Distributing Layered Encoded Video through Caches

Jussi Kangasharju†, Felix Hartanto⋆, Martin Reisslein⋆, Keith W. Ross†
†Institut Eurecom, ⋆GMD FOKUS

Abstract—**The efficient distribution of stored information has become a major concern in the Internet which has increasingly become a vehicle for the transport of stored video. Because of the highly heterogeneous access to the Internet, researchers and engineers have argued for layered encoded video. In this paper we investigate delivering layered encoded video using caches. Based on a stochastic knapsack model we develop a model for the layered video caching problem. We propose heuristics to determine which videos and which layers in the videos should be cached. We evaluate the performance of our heuristics through extensive numerical experiments. We also consider two intuitive extensions to the initial problem.**

Keywords— **Proxy Caching, Streaming Layered Video, Utility Heuristics, Stochastic Knapsack**

I. Introduction

In recent years, the efficient distribution of stored information has become a major concern in the Internet. In the late 1990s numerous companies - including Cisco, Microsoft, Netscape, Inktomi, and Network Appliance - began to sell Web caching products, enabling ISPs to deliver Web documents faster and to reduce the amount of traffic sent to and from other ISPs. More recently the Internet has witnessed the emergence of content distribution network companies, such as Akamai and Sandpiper, which work directly with content providers to cache and replicate the providers' content close to the end users. In parallel to all of this caching and content distribution activity, the Internet has increasingly become a vehicle for the transport of stored video. Many of the Web caching and content distribution companies have recently announced new products for the efficient distribution of stored video.

Access to the Internet is, of course, highly heterogeneous, and includes 28K modem connections, 64K ISDN connections, shared-bandwidth cable modem connections, xDSL connections with downstream rates in 100K-6M range, and high-speed switched Ethernet connections at 10 Mbps. Researchers and engineers have therefore argued that layered encoded video is appropriate for the Internet. When a video is layered encoded, the number of layers that are sent to the end user is a function of the user's downstream bandwidth.

An important research issue is how to efficiently distribute stored layered video from servers (including Web servers) to end users. As with Web content, it clearly makes sense to insert intermediate caches between the servers and clients. This will allow users to access much of the stored video content from nearby servers, rather accessing the video from a potentially distant server. Given the presence of a caching and/or content distribution network infrastructure, and of layered video in origin servers, a fundamental problem is to determine *which videos* and *which layers in the videos* should be cached. Intuitively, we will want to cache the more popular videos, and will want to give preference to the lower base layers rather than to the higher enhancement layers.

In this paper we present a methodology for selecting which videos and which layers should be stored at a finite-capacity cache. The methodology could be used, for example, by a cable or ADSL access company with a cache at the root of the distribution tree. Specifically, we suppose that the users have high-speed access to the cache, but the cache has limited storage capacity and a limited bandwidth connection to the Internet at large. For example, the ISP might have a terabyte cache with a 45 Mbps connection to its parent ISP. Thus, the video caching problem has two constrained resources, the cache size and the transmission rate of the access link between the ISP and its parent ISP. Our methodology is based on a stochastic knapsack model of the 2-resource problem. We suppose that the cache operator has a good estimate of the popularities of the the video layers. The problem, in essence, is to determine which videos and which layers within the video should be cached so that customer demand can best be met.

This paper is organized as follows. In Section II we present our layered video streaming model. In Section III we present our utility heuristics and evaluate their performance. Section IV extends our caching model by adding the possibility to negotiate the delivered stream quality. Section V considers a queueing scheme for managing client requests. Section VI considers the usefulness of partial caching. Section VII presents an overview of related work and Section VIII concludes the paper.

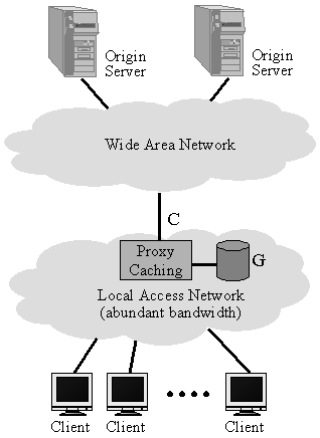

Fig. 1. Architecture for caching and streaming of layered encoded video.

II. Model of Layered Video Streaming with Proxy

Fig. 1 illustrates our architecture for continuous media streaming with proxy servers. We first give a rough overview of our streaming architecture and then discuss each component in detail. All available continuous media objects are stored on the origin servers. Popular streams are cached in proxy servers. The clients direct their streaming requests to the appropriate proxy server. If the requested stream is cached in the proxy, it is directly streamed over the local access network to the client. If the requested stream is not cached in the proxy, it is streamed from the origin server over the wide area network to the proxy. The proxy forwards the stream to the client.

A. Layered Video

The continuous media objects available on the origin servers are prerecorded audio and video objects, such as CD–quality music clips, short video clips (e.g., news clips, trailers or music videos) or full–length movies or on–line lectures. Our focus in this study is on video objects that have been encoded using layered (hierarchical) encoding techniques [1–3]. With hierarchical encoding each video object is encoded into a base layer and one or more enhancement layers. The base layer contains the most essential basic quality information. The enhancement layers provide quality enhancements. A particular enhancement layer can only be decoded if all lower quality layers are available. Therefore, an enhancement layer is useless for the client if the corresponding lower quality layers are not available.

Layered video allows service providers to offer flexible streaming services to clients with vastly different reception bandwidths and decoding capabilities. Typically, wireless clients and clients with modem–speed wireline Internet access will request only the base layer stream. Clients with high–speed ADSL or cable modem access, on the other hand, may wish to receive higher quality streams consisting of base layer as well enhancement layers. Furthermore, layered video allows for flexible pricing structures. A service provider may offer the base layer stream at a basic rate and charge a premium for the enhancement layers. In other words, clients are charged more when receiving more layers (i.e., higher quality streams). Such a pricing structure might prompt clients to request the cheaper base layer–only stream of a news clip or talk show, say, while requesting the more expensive high quality stream of an entertainment movie.

To make the notion of layered video objects more precise, suppose that there are M video objects. We assume that the video objects are encoded into Constant Bit Rate (CBR) layers, which is a reasonable first approximation of the output of hierarchical codecs. For notational simplicity we assume that all video objects are encoded into L layers. (Our model extends to video objects that differ in the number of layers in a straightforward manner.) Let $r_l(m)$ denote the rate (in bit/sec) of layer $l,\ l = 1, \ldots, L$, of video object $m,\ m = 1, \ldots, M$. We define a j–quality stream as a stream consisting of layers $1, 2, \ldots, j$. Let $T(m),\ m = 1, \ldots, M$, denote the length (in seconds) of video object m. Let $R(j, m)$ denote the revenue accrued from providing a j–quality stream of object m.

B. Proxy Server

The proxy server is located close to the clients. It is connected to the origin servers via a wide area network (e.g., the Internet). We model the bandwidth available for streaming continuous media from the origin servers to the proxy server as a bottleneck link of fixed capacity C (bit/sec). The proxy is connected to the clients via a local access network. The local access network could be a LAN running over Ethernet, or a residential access network using xDSL or HFC technologies. For the purpose of this study we assume that there is abundant bandwidth for continuous media streaming from the proxy to the clients. We model the proxy server as having a storage capacity of G (bytes). We assume that the proxy storage has infinite storage bandwidth (for reading from storage). We note that the proxy storage is typically a disk array with limited storage bandwidth due to the limited disk bandwidths and seek and rotational overheads. Our focus in this study, however, is on gaining a fundamental understanding of the impact of the two basic streaming resources (bottleneck bandwidth C and cache space G) on the proxy performance. We refer the interested reader to [4–6] for a detailed discussion of the disk array limitations as well as discussions on replica-

tion and striping techniques to mitigate these limitations.

We consider a caching scenario where the cache contents are updated periodically, say every few hours, daily, or weekly. The periodic cache updates are based on estimates of the request pattern of the proxy's client community. A service provider may estimate the request pattern from observations over the last couple of days or weeks. Suppose that the requests for video streams arrive according to a Poisson process with rate λ (requests/sec). Let $p(j,m)$ denote the popularity of the j–quality stream of object m, that is, $p(j,m)$ is the probability that a request is for the j–quality stream of object m. These popularities could be estimated from the observed requests using an exponential weighted moving average. As a proper probability mass distribution the $p(j,m)$'s satisfy $\sum_{m=1}^{M}\sum_{j=1}^{L} p(j,m) = 1$. Also, note that the arrival rate of requests for the j–quality stream of object m is given by $\lambda p(j,m)$.

Our focus in this study is on caching strategies that cache complete layers of video objects in the proxy. Our goal is to cache object layers so as to maximize the revenue accrued from the streaming service. When updating the cache our heuristics give layers of very popular objects priority over layers of moderately popular objects. Moreover, lower quality layers are given priority over higher quality layers (as these require the lower quality layers for decoding at the clients).

To keep track of the cached object layers we introduce a vector of cache indicators $\mathbf{c} = (c_1, c_2, \ldots, c_M)$, with $0 \le c_m \le L$ for $m = 1, \ldots, M$. The indicator c_m is set to i if layers 1 through i of object m are cached. Note that $c_m = 0$ indicates that no layer of object m is cached. With the cache indicator notation the cache space occupied by the cached object layers is given by

$$S(\mathbf{c}) = \sum_{m=1}^{M}\sum_{l=1}^{c_m} r_l(m)T(m). \qquad (1)$$

C. Stream Delivery

The client directs its request for a j–quality stream of a video object m to its proxy server (for instance by using the Real Time Streaming Protocol (RTSP) [7]). If all the requested layers are cached in the proxy ($c_m \ge j$), the requested layers are streamed from the proxy over the local access network to the client. If layers are missing in the proxy ($c_m < j$), the appropriate origin server attempts to establish a connection for the streaming of the missing layers $c_m + 1, \ldots, j$ at rate $\sum_{l=c_m+1}^{j} r_l(m)$ over the bottleneck link to the client. If there is sufficient bandwidth available, the connection is established and the stream occupies the link bandwidth $\sum_{l=c_m+1}^{j} r_l(m)$ over the lifetime of the stream. (The layers $1, \ldots, c_m$ are streamed from the proxy directly to the client.) We assume that the client watches the entire stream without interruptions, thus the bandwidth $\sum_{l=c_m+1}^{j} r_l(m)$ is occupied for $T(m)$ seconds. In the case there is not sufficient bandwidth available on the bottleneck link, we consider the request as blocked. (In Section IV we study a refined model where clients may settle for a lower quality stream in case their original request is blocked.)

Formally, let $B_{\mathbf{c}}(j,m)$ denote the blocking probability of the request for a j–quality stream of object m, given the cache configuration $\mathbf{c}$. Clearly, there is no blocking when all requested layers are cached, that is, $B_{\mathbf{c}}(j,m) = 0$ for $c_m \ge j$. If the request requires the streaming of layers over the bottleneck link ($c_m < j$), blocking occurs with a non–zero probability $B_{\mathbf{c}}(j,m)$. We calculate the blocking probabilities $B_{\mathbf{c}}(j,m)$ using results from the analysis of multiservice loss models [8]. An overview of the relevant loss modeling is provided in the Appendix. In summary, we model the bottleneck link as a stochastic knapsack of capacity C. Requests for j–quality streams ($j = 1, \ldots, L$) of object m, $m = 1, \ldots, M$ are modeled as a distinct class of requests, thus there is a total of ML distinct classes of requests. The load offered by requests for j–quality streams of object m is $\lambda p(j,m)T(m)$. The blocking probabilities $B_{\mathbf{c}}(j,m)$ for the request classes can be calculated using the recursive Kaufman–Roberts algorithm [8, p. 23] with a time complexity of $O(CML)$. The expected blocking probability of a client's request is given by

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

The service provider should strive to keep the expected blocking probability acceptably small, say, less than 5%. The throughput of requests for j–quality streams of object m, that is, the long run rate at which these requests are granted and serviced is $\lambda p(j,m)(1 - B_{\mathbf{c}}(j,m))$. The long run rate of revenue accrued from the serviced j–quality streams of object m is the revenue per served request, $R(j,m)$, multiplied by the throughput. Thus, the long run total rate of revenue of the streaming service is

$$R(\mathbf{c}) = \lambda \sum_{m=1}^{M}\sum_{j=1}^{L} R(j,m)p(j,m)(1 - B_{\mathbf{c}}(j,m)). \qquad (2)$$

Our goal is to cache object layers so as to maximize the total revenue rate.

III. Optimal Caching

In this section we study optimal caching strategies. Suppose that the stream popularities ($p(j,m)$) and the stream

characteristics (layer rates $r_l(m)$ and lengths $T(m)$) are given. The question we address is how to best utilize the streaming resources — bottleneck bandwidth C and cache space G — in order to maximize the revenue. Our focus in this study is on optimal caching strategies, that is, we focus on the question: which objects and which layers thereof should be cached in order to maximize the revenue? Formally, we study the optimization problem $\max_{\mathbf{c}} R(\mathbf{c})$ subject to $S(\mathbf{c}) \leq G$. Throughout this study we assume the complete sharing admission policy for the bottleneck link, that is, a connection is always admitted when there is sufficient bandwidth. We note that complete sharing is not necessarily the optimal admission policy. In fact, the optimal admission policy may block a request (even when there is sufficient bandwidth) to save bandwidth for more profitable requests arriving later. We refer the interested reader to [8, Ch. 4] for a detailed discussion on optimal admission policies. Our focus in this study is on the impact of the *caching* policy on the revenue; we assume complete sharing as a baseline admission policy that is simple to describe and administer.

The maximization of the long run revenue rate $R(\mathbf{c})$ over all possible caching strategies (i.e., cache configurations $\mathbf{c}$) is a difficult stochastic optimization problem, that — to the best of our knowledge— is analytically intractable. To illustrate the problem consider a scenario where all video layers have the same rate r and length T, i.e., $r_l(m) = r$ and $T(m) = T$ for all $l = 1, \ldots, L$, and all $m = 1, \ldots, M$. In this scenario all object layers have the size rT. Thus, we can cache up to $G/(rT)$ object layers (which we assume to be an integer for simplicity). Suppose that during the observation period used to estimate the stream popularities, the proxy has recorded requests for M distinct objects from its client community. Thus, there are a total of ML object layers to choose from when filling the cache (with "hot" new releases there might even be more objects to consider). Typically, the cache can accommodate only a small subset of the available object layers, i.e., $G/(rT) \ll ML$. For an exhaustive search there are $\begin{pmatrix} ML \\ G/(rT) \end{pmatrix}$ possibilities to fill the cache completely; a prohibitively large search space even for small ML.

Recall that with layered encoded video a particular enhancement layer can only be decoded if all lower quality layers are available. Therefore, a reasonable restriction of the search space is to consider a particular enhancement layer for caching only if all lower quality layers of the corresponding object are cached. Even the "reasonable" search space, however, is prohibitively large for moderate ML; with $M = 50$, $L = 2$, $G/(rT) = 20$, for instance, there are $2.929 \cdot 10^{16}$ possibilities to fill the cache completely.

TABLE I
UTILITY DEFINITIONS.

Popularity utility	$u_{l,m} = \sum_{j=l}^{L} p(j,m)$
Revenue utility	$u_{l,m} = \sum_{j=l}^{L} R(j,m)p(j,m)$
Revenue density utility	$u_{l,m} = \sum_{j=l}^{L} \frac{R(j,m)p(j,m)}{r_j(m)T(m)}$

Because the maximization problem $\max_{\mathbf{c}} R(\mathbf{c})$ subject to $S(\mathbf{c}) \leq G$ is analytically intractable and exhaustive searches over $\mathbf{c}$ are prohibitive for realistic problems, we propose heuristics for finding the optimal cache composition $\mathbf{c}$.

A. Utility Heuristics

The basic idea of our utility heuristics is to assign each of the ML object layers a cache utility $u_{l,m}$, $l = 1, \ldots, L$, $m = 1, \ldots, M$. The object layers are then cached in decreasing order of utility, that is, first we cache the object layer with the highest utility, then the object layer with the next highest utility, and so on. If at some point (as the cache fills up) the object layer with the next highest utility does not fit into the remaining cache space, we skip this object layer and try to cache the object layer with the next highest utility. Once a layer of an object has been skipped, all other layers of this object are ignored as we continue "packing" the cache. We propose a number of definitions of the utility $u_{l,m}$ of an object layer; see Table I for an overview.

The popularity utility is based exclusively on the stream popularities; it is defined by $u_{l,m} = p(l,m) + p(l+1,m) + \cdots + p(L,m)$. This definition is based on the decoding constraint of layered encoded video, that is, an object layer l is required (i.e., has utility) for providing l–quality streams (consisting of layers 1 through l), $l+1$–quality streams, ..., and L–quality streams. Note that $u_{l,m}$ is the probability that a request involves the streaming of layer l of object m. Also, note that by definition $u_{l,m} \geq u_{l+1,m}$ for $l = 1, \ldots, L-1$. This, in conjunction with our packing strategy ensures that a particular enhancement layer is cached only if all corresponding lower quality layers are cached.

B. Evaluation of Heuristics

In this section we present some numerical results from both analytical and simulation experiments to evaluate various aspects of the heuristics algorithms. The analytical experiments based on exhaustive optimal search are carried out to evaluate the proximity of the solution provided by the heuristics algorithm to the actual optimal solution.

The simulation experiments, on the other hand, are carried out to verify the correctness of blocking probability calculation used by the heuristics algorithm.

We assume that there are 1000 different movies, each encoded into two layers. The characteristics of each movie are defined by the rate for each layer and its length. The rate for each layer is drawn randomly from a uniform distribution between 0.1 and 3 Mbps, while the length of the movie is drawn from an exponential distribution with an average length of 1 hour.

In the simulation experiments client requests arrive according to a Poisson process. The average request arrival rate is 142 Erlangs. The client can request either a base layer only or a complete movie. The request type and the movie requested are drawn randomly from a Zipf distribution with a parameter of $\zeta = 1.0$. The revenue for each movie layer is uniformly distributed between 1 to 10.

The results of interest will be the revenue per hour and the blocking probabilities. To obtain the results with 99% confidence intervals, we run the experiments with different random seeds and we require a minimum of 10000 runs before calculating the confidence intervals. In each run we randomly assign the popularities of movies from the Zipf distribution, the rates and the lengths of the movie layers. The results are calculated as the average value of the revenue per hour from all the runs until the confidence intervals are reached.

We first tested the performance of our heuristics in small problems in order to be able to compare the heuristic against the "reasonable" exhaustive search. For the small problems we set $M = 10$ with each movie having two layers. We varied the link bandwidth C between 3 and 15 Mbit/s and the cache capacity between 3 and 7 Gbytes. The cache could therefore store on the average between 3.5 and 7.6 layers out of the total 20 layers, or between 23.1 and 41.7% of the total movie data.

The results of the small problems are shown in Table II. In Table II we show the average error obtained with each heuristic compared to the "reasonable" exhaustive search for four different cache configurations. The *Small Link* and *Large Link* refer to link capacities of 3 Mbit/s and 15 Mbit/s, respectively, and *Small Cache* and *Large Cache* refer to 3 Gbyte and 7 Gbyte caches, respectively.

As we can see, our heuristics achieve performance very close to the optimum in most cases. Only when both the link and the cache are small is there any marked difference in performance. This is largely due to the small link capacity, only 3 Mbit/s, which allows us to stream only one movie on the average. As both the link and cache grow in size, we can achieve the same performance as the optimal caching strategy.

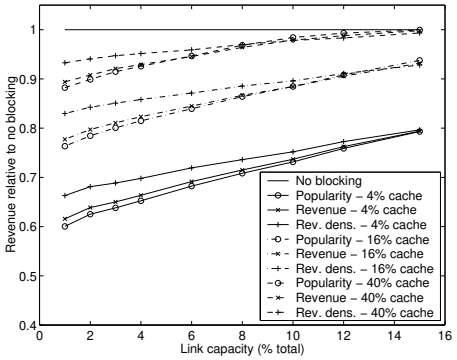

Fig. 2. Revenue as function of link capacity for 3 different cache sizes

To test the performance of our heuristics in real-world size problems, we ran the heuristics for 1000 movies. We varied the cache size between 12 and 560 Gbytes. The cache could therefore hold on the average between 13.9 and 625 layers, or between 0.9 and 41.7% of the total movie data. Given the average length of a movie T_{avg}, the average rate of a movie r_{avg}, and the client request rate λ, we would need on the average $T_{avg} r_{avg} \lambda$ Mbit/s of bandwidth to stream all the requested movies. We varied the link capacity between 10 and 150 Mbit/s, or between 1 and 15% of the total bandwidth required.

Because running the exhaustive search was not feasible for problems this large, we approximated the best possible performance by calculating the revenue when the blocking probability was zero. This means that all client requests are always satisfied and it provides us with an upper limit on the achievable revenue. In reality, this upper limit is not reachable unless the link and cache capacities are sufficiently large to ensure that no client requests are ever blocked. In our tests the smallest observed blocking probabilities were around 0.005%.

In Fig. 2 we show the revenue relative to the no blocking case obtained with 3 different cache sizes as a function of the link capacity. We can see that the revenue density heuristic performs the best overall and that the performance difference is biggest when the link capacity is smaller. As the link capacity increases, the performance difference disappears. We also see that the popularity heuristic has the worst overall performance.

In Fig. 3 we show the revenue obtained with 2 different link capacities as a function of the cache size. Here the difference between revenue density heuristic and the others is clearer. For example, with a 1% link and a 20% cache (10 Mbit/s link and a cache of 250 Gbytes in our case), revenue density heuristic achieves 87% of the upper limit while the revenue heuristic achieves only 79%. Again, as in Fig. 2, when we have enough link and cache capacity, the difference between the heuristics disappears.

TABLE II
AVERAGE ERROR OF HEURISTICS IN SMALL PROBLEMS

	Small Link		Large Link	
Utility heuristic	**Small Cache**	**Large Cache**	**Small Cache**	**Large Cache**
Popularity	1.6%	2.4%	0.006%	0%
Revenue	2.8%	0.4%	0.1%	0%
Revenue density	0.3%	0.3%	0.1%	0%

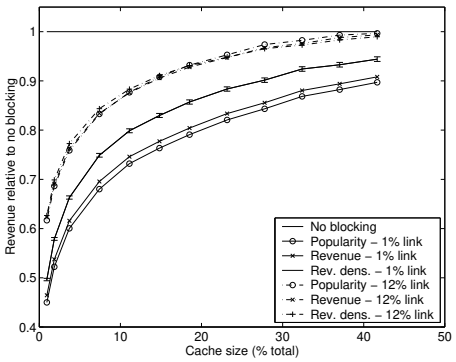

Fig. 3. Revenue as function of cache size for 2 different link capacities

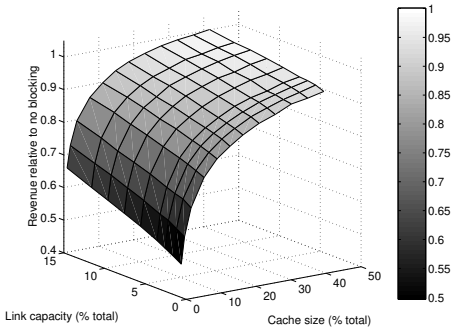

Fig. 4. Revenue as function of cache size and link capacity

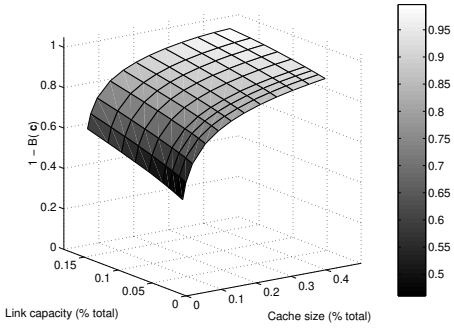

Fig. 5. $1 - B(\mathbf{c})$ as function of cache size and link capacity

To illustrate the tight confidence intervals we observed, we plot the revenue density heuristic in the 1% link case with the 99% confidence intervals.

Overall, we can conclude that the revenue density utility heuristic has the best performance of the three heuristics studied. This is especially true in situations where we have a shortage of one of the resources, link capacity or cache size. This implies that the revenue density heuristic predicts the usefulness of a layer more accurately than the other two heuristics.

In Fig. 4 we show the revenue obtained with the revenue density heuristic as a function of both link capacity and cache size. We observe that if we have a shortage of both resources, we should first increase the cache before increasing the link capacity. We see that when the cache size is around 20% of the total movie data (250 Gbytes in our case), further increase in cache size provides only small gains in revenue. At this point, increasing the link capacity provides larger gains in revenue. This behavior can also be observed in Figs. 2 and 3 where we can see that the revenue increases roughly linearly with the link capacity and roughly logarithmically with the cache size.

In Fig. 5 we show the expected blocking probability for the revenue density heuristic. Note that the plot shows $1 - B(\mathbf{c})$ and smallest expected blocking probability is therefore obtained when the curve is close to 1.

We also studied the effects of varying the parameter ζ in the Zipf-distribution and varying the client request rate, λ. Previous studies in Web caching and server access dynamics have found that ζ can vary from 0.6 in Web proxies [9] up to 1.4 in popular Web servers [10]. We studied four different values of ζ, namely 0.6, 0.8, 1.0, and 1.3. In Fig. 6 we show the revenue obtained with each of the four parameter values for three different link capacities as a function of the cache size. We can see that the curves corresponding to one value of ζ are close together and that there is a significant difference in groups of curves belonging to different values of ζ. This implies that a decrease in ζ (movies become more equally popular) requires significant increases in link capacity and cache size to keep the revenue at the same level. On the other hand, should ζ increase (small number of movies become very popular), we can achieve the same revenue with considerably less resources.

In Fig. 7 we show the effects of varying the client request rate. We plot curves for three different values of λ for two different link capacities. The curves for "Low λ at 6% link" and "Medium λ at 10% link" fall on top of each other. We can clearly see that the client request rate has

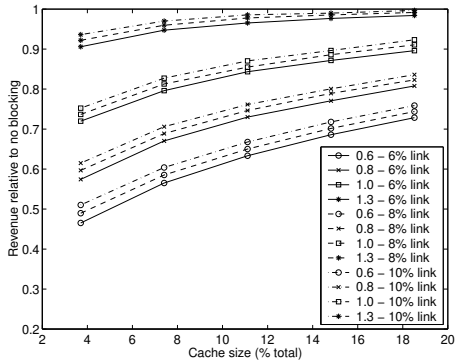

Fig. 6. Effect of Zipf-parameter ζ on revenue

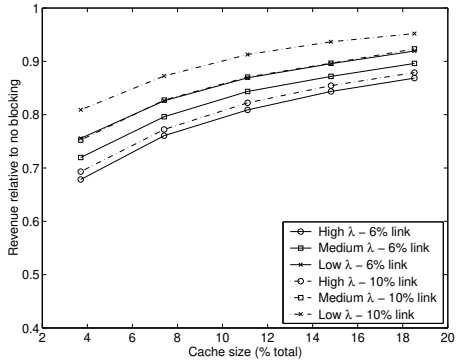

Fig. 7. Effect of client request rate on revenue

much less effect on the revenue than the Zipf-parameter. In some cases, it is possible to counter the changes in request rate by increasing the link capacity or cache size. For example, if the request rate goes from Low to Medium, increasing the link capacity from 6% to 10% (60 Mbit/s to 100 Mbit/s in this case) keeps the revenue the same.

In conclusion, all three of our heuristics perform well under many different link and cache size combinations. The revenue density heuristic achieves the best performance under constrained conditions.

IV. Negotiation about Stream Quality

In this section we study a negotiation scheme where in case the client's original request is blocked, the service provider tries to offer a lower quality stream of the requested object. The client may then settle for this lower quality stream. The question we address is: how much additional revenue is incurred with this "negotiation." As we shall demonstrate, this intuitively quite appealing approach adds very little to the revenue in most situations. For simplicity we focus in this section on video objects that are encoded into L =2 layers: a base layer and one enhancement layer. (Our arguments extend to the case of more encoding layers in a straightforward manner.) Suppose that a client requests a 2–quality stream (consisting of base layer and enhancement layer) of object m. Suppose that the cache configuration is given by $\mathbf{c}$. Clearly, the original request can only be blocked if not all requested layers are cached, that is, if $c_m < 2$. If the client's original request for a 2–quality stream of object m is blocked the service provider tries to offer a 1–quality (i.e., base layer) stream of the object. The service provider is able to make this offer if the base layer stream is not blocked.

Note that the negotiations increase the arrival rates of requests for base layer streams. This is because the blocked 2–quality stream requests "reappear" as base layer stream requests. With negotiations the arrival rates of base layer stream requests depend on the blocking probabilities of 2–quality stream requests, that is, the system becomes a generalized stochastic knapsack [8, Ch. 3]. Calculating the blocking probabilities of the generalized stochastic knapsack, however, is quite unwieldy. Therefore we approximate the blocking probabilities of the streaming system with negotiations. In typical streaming systems the blocking probabilities are small, typically less than 5 %. The increase in the arrival rates of base layer stream requests is therefore relatively small. We approximate the blocking probabilities of the system with negotiations by the blocking probabilities of the system without negotiations. The probability that the client's original request for a 2–quality stream of object m is blocked is approximately $B_{\mathbf{c}}(2,m)$. The probability that the corresponding base layer stream is not blocked is approximately $1 - B_{\mathbf{c}}(1,m)$. Suppose that the client accepts the quality degradation with probability $P_{\rm acc}(m)$. If the client does not accept the offer the negotiation terminates. Thus, given that the negotiation is entered, it ends in a success (i.e., service provider and client settle for a base layer stream) with probability $(1 - B_{\mathbf{c}}(1,m))P_{\rm acc}(m)$. The long run rate (successful negotiations per hour) at which negotiations settle for a base layer stream of object m is $\lambda p(2,m)B_{\mathbf{c}}(2,m)(1 - B_{\mathbf{c}}(1,m))P_{\rm acc}(m)$. Suppose that each successful negotiation resulting in the delivery of a base layer stream of object m incurs a revenue of $R_{\rm neg}(1,m)$ (which may be different from $R(1,m)$ as the service provider may offer the base layer at a discount in the negotiation). Thus, the long run total rate of revenue incurred from successful negotiations is

$$R_{\rm neg}(\mathbf{c}) = \lambda \sum_{m=1}^{M} R_{\rm neg}(1,m)p(2,m)B_{\mathbf{c}}(2,m) \\ (1 - B_{\mathbf{c}}(1,m))P_{\rm acc}(m).$$

The long run total rate of revenue of the streaming service with negotiations is $R(\mathbf{c}) + R_{\rm neg}(\mathbf{c})$, where $R(\mathbf{c})$, the revenue rate incurred from serving first–choice requests, is given by (2).

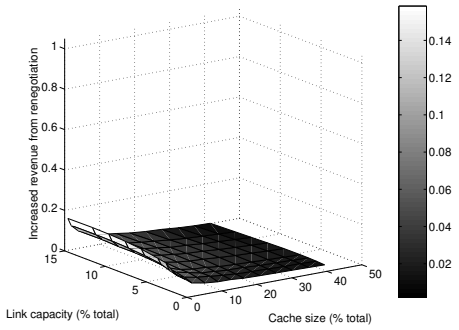

Fig. 8. Increased revenue from renegotiation

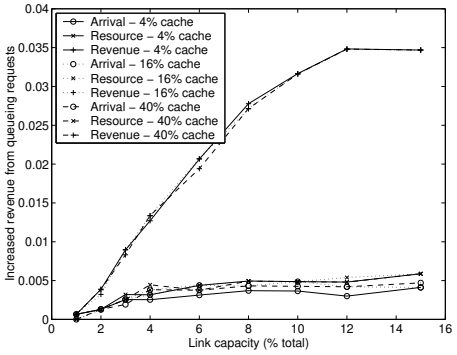

Fig. 9. Increased revenue from queueing requests for buffer size of 100

A. *Numerical Results*

We experimented with adding the renegotiation revenue to our tests. We first tested the quality of the approximation used in calculating the blocking probability of the system with renegotiation against the simulation results. We varied the link capacities between 10 to 120 Mbps. Our results show a close approximation of the analysis to the simulation results with an average error of 0.4–0.5% for 12 Gbyte cache and 0.7–1.1% for 560 Gbyte cache.

Fig. 8 shows how much extra revenue renegotiation could bring relative to the baseline revenue $R(\mathbf{c})$. The revenue in Fig. 8 is based on the assumption that the client will always accept the lower quality version if one is available, i.e., $P_{\rm acc}(m) = 1$ for $m = 1, \ldots, M$. We also assumed that $R_{\rm neg}(1, m) = R(1, m)$ for $m = 1, \ldots, M$, i.e., the revenue from the renegotiated stream is the same as if the client had requested the lower quality stream in the first place. These two assumptions give us the maximum possible gain from renegotiation.

As we can see from Fig. 8, the largest gains from renegotiation are achieved when the cache size is extremely small, only 1–2% of the total amount of data. The renegotiation gains are almost insensitive to link capacity with the exception of very small link capacities where the gains are slightly smaller. The maximum gain we observed is around 20% and the gain drops sharply as the cache size increases. The maximum gain would decrease as the client acceptance probability $P_{\rm acc}$ decreases. Also, if the cache size and link capacity are large, the potential gain from renegotiation is typically well below 1%. We can therefore conclude that renegotiation, although intuitively appealing, does not provide any significant increase in revenue in most situations. This is because renegotiation is only applicable to blocked requests and one of the goals of a cache operator would be to keep the expected blocking probability as low as possible.

V. Queueing of Requests

In this section we study a request queueing scheme where in case the client's request is blocked, the service provider queues the request. With the queueing strategies, we expect that the queued requests make use of the resources released by currently served requests. This has the potential of increasing the resource utilization and thus, bringing additional revenue. The question is how much additional revenue does it bring.

We use simulation experiments to answer this question. To align the experiments with the real-world practice, we assume that a client will cancel its request after waiting for some time, referred to as the *request timeout period*. We model the timeout period using an exponential distribution with an average of 5 minutes.

We assume that the queue is of a finite size. An incoming request finding a full buffer will be blocked. We consider three different strategies for ordering the requests in the queue, i.e., based on the order of request *arrivals*, their required *resources* and the potential *revenues*.

Fig. 9 shows how much extra revenue queueing of requests could bring relative to the baseline revenue $R(\mathbf{c})$. As we can see from the figure, the gain from introducing the queue is very small. The gain is not affected by the cache size. The gain generally increases with the link capacity.

With the limited bandwidth of the bottleneck link, which causes request blocking in the first place, the serving of one request from the queue will mean the blocking of another incoming request. This results in a near zero gain in the number of requests served. A possible gain can be achieved by changing the request service strategies, for example by serving the request according to the potential revenue that it brings.

VI. Is Partial Caching Useful ?

Consider a streaming system where *clients are only interested in complete streams* (consisting of all L layers) and *no revenue is incurred for partial streams* (consisting of less than L layers). The question we address is: in such a system is caching of partial streams (e.g., base layers) beneficial? Interestingly, the answer appears to be *no*.

We focus on the homogeneous two–layer case where the video objects are encoded into L = 2 layers: a base layer of rate $r_1(m)$ and one enhancement layer of rate $r_2(m)$. For simplicity we assume that (1) all videos have the same layer rates, i.e., $r_1(m) = r_b$ and $r_2(m) = r_e$ for $m = 1, \ldots, M$, and (2) all videos have the same length T. We study a system where clients request only complete streams (consisting of both base layer and enhancement layer), i.e., $p(1, m) = 0$ for $m = 1, \ldots, M$. For ease of notation we write $p(m)$ for $p(2, m)$ and note that $\sum_{m=1}^{M} p(m) = 1$. We order the video objects from most popular to least popular; thus, $p(m) \geq p(m+1),\ m = 1, \ldots, M-1$. In the considered system no revenue is incurred for streams consisting of only the base layer, i.e., $R(1, m) = 0$. We assume that all complete streams incur the same revenue, i.e., $R(2, m) = R$ for $m = 1, \ldots, M$.

We investigate a caching strategy that caches both base and enhancement layer of very popular video objects. For moderately popular objects only the base layer is cached (and the enhancement layer is streamed upon request over the bottleneck link of capacity C). For relatively unpopular objects neither base nor enhancement layer is cached. Let N_1 denote the number of completely cached objects. Clearly, $0 \leq N_1 \leq \lfloor G/(r_b + r_e)T) \rfloor := N_1^{\max}$. Let N_2 denote the number of cached base layers. The N_1 completely cached objects take up the cache space $N_1(r_b + r_e)T$. Hence, $0 \leq N_2 \leq \lfloor (G - N_1(r_b + r_e)T)/(r_b T) \rfloor := N_2^{\max}$. The investigated caching strategy caches base and enhancement layer of the N_1 most popular objects, that is, objects $1, \ldots, N_1$. It caches the base layers of the N_2 next most popular objects, that is of objects $N_1 + 1, \ldots, N_1 + N_2$.

The probability that a request is for a completely cached object is $P_1 = \sum_{m=1}^{N_1} p(m)$. The probability that a request is for an object for which only the base layer has been cached is $P_2 = \sum_{m=N_1+1}^{N_1+N_2} p(m)$. Note that the probability that a request is for an object which has not been cached at all is $P_3 = 1 - P_1 - P_2$.

We model the bottleneck link connecting the cache to the wide area network again as a stochastic knapsack [8]. The bottleneck link is modeled as a knapsack of capacity C. We refer to streams of completely cached video objects as class 1 streams. Class 1 streams consume no bandwidth on the bottleneck link, that is, $b_1 = 0$. The arrival rate of class 1 streams is $\lambda_1 = \lambda P_1$. Streams of video objects for which only the base layer is cached are referred to as class 2 streams. Class 2 streams consume the bandwidth $b_2 = r_e$. The arrival rate for class 2 streams is $\lambda_2 = \lambda P_2$. Streams of video objects which have not been cached at all are referred to as class 3 streams. Class 3 streams consume the bandwidth $b_3 = r_b + r_e$ and have an arrival rate of $\lambda_3 = \lambda P_3$. All streams have a fixed holding time T.

Our objective is to maximize the total long run revenue rate, or equivalently, the long run throughput of requests (i.e., the long run rate at which requests are granted and serviced). Towards this end let TH_k denote the long run throughput of class k requests. Also, let TH denote the long run total throughput of requests. Clearly, $TH = TH_1 + TH_2 + TH_3$. Let B_k denote the probability that a request for a stream of class k is blocked. Obviously, $B_1 = 0$ since class 1 streams do not consume any bandwidth. Thus, $TH = \lambda[P_1 + P_2(1 - B_2) + P_3(1 - B_3)]$.

A. Numerical Results

We used the same experiment setup as for evaluating the performance of the utility heuristics in Section III-B. In fact, we can consider the partial caching case as a special case of the utility heuristics. Note that for the partial caching case the utilities of the base and enhancement layer of a given movie are the same and thus base layer and enhancement layer are cached together.

In our experiments we question the usefulness of partial caching where a portion of the cache is reserved for caching base layers only. Doing so allows us to cache (at least the base layers of) a larger number of movies for the same cache size. An intuitive question to follow is whether *trunk reservation* is beneficial. With trunk reservation a portion of the link bandwidth, say $C_2 = x\%$ of C, $x = 0 - 100$, is reserved for streaming the enhancement layers of the class 2 movies which have base layers in the cache. We naturally expect that a combination of these two strategies may give us the best throughput.

Fig. 10 shows the normalized throughput as a function of the percentage of cache space used for caching complete movies. The figure also shows the throughput for different link reservation and cache sizes. The link reservation of 0% implies a complete sharing of the link bandwidth between class 2 and class 3 streams. This case can be analyzed using the *stochastic knapsack* formulation, see Section II-C, which gives us the blocking probabilities B_2 and B_3 and hence the throughput. On the other hand, the link reservation of 100% implies a total blocking of class 3 streams. The link is solely used for streaming enhancement layers for class 2 streams which have base layers

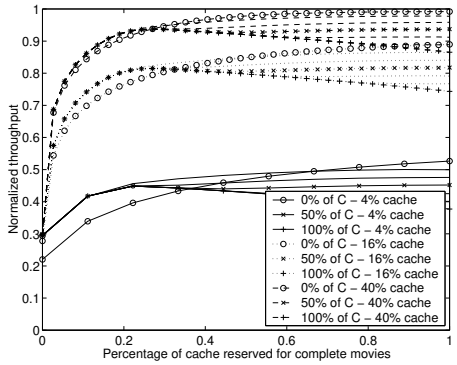

Fig. 10. Normalized throughput for partial caching and trunk reservation with $C = 150$ Mbps

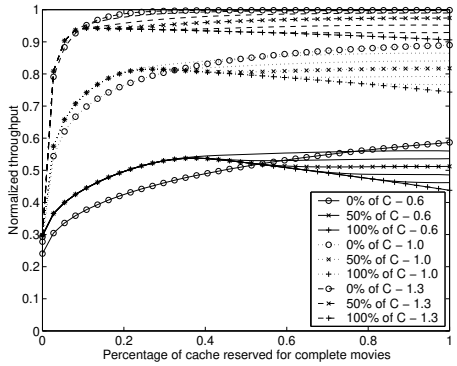

Fig. 11. Normalized throughput for partial caching and trunk reservation with different Zipf parameters

cached. As we have only one traffic class, this case can be analyzed using the Erlang–B formula with the number of trunks being C/r_e. For the other cases with the link reservations between 0 to 100%, we use simulations to obtain the throughput. In the figure we plot the normalized throughput for link reservations in an increment of 25 Mbps for a link bandwidth $C = 150$ Mbps.

The results confirm our intuition that once the base layers are cached, it is beneficial to reserve some bandwidth to give us an optimum throughput. For example, if we reserve 30% of the cache space for complete movies, which also means that we reserve 70% of the cache for base layers, then reserving any amount of bandwidth for streaming class 2 movies will give us better throughput than complete sharing. However, the figure shows that the maximum throughput for all cases is always achieved when we use the entire cache for caching complete movies. In this case, there is no class 2 streams and thus, the link is used exclusively for streaming the class 3 movies.

Fig. 11 shows the effect of varying the popularity of the movies. We observe that the proportion of the cache space that needs to be reserved to achieve the optimum throughput for the partial caching case changes with the Zipf parameter. This makes it harder to dimension the cache properly to achieve the optimum throughput at all times. Considering this difficulty and the fact that reserving the entire cache for caching complete movies give the maximum throughput, our experiments indicate that the partial caching is not beneficial.

VII. Related Work

There are only few studies on distributing video objects with caches, all of which are complementary to the issues studied in this paper. Rejaie *et al*. propose a proxy caching mechanism [11] in conjunction with a congestion control mechanism [12, 13] for layered–encoded video. The basic idea of their caching mechanism is to cache segments of layers according to the objects' popularities: the more popular an object, the more complete are the individual layers cached and the more layers are cached (partially). When streaming an object to a client, the layer segments that are not cached at the proxy are obtained from the origin server.

A related idea is explored by Wang *et al*. in their study on video staging [14]. With video staging the part of the VBR video stream, that exceeds a certain cut–off rate (i.e., the bursts of a VBR stream) is cached at the proxy while the lower (now smoother) part of the video stream is stored at the origin server.

Sen *et al*. [15] propose to cache a prefix (i.e., the initial frames) of video streams at the proxy and to employ work–ahead smoothing while streaming the object from the proxy to the client. The cached prefix hides the potentially large initial start–up delay of the work–ahead transmission schedule from the client.

Tewari *et al*. [16] propose a Resource Based Caching (RBC) scheme for video objects encoded into one CBR layer. They model the cache as a two resource (storage space and bandwidth) constrained knapsack and study replacement policies that take the objects' sizes as well as CBR bandwidth into account. The replacement policies are evaluated through simulations. Our work differs from RBC in that we develop an *analytical* stochastic knapsack model for the two resource problem. Moreover, we analyze a streaming system where videos are encoded into multiple layers.

VIII. Conclusion

In this paper we have formulated an analytical stochastic knapsack model for the layered video caching problem. We have proposed three different heuristics for determining which layers of which videos to cache. Through extensive numerical experiments we have found that all our heuristics perform well and that the best performance is obtained with the revenue density heuristic. Our heuristics are useful for cache operators in both provisioning the

caching system as well as deciding on-line the gain from caching a given layer of a given video. To the best of our knowledge, this is the first study to consider an analytical model of this 2-resource problem.

We also considered two intuitive extensions, renegotiation and queueing of requests, but found that they provide little extra gain to the cache operator. As a special case we considered a situation where clients only request complete video streams. Our results indicate that in this special case, best performance is obtained if videos are cached completely.

APPENDIX

I. CALCULATION OF BLOCKING PROBABILITIES $B_{\mathbf{c}}(j,m)$

In this appendix we give an overview of the calculation of the blocking probabilities $B_{\mathbf{c}}(j,m)$, which are non–zero for $c_m < j$. We calculate the blocking probabilities using results from the analysis of multiservice loss models. We refer the interested reader to [8] for a detailed discussion of this analysis. We model the bottleneck link for continuous media streaming from the origin servers to the proxy server as a stochastic knapsack of capacity C. We model requests for j–quality streams of object m as a distinct class of requests. Let $\boldsymbol{b}_{\mathbf{c}} = (b_{\mathbf{c}}(j,m)),\ m = 1,\ldots,M,\ j = 1,\ldots,L$, be the vector of the sizes of the requests. Note that this vector has ML elements. Recall that a request for a j–quality stream of object m of which the c_m–quality stream is cached requires the bandwidth $\sum_{l=c_m+1}^{j} r_l(m)$ on the bottleneck link; hence $b_{\mathbf{c}}(j,m) = \sum_{l=c_m+1}^{j} r_l(m)$ for $c_m < j$ and $b_{\mathbf{c}}(j,m) = 0$ for $c_m \geq j$. Without loss of generality we assume that C and all $b_{\mathbf{c}}(j,m)$'s are positive integers. Let $\boldsymbol{n} = (n(j,m)),\ m = 1,\ldots,M,\ j = 1,\ldots,L$, be the vector of the numbers of $b_{\mathbf{c}}(j,m)$–sized objects in the knapsack. The $n(j,m)$'s are non–negative integers. Let $\mathcal{S}_{\mathbf{c}} = \{\boldsymbol{n} : \boldsymbol{b}_{\mathbf{c}} \cdot \boldsymbol{n} \leq C\}$ be the state space of the stochastic knapsack, where $\boldsymbol{b}_{\mathbf{c}} \cdot \boldsymbol{n} = \sum_{m=1}^{M} \sum_{j=1}^{L} b_{\mathbf{c}}(j,m) n(j,m)$. Furthermore, let $\mathcal{S}_{\mathbf{c}}(j,m)$ be the subset of states in which the knapsack (i.e., the bottleneck link) admits an object of size $b_{\mathbf{c}}(j,m)$ (i.e., a stream of rate $\sum_{l=c_m+1}^{j} r_l(m)$). We have $\mathcal{S}_{\mathbf{c}}(j,m) = \{\boldsymbol{n} \in \mathcal{S}_{\mathbf{c}} : \boldsymbol{b}_{\mathbf{c}} \cdot \boldsymbol{n} \leq C - b_{\mathbf{c}}(j,m)\}$ The blocking probabilities can be explicitly expressed as

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

where $\rho(j,m) = \lambda p(j,m) T(m)$. Note that $\rho(j,m)$ is the load offered by requests for j–quality streams of object m. The blocking probabilities can be efficiently calculated using the recursive Kaufman–Roberts algorithm [8, p. 23]. The time complexity of the algorithm is $O(CML)$. The complexity is linear in the bandwidth C of the bottleneck link and the number of objects M, which can be huge. The complexity is also linear in the number of encoding layers L, which is typically small (2 – 5).

REFERENCES

[1] J. Lee, T. Kim, and S. Ko, "Motion prediction based on temporal layering for layered video coding," in *Proc. of ITC–CSCC, Vol. 1*, July 1998.

[2] S. McCanne and M. Vetterli, "Joint source/channel coding for multicast packet video," in *Proc. of IEEE International Conference on Image Processing*, Oct. 1995.

[3] M. Vishwanath and P. Chou, "An efficient algorithm for hierarchical compression of video," in *Proc. of IEEE International Conference on Image Processing*, Nov. 1994.

[4] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," in *Proc. of First International Workshop on Intelligent Multimedia Computing and Networking (IMMCN)*, Atlantic City, NJ, Feb. 2000.

[5] Y. Birk, "Random RAIDs with selective exploitation of redundancy for high performance video servers," in *Proc. of NOSSDAV*, St. Louis, Missouri, May 1997.

[6] D. J. Gemmel, H. M. Vin, D. D. Kandalur, P. V. Rangan, and L. A. Rowe, "Multimedia storage servers: A tutorial," *IEEE MultiMedia*, vol. 28, no. 5, pp. 40–49, May 1995.

[7] H. Schulzrinne, A. Rao, and R. Lanphier, "*RFC 2326: Real time streaming protocol (RTSP)*", Apr. 1998.

[8] K. W. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*, Springer–Verlag, 1995.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. of INFOCOM*, New York, NY, Mar. 21–25, 1999.

[10] V. N. Padmanabhan and L. Qiu, "The content and access dynamics of a busy web site: Findings and implications," in *Proc. of SIGCOMM*, Stockholm, Sweden, Aug. 28 – Sept. 1, 2000, (to appear).

[11] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proc. of INFOCOM*, Tel Aviv, Israel, Mar. 26–30, 2000.

[12] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proc. of SIGCOMM*, Cambridge, MA, Aug. 30 – Sept. 3, 1999.

[13] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end–to–end rate–based congestion control mechanism for realtime streams in the internet," in *Proc. of INFOCOM*, New York, NY, March 1999.

[14] Y. Wang, Z. Zhang, D. Du, and D. Su, "A network–conscious approach to end–to–end video delivery over wide area networks using proxy servers," in *Proc. of INFOCOM*, San Francisco, CA, April 1998.

[15] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. of INFOCOM*, New York, NY, March 1999.

[16] R. Tewari, H.M. Vin, A. Dan, and D. Sitaram, "Resource–based caching for web servers," in *Proc. of SPIE/ACM Conf. on Multimedia Computing and Networking*, San Jose, 1998.