same high-quality level, the total rate of the base and enhancement layer combined is larger than the rate of the high-quality version. Also, for the same low-quality level, the rate of base layer is often larger than that of the low-quality version. This overhead impacts both transmission and storage resources.

In summary, in this study each video can be encoded into either versions or layers. For versions, we suppose that there are two possible versions, namely, a high-quality version and a low-quality version. For layers, we suppose that the video is encoded into two layers, namely, a base layer and an enhancement layer. Thus, each video has four objects associated with it: a low-quality version, a high-quality version, a base layer, and an enhancement layer. We denote these four objects by l, h, b, and e, respectively.

If $T(m)$ is the length of video m, $m = 1, \ldots, M$, in seconds and $r(m)$ is the encoding rate for one of the versions or layers in bit/s, then the corresponding storage requirement for the object is $S(m) = T(m) \times r(m)$ bit. Table 1 summarizes the notation we will use for the two versions and the two layers. We naturally assume that the rate of the high-quality version is larger than the rate of the low-quality version, i.e., $r_h(m) > r_l(m)$.

In order to compare the caching of layers and versions, we suppose throughout that the encodings are such that the visual quality of the base layer is the same as the visual quality of the low-quality version; and the video quality of the base and enhancement layer combined is the same as the high-quality version. However, due to encoding overhead to create layers, we do not assume that the layers and versions have the same encoding rates. Instead, we make the following three natural *Rate Assumptions* which are based on video encoding experiments [2, 9]:

1. Due to the overhead of layered encoding, the base layer has at least the same rate as the low-quality version, i.e., $r_b(m) = r_l(m) \times [1 + O_l(m)]$ where $O_l(m) \geq 0$ is the low-quality coding overhead.
2. Again due to the overhead of layered encoding, the base and enhancement layers together have at least the same rate as the high-quality version, i.e., $r_b(m) + r_e(m) = r_h(m) \times [1 + O_h(m)]$ where $O_h(m) \geq 0$ is the high-quality coding overhead.
3. The base and enhancement layers together have smaller rate than the two versions, i.e., $r_b(m) + r_e(m) < r_l(m) + r_h(m)$.

For any video, the proxy can contain objects made from versions and/or layers. However, we assume the *decoding constraint*, namely, that the proxy never caches the enhancement layer if the base layer is not cached. When a request arrives to the proxy for some low-quality video, the proxy can satisfy the request if it is currently storing either the low-quality version or the base layer of the video. Otherwise, the proxy must obtain either the low-quality version or the base layer from the origin server and relay the object to the requesting client. When a request arrives to the proxy for some high-quality video, the proxy can satisfy the request if it is currently storing either the high-quality version or if it is storing both the base and enhancement layers of the video. Otherwise, it must retrieve an object from the network to satisfy the request. If the proxy has stored the base layer, then the proxy can retrieve either the enhancement layer or the high-quality version.

Note that we do not consider the details of the layered codec used to create the layered videos. In our caching model each video object (layer or version) is always treated as an entity, i.e., either we have all of it or none of it. This applies to both objects that are cached and objects that are streamed from the origin server. Also, for simplicity, we assume that the users always watch the complete video without interruptions. For these reasons, we do not need to consider the details of the

Table 1 Notation: rates and storage requirements of layers and versions of video m, $m = 1, \ldots, M$

	Base layer	Enh. layer	Low quality	High quality
Encoding rate	$r_b(m)$	$r_e(m)$	$r_l(m)$	$r_h(m)$
Size	$S_b(m)$	$S_e(m)$	$S_l(m)$	$S_h(m)$

codec, but only the *aggregate effect* of the layered codec, i.e., how much overhead it introduces. How this overhead is distributed over the individual frames has no effect on the results in this model. As we present in Section 3, a client specifies a target quality (low or high) and the request is either accepted or blocked. If the request is accepted, then the *entire video* is delivered at the requested target quality, with absolutely no quality changes.

2.4 Basic properties

For a given video, there are four cachable objects: the low-quality version (l), the high-quality version (h), the base layer (b), and the enhancement layer (e). Thus for any given video, there are $2^4 = 16$ different combinations of objects that can be put in the cache, including putting no object in the cache. This is a daunting number of combinations to analyze. Recall that our goal is to maximize the number of supported streams. Fortunately, without loss of generality, we may restrict ourselves to only five of the combinations:

Theorem 1 *There is an optimal caching configuration such that for each video one of the following five object combinations is used:* $\emptyset$, $\{l\}$, $\{h\}$, $\{b\}$, *or* $\{b, e\}$. *In other words, for each given video we either cache just the low-quality version, just the high-quality version, just the base layer, the base and enhancement layers together, or no objects at all.*

Proof Because of the decoding constraint for layered video, we can rule out all combinations that include e but not b.

Now consider $\{b, h\}$. Note that Rate Assumptions 3 and 1 together imply that $r_{\mathrm{h}}(m) > r_{\mathrm{e}}(m)$. Hence $r_{\mathrm{b}}(m) + r_{\mathrm{h}}(m) > r_{\mathrm{b}}(m) + r_{\mathrm{e}}(m)$. It follows from this last expression that we can replace the combination $\{b, h\}$ with $\{b, e\}$ and use less storage while still satisfying all requests at the proxy for the video. Thus we can rule out $\{b, h\}$.

Now consider $\{b, l\}$, $\{b, l, e\}$, $\{b, l, h\}$, $\{b, l, h, e\}$. By caching the base layer, we satisfy all low-quality requests and we partially satisfy higher quality requests (only need to get enhancement layer from network). If we additionally cache the low-quality version, we take up more storage and we do not satisfy more requests for low-quality video. Combining this observation with $r_{\mathrm{h}}(m) > r_{\mathrm{e}}(m)$ implies that if we cache the base layer, then there is no need to also cache the low-quality layer. Thus we can rule out all these four cases.

Now consider $\{l, h\}$. This combination will satisfy all requests at the proxy. However, the combination $\{b, e\}$ also satisfies all requests and, by Rate Assumption 3, takes less storage. Thus, we can rule out $\{l, h\}$.

Finally, we can also rule out $\{b, e, h\}$ since the combination $\{b, e\}$ also satisfies all requests but takes less storage. ■

As a corollary to the above theorem, for any given video we use either versions or layers but not both.

Theorem 1 generalizes to a scenario where we have three levels of quality without any need for additional rate assumptions. However, for scenarios with four or more levels of quality, we will need additional information about the details of the video

codec and the encoding rates of the different objects. This observation can be explained as follows. Say we have four quality levels. The layers have rates r_{l_1}, r_{l_2}, r_{l_3}, and r_{l_4}, and the corresponding versions have rates r_{v_1}, r_{v_2}, r_{v_3}, and r_{v_4}. Suppose that the cache contains the first two layers (l_1 and l_2) and the highest quality version, v_4. This clearly would violate Theorem 1, since we have both layers and versions for the same video. Making similar basic rate assumptions as for two layers, we know that $r_{l_1} + r_{l_2} < r_{v_1} + r_{v_2}$, hence we cannot replace the lower layers with the corresponding versions. To replace v_4 with layers, we would need to get both l_3 and l_4 into the cache. From the same basic rate assumptions we know that $r_{l_4} < r_{v_4}$, however we do *not* know whether $r_{l_3} + r_{l_4} < r_{v_4}$ would hold. For this, we would need additional information about how the codec distributes the video data into the layers.

If the codec puts the bulk of the data into the lower layers, then the higher layers are likely to be small which would mean that Theorem 1 would hold in the general case. However, if the lower layers are small and the higher layers contain a large amount of data, then the question of whether Theorem 1 holds would remain open. If we know how the codec distributes the data, then the actual verification of Theorem 1 is simple, since we only need to compare the rates of the higher layers to the rates of the higher quality versions. In a scenario with a large number of layers this could mean a large number of possible combinations which we would need to verify; however, the actual verification would be a simple operation which could easily be performed.

Motivated by the above theorem, in the following sections we will propose and examine strategies for caching layer and version objects. But it is also useful to make a few additional *Observations* about extreme cases:

1. For a given video if all (or "nearly all") requests are for the low-quality version (and none or "nearly none" are for the high-quality version), then we would either cache the low-quality version or cache no objects for that video, i.e., as object combination we would use either $\{l\}$ or $\emptyset$.
2. Similarly, if for a given video if all (or "nearly all") requests are for the high-quality version, we would use either $\{h\}$ or $\emptyset$.
3. If there is no overhead for layered encoding, that is, if $O_l(m) = O_h(m) = 0$, then for video m we would only use layers; in particular, we would use either $\emptyset$, $\{b\}$, or $\{b, e\}$.

However, when (*a*) there is layering overhead, and (*b*) request rates for low- and high-quality versions are both significant, then it is not obvious whether we should use versions or layers; furthermore, for some videos it may be preferable to use versions whereas for others it may be preferable to use layers.

3 Known request distribution

We start by modeling the steady-state cache performance using a static caching model. With this model, we assume that the request pattern is known a priori and does not change dynamically. Suppose that there are M videos. Suppose that requests for video streams arrive according to a Poisson process with rate λ (requests/hour). Let j denote the requested quality level with $j = 0$ indicating a request for a low quality video, and $j = 1$ indicating a request for a high quality video. Let

$p(j,m),\ \ j=0,\ 1;\ \ m=1,\ldots,M$, denote the probability that a given request is for the j-quality stream of video m. As a proper mass distribution the $p(j,m)$'s satisfy $\sum_{m=1}^{M}\sum_{j=0}^{1} p(j,m)=1$.

The corollary to Theorem 1 suggests three caching strategies, namely:

1. Pure version caching, where we cache only video versions.
2. Pure layer caching, where we cache only video layers.
3. Mixed caching, where we cache layers for some videos and versions for others.

For all three caching strategies we first order the request probabilities $p(j,m)$, $j=0,1$; $m=1,\ldots,M$, in decreasing order. We then fill the cache by considering the objects (j,m) that are the most requested. First, we put the object (j,m) with the largest request probability $p(j,m)$ into the cache. Next, we cache the object (j,m) with the next largest probability $p(j,m)$, and so on. If at some point (as the cache fills up) the object needed to satisfy the request with the next largest request probability does not fit into the remaining cache space, we skip this object and try to cache the objects with the next largest request probabilities.

With *pure version caching* we cache only versions of the videos. We cache the high quality version of video m if the next largest probability $p(j,m)$ is for the high quality stream of video m (i.e., $j=1$). If the next largest probability is for the low quality stream of video m, then we cache the low quality version of video m. Note that with pure version caching we may end up caching both high and low quality versions of the same video (which we know from Theorem 1 is sub-optimal).

With *pure layer caching* we cache only video layers. If the next largest request probability $p(j,m)$ is for the low quality stream of video m (i.e., $j=0$), then we cache the base layer of video m. On the other hand, if the next largest probability is for the high quality stream of video m (i.e., $j=1$), then we cache both base and enhancement layer of video m. If the base layer has already been cached, i.e., if $p(0,m)>p(1,m)$, then we need to cache the enhancement layer only. Due to the decoding constraint, we never cache the enhancement layer of a given video without caching the corresponding base layer.

With *mixed caching* we cache the high quality version of video m if the next largest $p(j,m)$ is for the high quality stream of video m and no other object of the video has been cached. On the other hand, if the next largest probability is for the low quality stream of video m and no other object of the video has been cached, then we (a) cache the low quality version of video m if $r_b(m)>r_l(m)$, and (b) cache the base layer of video m if $r_b(m)=r_l(m)$. However, if we have already cached the low (or high) quality version of a given video and the next largest probability is for a different quality of the video, then we replace the low (or high) quality version of the video with the base and enhancement layer of the video.

3.1 Video caching model

In this section we develop an analytical model for the caching and streaming of video layers and versions. We derive expressions for the blocking probability of a client request and the long run rate at which client requests are satisfied. To keep track of the objects in the cache we introduce a vector of cache indicators $\boldsymbol{c}=(c_1,c_2,\ldots,c_M)$, with $c_m=\{\emptyset\},\ \{l\},\ \{h\},\ \{l,\ h\},\ \{b\},$ or $\{b,\ e\}$, for $m=1,\ldots,M$. c_m indicates whether no object, the low-quality version, the high-quality version,

both the low- and high-quality version, the base layer, or the base layer together with the enhancement layer is cached for video m. (We allow for $c_m = \{l,\ h\}$ to accommodate pure version caching in our model; note, however, that by Theorem 1 it is sub-optimal to cache both the low- and high-quality version for a given video m.) In our model we focus on the bottleneck link of capacity C, that connects the proxy server to the origin servers. We model this link as a stochastic knapsack [13]. Let $b_{c_m}(j,m),\ \ j=0,\ 1,\ \ m=1,\ldots,M$, denote the link capacity required for satisfying a request for a j-quality stream of video m, given that the object(s) c_m are cached for video m. Table 2 gives the $b_{c_m}(j,m)$'s for all possible combinations of c_m and j. We assume that the lower rate versions are streamed over the bottleneck link whenever a request cannot be satisfied by the cache; except in the case where the base layer is cached and the high-quality stream is requested, in that case we stream the enhancement layer. Without loss of generality we assume that C and all $b_{c_m}(j,m)$'s are positive integers. Let $\boldsymbol{b_c} = (b_{c_m}(j,m)),\ \ j=0,\ 1,\ \ m=1,\ldots,M$, be the vector of the bandwidth requirements of the requests. Note that this vector has $2M$ elements. Throughout we assume that the client watches the entire stream without interruption, thus the bandwidth $b_{c_m}(j,m)$ is occupied for $T(m)$ seconds. Let $\boldsymbol{n} = (n(j,m)),\ j=0,\ 1, m=1,\ldots,M$, be the vector of the numbers of ongoing j-quality streams of video m. The $n(j,m)$'s are non-negative integers. Let $\mathcal{S}_{\boldsymbol{c}} = \{\boldsymbol{n} : \boldsymbol{b_c} \times \boldsymbol{n} \le C\}$ be the state space of the stochastic knapsack model of the bottleneck link, where $\boldsymbol{b_c} \times \boldsymbol{n} = \sum_{m=1}^{M}\sum_{j=0}^{1} b_{c_m}(j,m) \times n(j,m)$. Furthermore, let $\mathcal{S}_{\boldsymbol{c}}(j,m)$ be the subset of states in which the knapsack (i.e., the bottleneck link) admits a stream with the bandwidth requirement $b_{c_m}(j,m)$. We have $\mathcal{S}_{\boldsymbol{c}}(j,m) = \{\boldsymbol{n} \in \mathcal{S}_{\boldsymbol{c}} : \boldsymbol{b_c} \times \boldsymbol{n} \le C - b_{c_m}(j,m)\}$. The blocking probabilities can be explicitly expressed as

$$B_{\boldsymbol{c}}(j,m) = 1 - \frac{\sum_{\boldsymbol{n}\in\mathcal{S}_{\boldsymbol{c}(j,m)}} \prod_{m=1}^{M}\prod_{j=0}^{1}(\rho(j,m))^{n(j,m)}/(n(j,m))!}{\sum_{\boldsymbol{n}\in\mathcal{S}_{\boldsymbol{c}}} \prod_{m=1}^{M}\prod_{j=0}^{1}(\rho(j,m))^{n(j,m)}/(n(j,m))!}, \tag{1}$$

where $\rho(j,m) = \lambda p(j,m) T(m)$ is the load offered by requests for j-quality streams of video m. These blocking probabilities can be efficiently calculated using the recursive Kaufman–Roberts algorithm [13, p. 23]. The expected blocking probability of a client's request is given by

$$B(\boldsymbol{c}) = \sum_{m=1}^{M}\sum_{j=0}^{1} p(j,m) B_{\boldsymbol{c}}(j,m).$$

Table 2 Bandwidth requirement for streaming j-quality stream of video m given cache configuration $b_{c_m}(j,m)$

$b_{c_m}(j,m)$	$c_m = \{\emptyset\}$	$c_m = \{l\}$	$c_m = \{h\}$	$c_m = \{l,\ h\}$	$c_m = \{b\}$	$c_m = \{b,\ e\}$
$j=0$	$r_{\mathrm{l}}(m)$	0	$r_{\mathrm{l}}(m)$	0	0	0
$j=1$	$r_{\mathrm{h}}(m)$	$r_{\mathrm{h}}(m)$	0	0	$r_{\mathrm{e}}(m)$	0

The long run throughput, i.e., the long run rate at which client requests are satisfied is given by

$$\mathrm{TH}(\boldsymbol{c}) = \lambda \times \sum_{m=1}^{M} \sum_{j=0}^{1} p(j,m)(1 - B_{\boldsymbol{c}}(j,m)).$$

We define the normalized throughput $\mathrm{TH}_n(\boldsymbol{c})$ as the ratio of the rate of satisfied requests to the total request arrival rate, i.e., $\mathrm{TH}_n(\boldsymbol{c}) = \mathrm{TH}(\boldsymbol{c})/\lambda$.

3.2 Numerical results

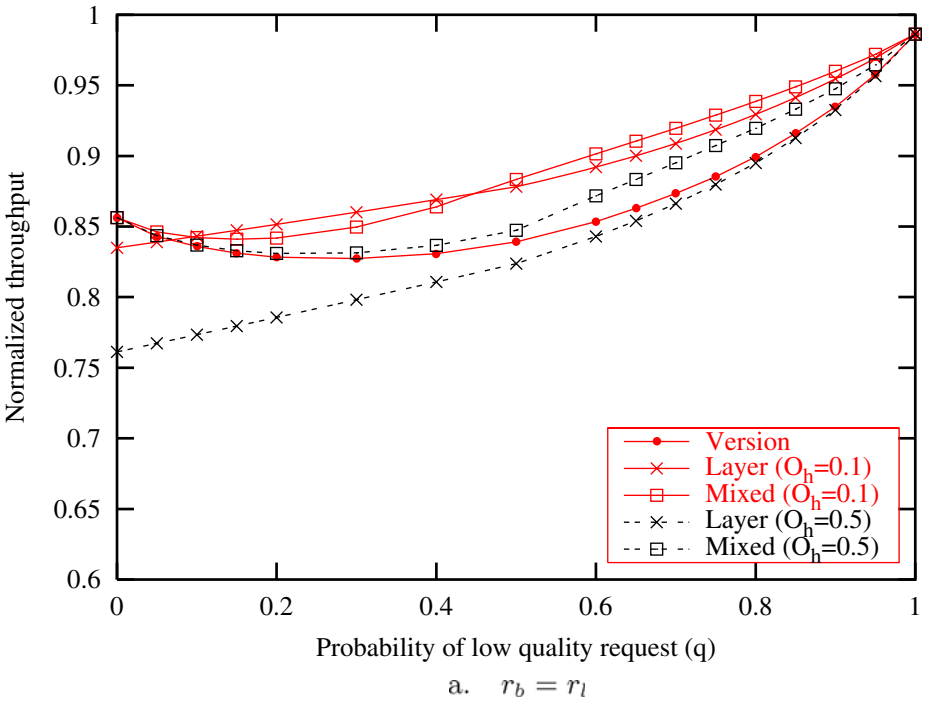
We assume that there are M = 1,000 different videos. For a given video m we generate the version and layer rates as follows. The rate of the high quality version $r_{\mathrm{h}}(m)$ is drawn randomly from a uniform distribution between 2 and 6 Mbps with a granularity of 0.1 Mbps and an average of 4 Mbps. The rate of the low quality version $r_{\mathrm{l}}(m)$ is uniformly drawn between $0.5 \times r_{\mathrm{h}}(m)$ and $0.7 \times r_{\mathrm{h}}(m)$ with an average of $0.6 \times r_{\mathrm{h}}(m)$. The length of the video $T(m)$ is drawn from an exponential distribution with an average length of 1 h.

We assume that the aggregate rate for the layered video has an overhead $O_{\mathrm{h}}(m)$ over the high quality version, i.e., $r_{\mathrm{b}}(m) + r_{\mathrm{e}}(m) = [1 + O_{\mathrm{h}}(m)] \times r_{\mathrm{h}}(m)$. We consider two cases: (a) $r_{\mathrm{b}}(m) = r_{\mathrm{l}}(m)$, and (b) $r_{\mathrm{b}}(m) > r_{\mathrm{l}}(m)$, in this case we vary $r_{\mathrm{b}}(m)$ between $r_{\mathrm{l}}(m)$ and $[1 + O_{\mathrm{h}}(m)] \times r_{\mathrm{l}}(m)$. With $r_{\mathrm{b}}(m)$ fixed, the rate of the enhancement layer $r_{\mathrm{e}}(m)$ is then computed as $r_{\mathrm{e}}(m) = [1 + O_{\mathrm{h}}(m)] \times r_{\mathrm{h}}(m) - r_{\mathrm{b}}(m)$.

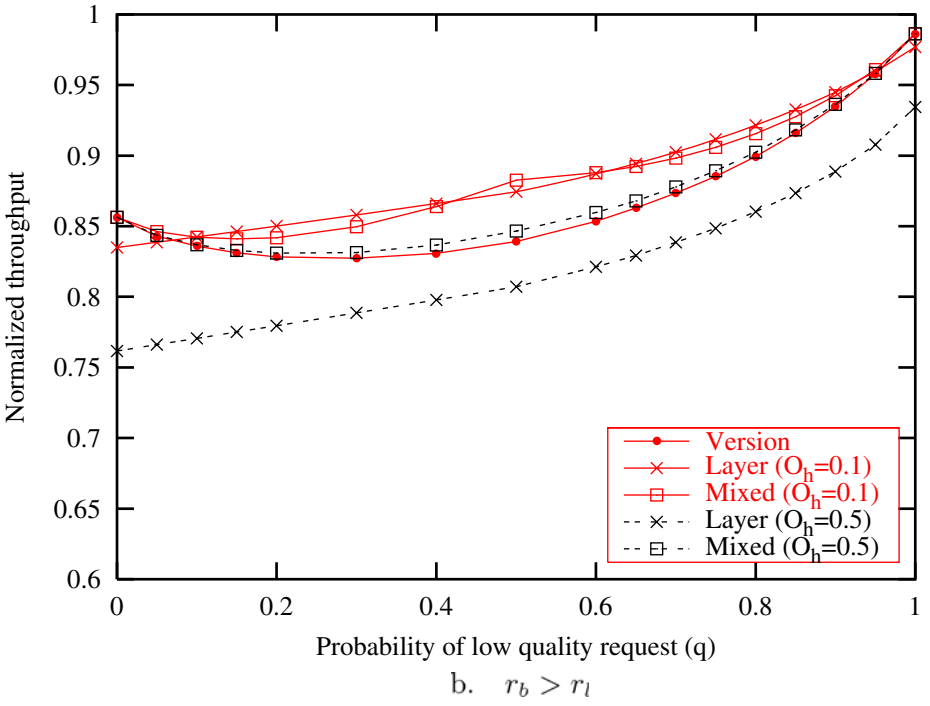
The $p(j,m)$s are determined as follows. Let $p_m,\ m = 1, \ldots, M$, denote the probability that a given client request is for video m (irrespective of whether the request is for the low quality stream or the high quality stream of the video). We draw the p_ms from a Zipf distribution with parameter $\zeta = 1$. Let q denote the probability that the request for a given video is for the low quality stream of the video. We fix q as a system parameter in our numerical analysis. We set $p(0,m) = q \times p_m$ and $p(1,m) = (1 - q) \times p_m$. Client requests arrive according to a Poisson process. The average request arrival rate is λ = 270 requests/hour, chosen to give a blocking probability of 2% when $q = 1.0$.

The cache size is set to G = 200 GB and the link capacity is C = 150 Mbps. For a given realization of the layer and version rates $(r_{\mathrm{l}}(m),\ r_{\mathrm{h}}(m),\ r_{\mathrm{b}}(m),\ r_{\mathrm{e}}(m))$ as well as video lengths $T(m),\ m = 1, \ldots, M$, we apply the three outlined caching strategies to obtain the cache indicators $c_m,\ m = 1, \ldots, M$. With these cache indicators we calculate the normalized throughput using the stochastic knapsack analysis introduced in the previous section. We run many independent replications of this procedure to obtain confidence intervals for the normalized throughput. For every independent replication we draw a new independent set of layer and version rates and video lengths. We repeat this procedure until the 95% confidence interval of the normalized throughput is less than 1% of the corresponding sample mean.

In figure 2 we plot the normalized throughput as a function of the probability of a low quality request q. The results show that if no overhead is incurred in generating layered videos (i.e., $O_{\mathrm{h}} = 0$), then pure layer caching is the best strategy as suggested by Observation 3 above. Caching layers is also favorable when the requests are non-homogeneous ($0.1 < q < 1$) and the overhead is low. We see that the throughput for pure layer caching increases monotonically as more requests are for low quality videos and decreases with increasing overhead. The throughput for



a. $r_b = r_l$



b. $r_b > r_l$

Fig. 2 Static caching scenario with varying probability of low quality requests

pure layer caching is strongly affected when the base layer includes overhead (i.e., $r_b > r_l$). This can be considered as the worst case and therefore, we always assume $r_b > r_l$ in future plots.

Pure version caching is only favorable in case of homogeneous request quality, i.e., all requests are either for low quality ($q = 0$) or for high quality streams ($q = 1$). The largest throughput is achieved if all requests are for low quality streams. This is expected because in this scenario more videos are cached and hence the cache hit rate is higher compared to a scenario where all requests are for high quality streams. The throughput is lowest when the requests are non-homogeneous as sometimes we need to cache both the low- and the high-quality version.

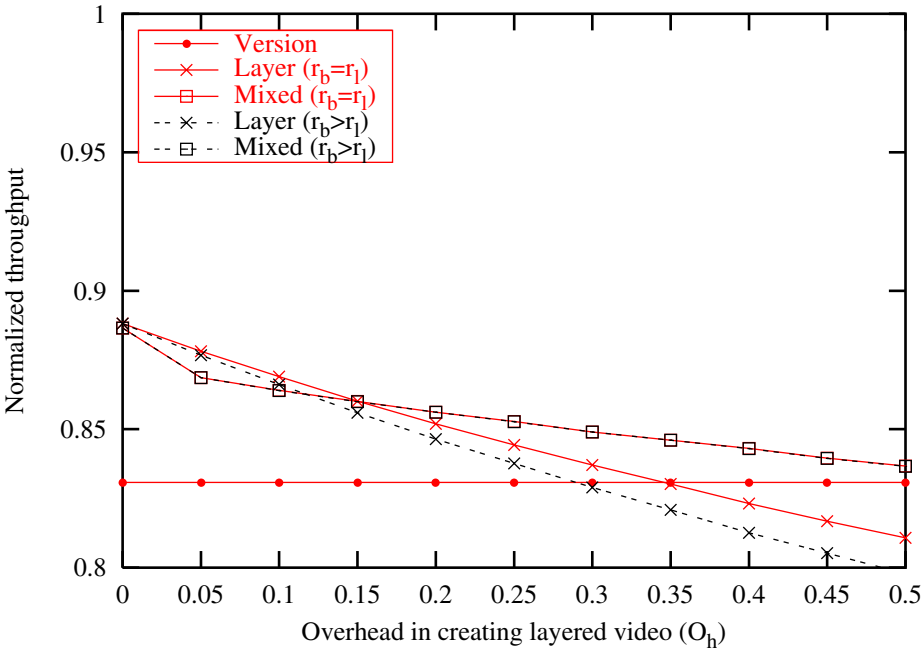
The results indicate that mixed caching strikes a good balance between pure layer caching and pure version caching for all cases and offers the best overall performance. It performs as well as pure layer caching when the overhead is zero and as well as pure version caching when $O_h = 0.5$. Since the smallest r_l is $0.5 \times r_h$, $O_h = 0.5$ is the largest overhead incurred in creating layered video while meeting Rate Assumption 3.

Figure 3 gives the normalized throughput as a function of the overhead O_h of layered encoding. We can clearly see that mixed caching gives better performance than pure version caching and pure layer caching for the range of overhead. Its performance is less sensitive to the overhead than pure layer caching.

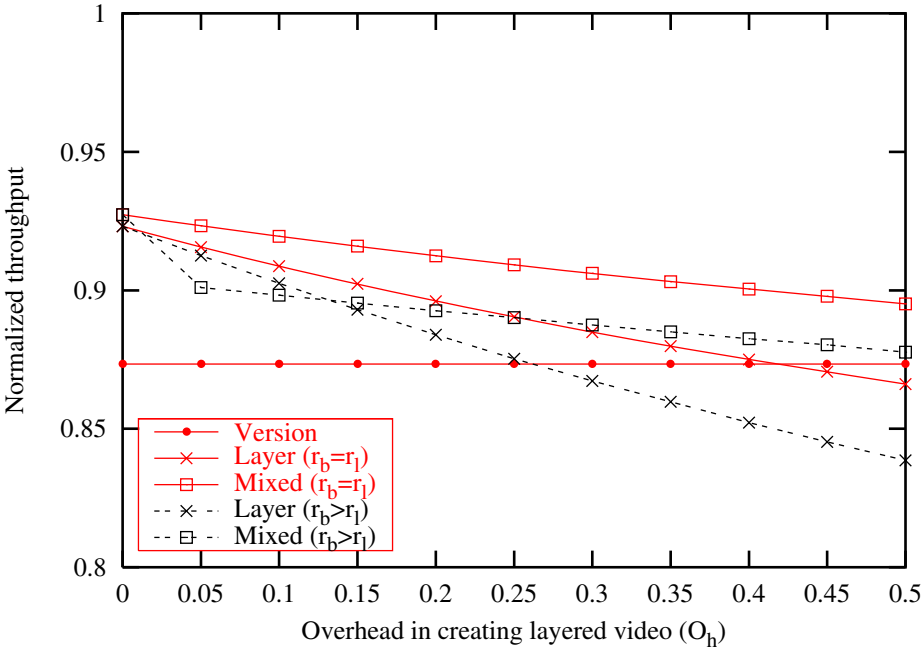
The superiority of mixed caching is independent of the cache size and the link capacity. In figure 4 we plot the normalized throughput as a function of the cache size G and the link capacity C. The cache size is chosen between $G = 45$ and 900 GB or between 2.5 and 50% of the total video data. Given the average video length T_{avg} (in seconds), the average rate of a video r_{avg} (in bit/s), and the client request rate λ (in requests/second), we would need on average $C = T_{avg} \times r_{avg} \times \lambda$ Mbps of bandwidth to stream all the requested videos. We varied the link capacity between $C = 10$ and 160 Mbps or between 1 and 16% of the total requested video bit rate. Both figures show that in all cases mixed caching offers the best overall performance. It shows that mixed caching gives similar performance to pure layer caching for small overhead and similar performance to the pure version caching for $O_h = 0.5$. In summary, the results for the static caching model demonstrate that a mixed caching strategy can strike a good balance between pure layer caching and pure version caching.

4 Adaptive caching

With the static caching model, the request distribution is assumed to be known beforehand. However, in practice, the actual request distribution may not be known. When the distribution is unknown, we need to make caching and replacement decisions on-the-fly. Moreover, in most video distribution systems, new videos are being continually released. As the video popularities change, providers replace the least popular videos in their systems with new videos. In this section we will consider adaptively caching and replacing videos when the request distribution is unknown and new videos are being continuously released. We will compare the performance differences of static caching (with known distributions) and adaptive caching, and identify the factors causing the differences. We will also investigate whether the basic observations of Section 2 still apply.



a. $q = 0.4$



b. $q = 0.7$

Fig. 3 Static caching with varying amount of overhead of layered encoding

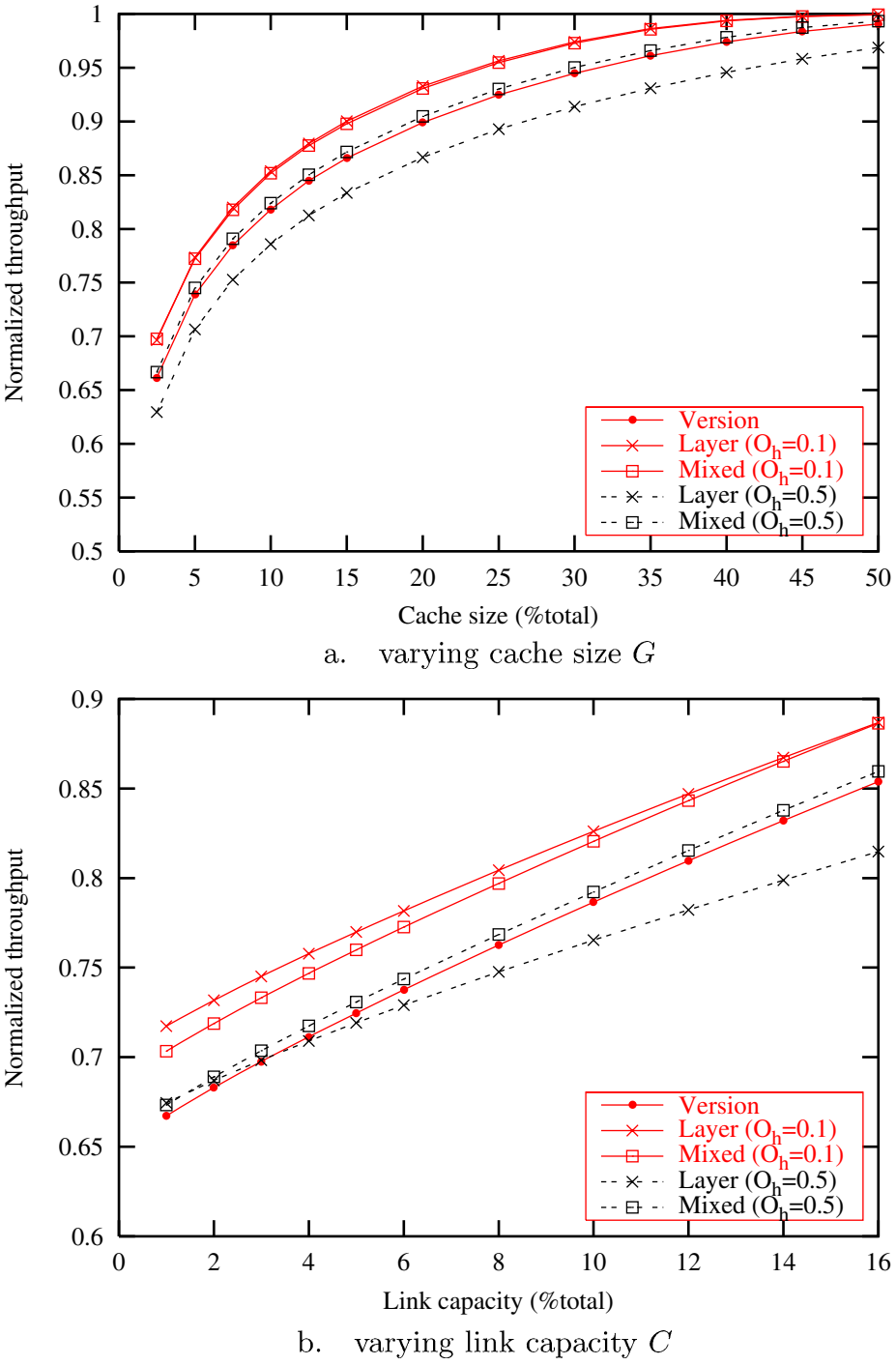


Fig. 4 Static caching with varying cache size G and link capacity C ($q = 0.4$)

In order to allow for direct comparison with the static caching scenario, we model the dynamic request distribution as follows. We start with the same 1,000 videos and generate requests in a similar fashion as in the static caching scenario. However, in this adaptive model, we assume that a fresh set of videos is made available periodically and the least popular videos are replaced by this new set of videos. We assume that 1–50 new videos are released every week and that the exact number of new videos is uniformly distributed between 1 and 50. The characteristics of these new videos follow the same distribution as used in Section 3.2. Once the least popular videos are replaced by these new videos, the popularity of all videos in the system are re-shuffled and requests are generated based on the new popularity distribution. Upon re-shuffling, we also evict all currently cached objects from the videos that have been replaced. If at least one stream is currently using the objects, then we remove the objects as soon as the ongoing stream(s) finishes.

Now, we explain the caching strategies. We start with an empty cache and cache the layer or version of a video as it is requested and streamed to the client. If the cache is full, then we replace the video in the cache following a *least recently used (LRU)* replacement strategy. We replace videos in the cache until enough space is obtained. In all strategies, we do not replace a video object from the cache if the object is currently being used for streaming the video. In the following we describe the three caching strategies identified in Section 3.

With *pure version caching*, we cache the high-quality version if the high quality video is requested and that version is not in the cache regardless of whether we have the low version in the cache or not. Likewise, we cache the low-quality version if the low quality video is requested and that version is not in the cache. Again, we do it regardless of whether we have the high-quality version in the cache or not. Therefore, we can have both high- and low-quality versions in the cache (which we know from Theorem 1 is suboptimal).

With *pure layer caching*, we cache both base and enhancement layer if the high quality video is requested and the video is not in the cache. If we already have the base layer, then we only stream the enhancement layer from the origin server and cache it. We stream the base layer from the origin server and cache it if the request is for low quality video and the base layer is not cached. During replacement, we remove the enhancement layer before the base layer.

With *mixed caching*, we have a similar objective as in the static caching model. We basically want to reduce the resource usage by mixing layers and versions in the cache. Here, we consider two simple heuristics to illustrate our findings.

The *first heuristic* corresponds exactly to mixed caching in the static model. Its objective is to replace the caching of both high- and low-quality version of a video with the layers of the video since they use less resources as $r_b(m) + r_e(m) < r_h(m) + r_l(m)$. The caching proceeds as follows. For the first request we stream and cache the version of the video. So, if the request is for high-quality video, we stream and cache the high-quality version of the video. If the request is for the low-quality video, then we stream and cache the low-quality version if $r_b(m) > r_l(m)$, or the base layer if $r_b(m) = r_l(m)$. If there is a second request for a different quality level of the same video, then we try to satisfy the request with layers and remove the version from the cache. Otherwise, we proceed with pure version caching. In replacing the version by layers, we reject the request and keep the version if we do not have enough link capacity to stream the layers. Moreover, since we do not want to interrupt ongoing streams, we cannot remove the version if it is being used, but we still cache the layers.

Once we have cached the layers, the version will be removed as soon as the ongoing streams using that version are terminated. If both the base and enhancement layers of a video are removed from the cache (by LRU replacement), then we will start again with the streaming and caching of versions for the next request. The motivation is that if a video object can be removed from the cache, then the video object is probably not very popular. So, it is better to start again with versions.

The *second heuristic* is similar to the first heuristic, except that for the first request we stream the version of the video but we do *not* cache it. So, if the request is for high quality video, then we stream the high-quality version of the video but do not cache it, and if it is for low-quality video, then we stream the low-quality version of the video but do not cache it. If there is a second request for the same video then we cache the layers of the video. If the second request is for high-quality video then we stream and cache both base and enhancement layers. If it is for low-quality video, then we stream and cache the base layer only. In this way, we stream versions but never cache them. Instead, we cache only layers. The motivation of this heuristic is to avoid caching objects for videos which are requested once only. Moreover, for videos which are requested more than once, caching video layers can serve requests of different quality while using less resources. This caching strategy requires the proxy to keep track of videos that have been previously streamed. If all layers of a video are removed from the cache, then we will start again, streaming but not caching versions, and caching layers upon second requests.

4.1 Numerical results

We now present simulation results for adaptive caching. We use the same distributions for the layer and version rates as well as the video lengths as were used in Section 3.2. While we evaluated the normalized throughput with the stochastic knapsack analysis in Section 3.2, we now obtain the normalized throughput from simulations of the cache operation. We use sequential simulation [6] to stop a simulation run automatically once the 95% confidence interval is reached or the simulation has run for 10^8 s. We then repeat the simulation by using different seeds. This ensures a different mixture of videos and hence cache composition. The final results are obtained by averaging the values from all runs. The simulation runs are repeated until the final results with 95% confidence intervals across different video mixtures are reached.

Figure 5 gives the normalized throughput as a function of the probability of a low quality request q. The figure shows that pure version caching is only favorable in case of homogeneous requests. For heterogeneous requests, pure layer caching offers better performance than pure version caching, especially when the layering overhead is low and no overhead is incurred in creating the base layer. As with the static model, we see that mixed caching—using both heuristics 1 and 2—provides a good balance between pure layer and pure version caching. It performs better than pure layer caching for small overhead and as well as pure version caching for large overhead. We also observe that heuristic 2 gives excellent results for a small layered encoding overhead. Note that heuristic 2 can be considered a variation of pure layer caching where we require to see two requests before caching layers of a video. Throughout, heuristic 2 performs much better than pure layer caching. This demonstrates the importance of weeding out the one-timer requests.

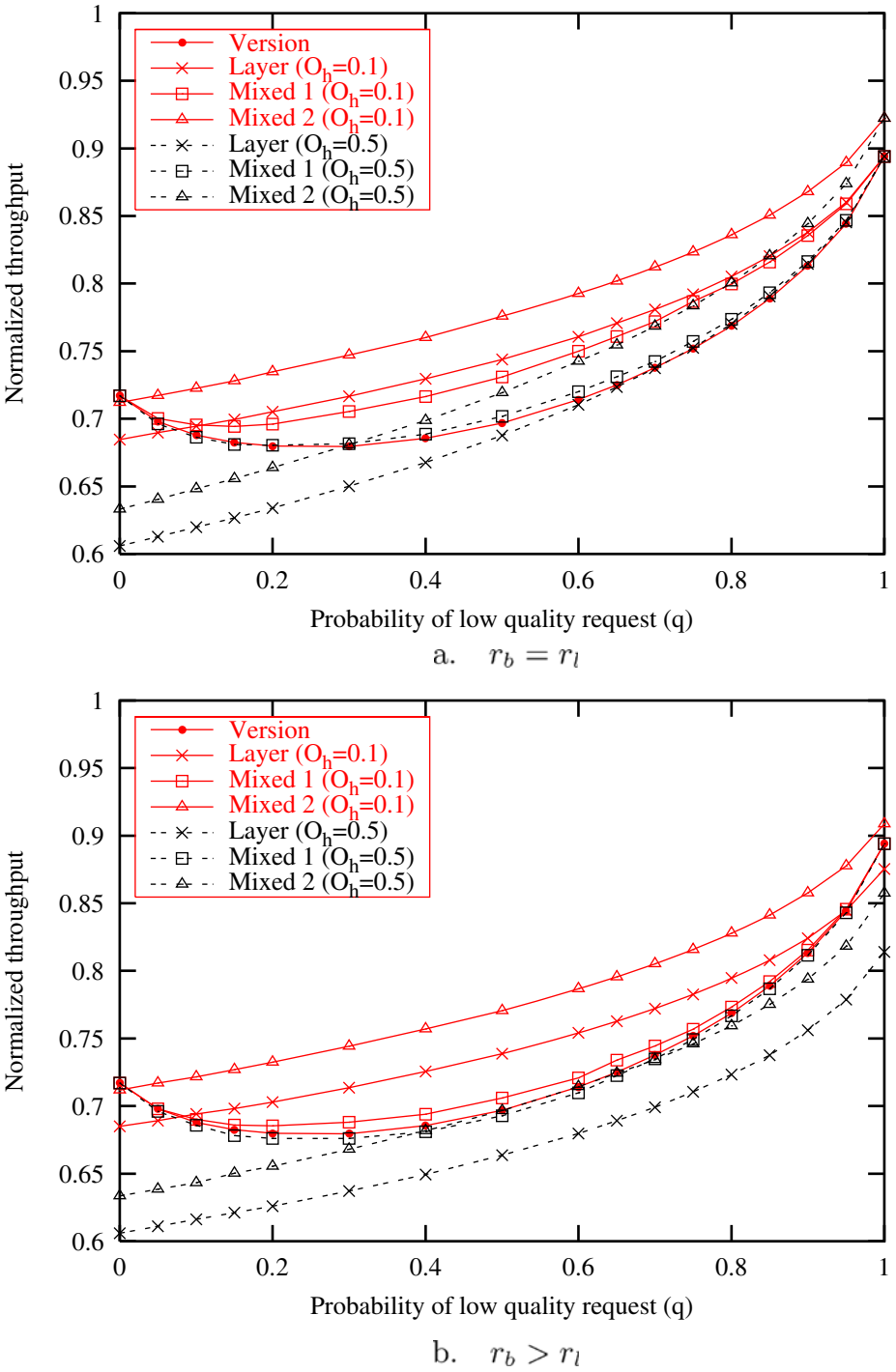


Fig. 5 Adaptive caching for varying probability of low quality request

Comparing the plots with figure 2 we notice that in general the throughput for adaptive caching is smaller than the throughput for static caching. This is mainly because the request pattern is not known a priori in the adaptive caching model. In adaptive caching, videos are (a) cached as requests arrive, and (b) evicted from the cache when there is not enough space for new video objects. Thus, the order of the request arrivals has a strong impact on the cache composition, whereas the cache composition is exclusively based on the stream popularities in the static caching model. The difference in performance between static caching and adaptive caching widens as the average request arrival rate λ increases, as is illustrated in figure 6. This can be explained as follows. Consider a cache with a large request arrival rate and suppose that a "mistake" has been made by caching a moderately popular object. With a large request arrival rate even a moderately popular object could receive enough requests to have continuously one or more ongoing streams. These ongoing streams, however, keep the object in the cache and prevent more popular objects (which would have been cached in the known request distribution scenario) from entering the cache.

Figure 7 gives the normalized throughput as a function of the amount of overhead incurred in layered encoding. We observe that heuristic 2 offers the best overall performance. However, similar to pure layer caching, it is highly sensitive to the overhead. On the other hand, heuristic 1 behaves similar to pure version caching and is hence less sensitive to the overhead.

The effects of varying the cache size and link capacity on the normalized throughput are shown in figure 8. Comparing figures 8a and b with figures 2a and b we see

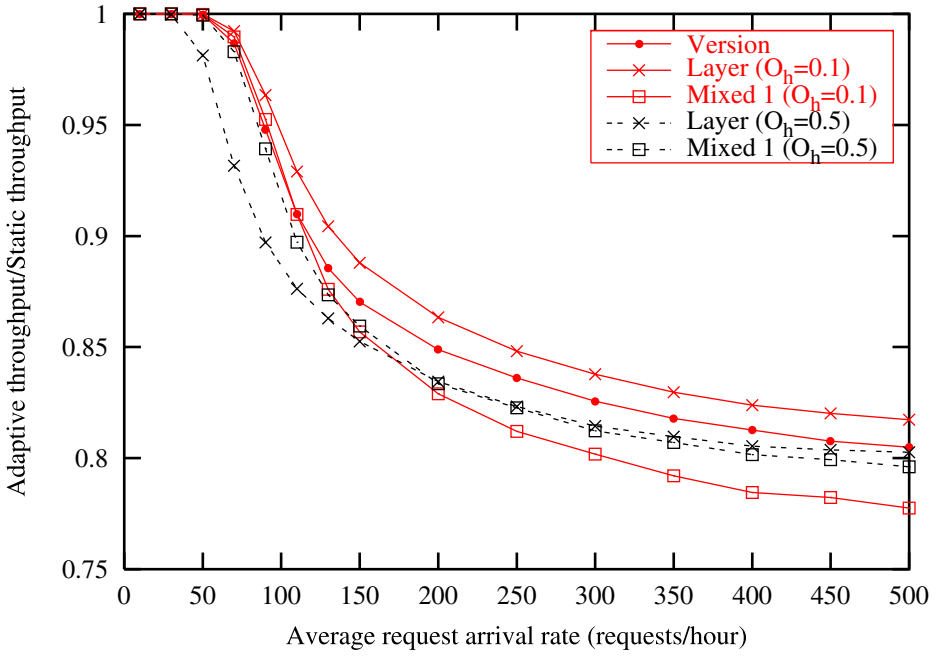


Fig. 6 Differences in throughput between adaptive and static caching ($q = 0.4$)

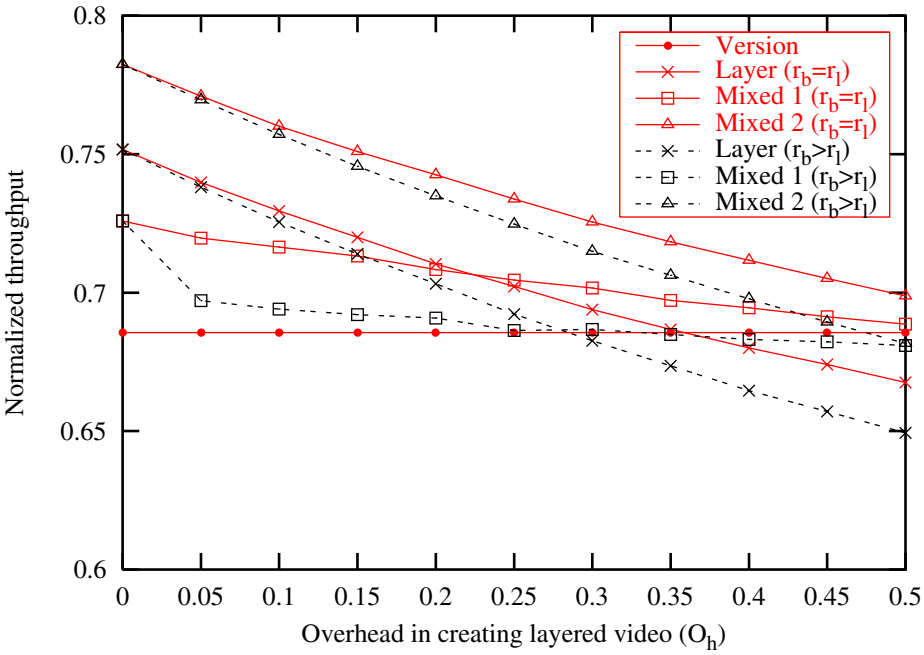


Fig. 7 Adaptive caching with varying amount of overhead for layered encoding

that the normalized throughput in the adaptive caching model does not grow as fast as in the static caching model for small cache sizes and small link capacities. This is again due to the fact that without a priori knowledge of the request distribution the order of the request arrivals has a strong impact on the cache composition. Also, moderately popular objects tend to keep the few extremely popular objects from being cached. Note again that by weeding out one-timer requests, heuristic 2 achieves a higher throughput than the other strategies.

5 Transcoding

In this section we discuss transcoding (see for instance [14] and references therein), that is, creating lower quality versions of high quality versions that are already present in the cache. Transcoding has been studied in the context of image caching [5, 11], but its applicability to video caching is unknown because of the much higher resource requirements needed to re-encode the video objects. We consider two possibilities for transcoding: real-time, or online transcoding, where we can create the lower quality version from the high quality version in real-time and stream it to a client, and non-real-time, or offline transcoding, where the transcoding operation takes longer than the duration of the video.

The online transcoding scenario is particularly attractive, because it would allow us to cache only the highest quality versions and derive any lower quality versions as they are requested. However, this scenario may be expensive because of the high

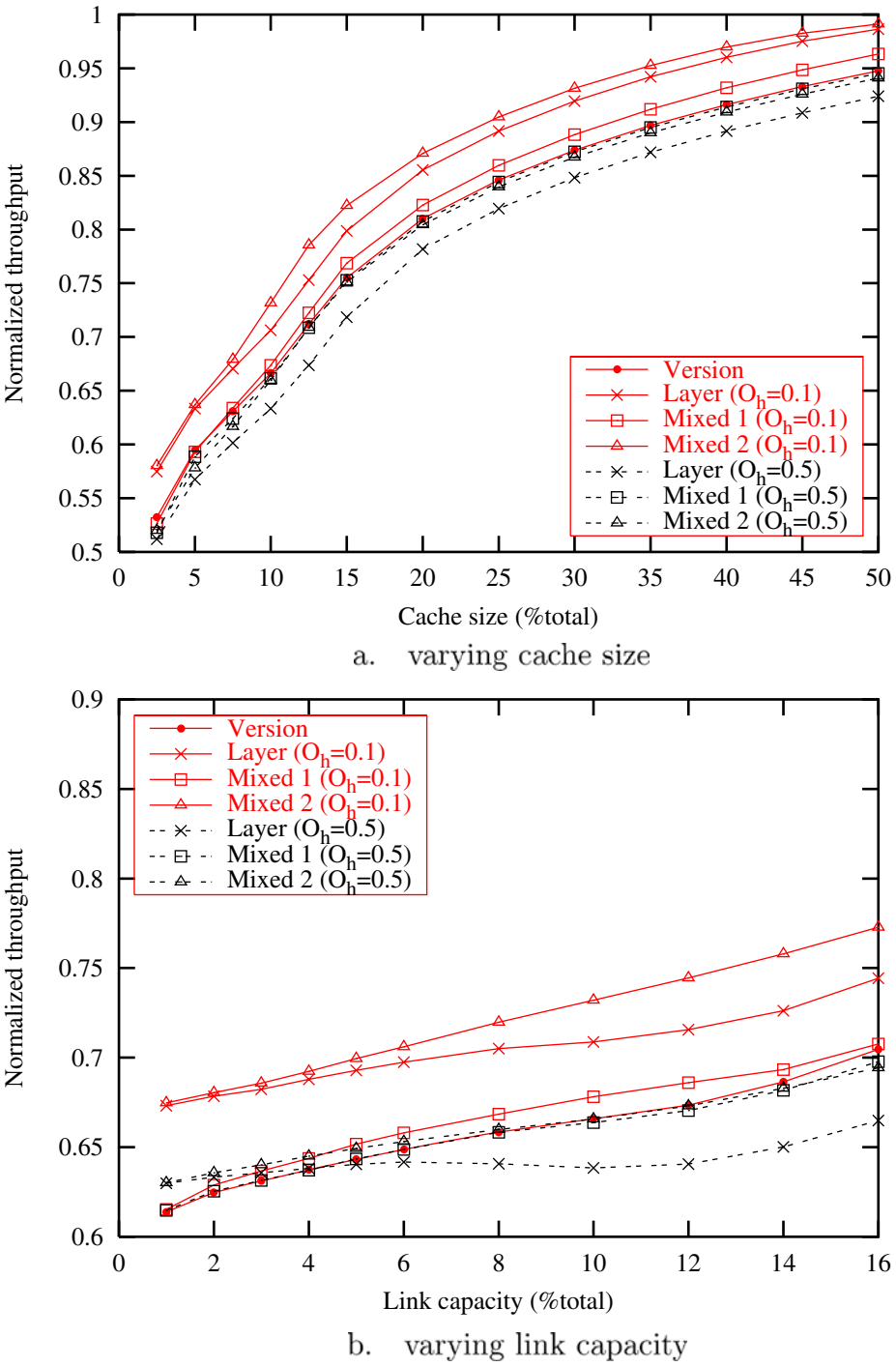


Fig. 8 Adaptive caching with varying cache size and link capacity ($q = 0.4$)

processing requirements and specialized hardware needed to perform real-time re-encoding in a proxy where we might have several transcoding operations in progress at any time.

Offline transcoding can take advantage of available processing capacity when the proxy is not so busy, but this would require us to predict the need for lower quality versions in advance. This is similar to prefetching the lower quality versions and, in order to be effective, would require efficient prefetching prediction algorithms. Any wrong prediction would mean that a significant amount of resources had been wasted on the now useless lower quality version. Prefetching has been studied in the Web (see for instance [3]), but the results can, at best, be described as a mixed success, with wrong predictions using almost as much resources as have been saved with correct predictions.

In summary, while transcoding seems like an attractive method for improving the performance of the cache, both possible approaches have their problems. Online transcoding is expensive to implement in current hardware and software and offline transcoding risks to waste more resources than it saves.

6 Conclusion

In this paper we have studied pure versions, pure layers, and mixed distribution strategies. We found that mixed distribution strikes a good balance to offer the best overall performance. Our study leads to the following guidelines for distributing multi-quality video in the Internet:

1. Caches and CDN servers should be partially pre-filled with the most popular videos. If there are requests for both quality levels of a popular video, than the server should cache both the base and the enhancement layer of the video (rather than use versions). It is important to pre-fill the cache with the popular videos; otherwise, continuously streaming moderately-popular videos may prevent popular videos from getting stored in the cache.
2. For a first-time request of a video with unknown popularity, the origin server should stream the requested quality level as a *version*, and the proxy should not cache the version. If the video experiences multiple requests, then layers should be streamed and stored in the cache.
3. Although we should use versions to stream first-time requests from origin server to client, we should not cache versions (unless all the requests for a specific video are for one quality level).

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. Career ANI-0133252 and Grant No. ANI-0136774.

References

1. Abdelzaher T, Bhatti N (1999, May) Web server QoS management by adaptive content delivery. In: Proc. of International Workshop on QoS, London, UK
2. Chandra K, Reibman A (1999, June) Modeling one- and two-layer variable bit rate video. IEEE/ACM Trans Netw 7(3):398–413

3. Crovella M, Barford P (1998, March) The network effects of prefetching. In: Proc. of IEEE Infocom, San Francisco, California, pp 1232–1239
4. DeCuetos P, Saparilla D, Ross K (2001, April) Adaptive streaming of stored video in a TCP-friendly context: multiple versions or multiple layers? In: Proc. of international packet video workshop, Kyongju, Korea
5. Fox A, Brewer EA (1996, May) Reducing WWW latency and bandwidth requirements by real-time distillation. In: Proc. 5th WWW conference, Paris, France
6. Hartanto F, Pawlikowski K, Sirisena H, Kreutzer W (1996, December) Quantitative simulation of telecommunication networks in DESC++. Comput Electr Eng 22(6):367–381
7. Kangasharju J., Hartanto F, Reisslein M, Ross K (2002, June) Distributing layered encoded video through caches. IEEE Trans Comput 51(6):622–636
8. Kim T, Ammar MH (2001, June) A comparison of layering and stream replication video multicast schemes. In: Proc. of NOSSDAV 2001, Port Jefferson, New York
9. Kimura J, Tobagi F, Pulido J, Emstad P (1999, September) Perceived quality and bandwidth characterization of layered MPEG-2 video encoding. In: SPIE international symposium on voice, video and data communications, Boston, Massachusetts
10. Ma W, Bedner I, Chang G, Kuchinsky A, Zhang HJ (2000, January) A framework for adaptive content delivery in heterogeneous network environments. In: Proc. of MMCN 2000, San Jose, California
11. Ortega A, Carignano F, Ayer S, Vetterli M (1997, June) Soft caching: web cache management techniques for images. In: Proc. of MMSP, Princeton, New Jersey
12. Real Networks. www.realnetworks.com
13. Ross KW (1995) Multiservice loss models for broadband telecommunication networks. Springer, Berlin Heidelberg New York
14. Shanableh T, Ghanbari M (2000, June) Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats. IEEE Trans Multimedia 2(2):101–110

Felix Hartanto received his B.E. and Ph.D. in Electrical and Electronic Engineering from the University of Canterbury, New Zealand, in 1990 and 1994, respectively. From 1994 to 1996, he was a postdoctoral researcher at the University of Canterbury. Next he was employed at Digital Equipment Corporation (now Compaq), New Zealand, as a software developer and project leader from 1996 to 1998. There he led a number of mobile service provisioning and billing projects. From 1998 to 2000 he was a scientist with the German National Research Center for Information Technology (GMD FOKUS) in Berlin, Germany. He is currently an Assistant Professor in the Department of Information Engineering, The Chinese University of Hong Kong. His research interests include multimedia communications, Internet quality of service, service and network management.

Jussi Kangasharju is a post-doctoral researcher in the Telecooperation group in the department of computer science at the Darmstadt University of Technology, Germany. He received his Master of Science in Technology from the Department of Computer Science and Engineering in Helsinki University of Technology, Finland, in 1998. He received his DEA from the University of Nice (Sophia Antipolis), France in 1998. He conducted his Ph.D. research in the Multimedia Communications Department of Institut Eurecom and received his Ph.D. from the University of Nice (Sophia Antipolis) in 2002. His research interests include web content distribution, peer-to-peer networking, and Internet protocols.

Martin Reisslein is an Associate Professor in the Department of Electrical Engineering at Arizona State University, Tempe. He is affiliated with the Wireless Integrated Nano Technology (WINTech) center at ASU. He received the Dipl.-Ing. (FH) degree from the Fachhochschule Dieburg, Germany, in 1994, and the M.S.E. degree from the University of Pennsylvania, Philadelphia, in 1996. Both in electrical engineering. He received his Ph.D. in systems engineering from the University of Pennsylvania in 1998. During the academic year 1994–1995 he visited the University of Pennsylvania as a Fulbright scholar. From July 1998 through October 2000 he was a scientist with the German National Research Center for Information Technology (GMD FOKUS), Berlin, and lecturer at the Technical University Berlin. He is editor–in–chief of the *IEEE Communications Surveys and Tutorials* and has served on the Technical Program Committees of *IEEE Infocom*, *IEEE Globecom*, and the *IEEE International Symposium on Computer and Communications*. He has organized sessions at the *IEEE Computer Communications Workshop (CCW)*. He maintains an extensive library of video traces for network performance evaluation, including frame size traces of MPEG–4 and H.263 encoded video, at `http://trace.eas.asu.edu`. He is co-recipient of the Best Paper Award of the *SPIE Photonics East 2000—Terabit Optical Networking* conference. His research interests are in the areas of Internet Quality of Service, video traffic characterization, wireless networking, and optical networking.

Keith W. Ross joined Polytechnic University as the Leonard Shustek Chair Professor of Computer Science in January 2003. Before joining Polytechnic University, he was a professor for 5 years in the Multimedia Communications Department at Eurecom Institute in Sophia Antipolis, France. From 1985 through 1997, he was a professor in the Department of Systems Engineering at the University of Pennsylvania. He received a B.S.E.E from Tufts University, a M.S.E.E. from Columbia University, and a Ph.D. in Computer and Control Engineering from The University of Michigan. Professor Ross has worked in stochastic modeling, QoS in packet-switched networks, video streaming, video on demand, multi-service loss networks, web caching, content distribution networks, peer-to-peer networks, application-layer protocols, voice over IP, optimization, queuing theory, optimal control of queues, and Markov decision processes. He is an associate editor for *IEEE/ACM Transactions on Networking*. Professor Ross is co-author (with James F. Kurose) of the best-selling textbook, *Computer Networking: A Top-Down Approach Featuring the Internet*, published by Addison-Wesley. He is also the author of the research monograph, *Multiservice Loss Models for Broadband Communication Networks*, published by Springer. From July 1999 to July 2001, Professor Ross founded and led Wimba, an Internet startup which develops asynchronous voice products.