Designs of High Quality Streaming Proxy Systems

Songqing Chen
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187, USA
sqchen@cs.wm.edu

Bo Shen, Susie Wee
Mobile and Media System Lab
Hewlett-Packard Laboratories
Palo Alto, CA 94304, USA
{boshen, swee}@hpl.hp.com

Xiaodong Zhang
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187, USA
zhang@cs.wm.edu

***Abstract*— Researchers often use segment-based proxy caching strategies to deliver streaming media by partially caching media objects. The existing strategies mainly consider increasing the byte hit ratio and/or reducing the client perceived startup latency (denoted by the metric delayed startup ratio). However, these efforts do not guarantee continuous media delivery because the to-be-viewed object segments may not be cached in the proxy when they are demanded. The potential consequence is playback *jitter* at the client side due to proxy delay in fetching the uncached segments, which we call *proxy jitter*. Thus, for the best interests of clients, a correct model for streaming proxy system design should aim to minimize proxy jitter subject to reducing the delayed startup ratio and increasing the byte hit ratio. However, we have observed two major pairs of conflicting interests inherent in this model: (1) one between improving the byte hit ratio and reducing proxy jitter, and (2) the other between improving the byte hit ratio and reducing the delayed startup ratio. In this study, we first propose an active prefetching method for in-time prefetching of uncached segments, which provides insights into the first pair of conflicting interests. Second, we further improve our lazy-segmentation scheme [1] which effectively addresses the second pair of the conflicting interests. Finally, considering our main objective of minimizing proxy jitter and optimizing the two trade-offs, we propose a new streaming proxy system called *Hyper Proxy* by effectively coordinating both prefetching and segmentation techniques. Synthetic and real workloads are used to systematically evaluate our system. The performance results show that the Hyper Proxy system generates minimum proxy jitter with a low delayed startup ratio and a small decrease of byte hit ratio compared with existing schemes.**

Keywords: System Design, Content Distribution, Streaming Media, Proxy Caching

I. Background and Motivation

Proxy caching has been widely used to cache static (text/image) objects on the Internet so that subsequent requests to the same objects can be served directly from the proxy without contacting the server. However, the proliferation of multimedia content makes caching challenging (e.g. [2], [3], [4], [5], [6], [7], [8], [9]) due to the typical large size and the low-latency and continuous streaming demand of media objects.

To solve the problems caused by large-sized media objects, researchers have developed a number of segment-based proxy caching strategies (e.g. [1], [8], [10], [11], [12], [13], [14]) that cache partial segments of media objects instead of their entirety. The existing segment-based proxy caching strategies can be classified into the following two types based on their performance objectives. The first type focuses on the reduction of the client perceived startup latency (denoted by the delayed startup ratio) by always giving a higher priority to caching the beginning segments of media objects based on the observation [15], [16] that clients tend to watch the beginning portions. For example, prefix caching [8], [17] breaks the media object into prefix segment and suffix segment. The proxy caches the prefix segments only so that the cache can preserve prefix segments for more objects. The second type aims at reducing network traffic and server workload by improving proxy caching efficiency, namely the byte hit ratio. For example, uniform segmentation strategy [10] considers caching of fixed-sized segments of layer-encoded video objects. The exponential segmentation strategy [12], [13] caches segments of media objects in a way that the succeeding segment doubles the size of the preceding one. The most recently proposed adaptive-lazy segmentation strategy [1] can achieve the highest byte hit ratio by delaying the object segmentation as late as possible till some real time access information is collected for this object. Some of these caching strategies also consider the client perceived startup latency while emphasizing on the proxy caching efficiency.

However, these segment-based proxy caching strategies can not automatically ensure continuous streaming delivery to the client. In a segment-based proxy caching system, since only partial segments of objects are cached in the proxy, it is important for the proxy to fetch and relay the uncached segments in time whenever necessary. A delayed fetch of the uncached segments, which we call *proxy jitter*, causes the discontinuous delivery of media content. Proxy jitter aggregates onto the playback jitter at the client side. Once a playback starts, jitter is not only annoying but can also potentially drive the user away from accessing the content. Thus, for the best interests of clients, the highest priority must be given to minimize proxy jitter, and a correct model for media proxy cache design should aim to minimize proxy jitter subject to reducing the delayed startup ratio and increasing the byte hit ratio.

To reduce proxy jitter, one key is to develop prefetching schemes that can timely prefetch uncached segments. Some early work has studied the prefetching of multimedia objects (e.g. [10], [11], [18], [19]). For layer-encoded objects [10], [11], the prefetching of uncached layered video is done by always maintaining a prefetching window of the cached stream, and identifying and prefetching all the missing

data within the prefetching window with a fixed time period (length of T) ahead of their playback time. In [19], the proactive prefetch utilizes any partially fetched data due to the connection abortion to improve the network bandwidth utilization. In [18], prefetching is used to prefetch a certain amount of data so that caching is feasible. Unfortunately, little prefetching work has been found to efficiently solve the proxy jitter problem in the context of segment-based proxy caching.

Improving the byte hit ratio increases proxy caching efficiency, while reducing proxy jitter provides clients with a continuous streaming service. Unfortunately, these two objectives conflict with each other. Furthermore, we have also observed that improving the byte hit ratio conflicts with reducing the delayed startup ratio [1]. These three conflicting objectives form two pairs of trade-offs that complicate the design model. No previous work has been found to address the balancing of these trade-offs, which are uniquely important to streaming media proxy systems.

In this study, we first propose an active prefetching method for the in-time prefetching of uncached segments, which not only gives an effective solution to address the proxy jitter problem, but also provides insights into the trade-off between improving the byte hit ratio and reducing proxy jitter. Second, we further improve our lazy-segmentation scheme [1], which effectively addresses the conflicting interests between reducing startup latency and improving byte hit ratio. Finally, considering our main objective of minimizing proxy jitter and balancing the two trade-offs, we propose a new media proxy caching system called *Hyper Proxy* by effectively coordinating these two schemes. Hyper Proxy depends on the HTTP channel for prefetching, while it interfaces with clients in a RTP [20]/RTSP [21] streaming channel. Synthetic and real workloads are used to systematically evaluate the system. The performance results show that the Hyper Proxy system generates minimum proxy jitter with a low delayed startup ratio and a small decrease of byte hit ratio compared with existing schemes. Our study also indicates that the standard objective of improving the byte hit ratio commonly used in proxy caching is not suitable to streaming media delivery.

The paper is organized as follows. We propose the active prefetching method and provide insights into proxy jitter in Section II. The second pair of conflicting interests is addressed in Section III. The Hyper Proxy system is presented in Section IV. We evaluate it in Section V. Some related work is introduced in Section VI and we make concluding remarks in Section VII.

II. Active Prefetching Method and Insights into Proxy Jitter

Prefetching schemes can reduce proxy jitter by fetching uncached segments before they are accessed. However, an efficient prefetching method should consider the following two conflicting interests in the proxy. On one hand, proxy jitter occurs if the prefetching of uncached segments is delayed. To avoid jitter, the proxy should prefetch uncached segments as early as possible. On the other hand, aggressive prefetching of uncached segments requires extra network traffic and storage space to temporarily store the prefetched data. Even worse, the client session may terminate before the prefetched segments are accessed. This observation indicates that the proxy should prefetch uncached segments as late as possible. This contradiction requires that the proxy accurately decides when to prefetch which uncached segment in a way to minimize the proxy jitter as well as to minimize the resource usage (network and storage). In this section, we propose a prefetching method, called *active prefetching*, which jointly considers both objectives. Our subsequent analysis further provides the insights into the conflicting interests between reducing proxy jitter and improving the byte hit ratio.

A. Active Prefetching

The objective of our active prefetching is to determine when to prefetch which uncached segment so that proxy jitter is minimized with the minimum amount of resource requirement. In our analysis, the following assumptions are made.

- The object has been segmented and is accessed sequentially;
- The bandwidth of the proxy-client link is large enough for the proxy to stream the content to the client smoothly; and
- Each segment of the object can be fetched from the server (either the origin server or a cooperative one) in a unicast channel.

Since the prefetching is segment based, several related notations used in the discussion are listed in Table I. Note

B_s	the average encoding rate of a certain object segment
B_t	the average network bandwidth of the proxy-server link
k	the total number of segments of the object
n	the number of cached segments of the object
S_i	the i^{th} segment of the object
L_i	the length of the i^{th} segment
L_b	the base segment length of the object, $L_b = L_1$

TABLE I
The Notations for Active Prefetching

that each media object has its inherent encoding rate, which is the playback rate. The rate is not a constant in variable bit rate video, but we use B_s to denote its average value. B_t may vary dynamically when different segments are accessed. The proxy monitors B_t by keeping records of the data transmission rate of the most recent prior session with the same server. The transmission rate is calculated by dividing the amount of transferred data by the transferring duration.

For a requested media object, assume there are n segments cached in the proxy. The goal is to determine when to schedule the prefetching of S_{n+1} so that proxy jitter is avoided. We denote the scheduling point as x.

Note that prefetching is not necessary when $B_s \leq B_t$, so the following discussion is based on $B_s > B_t$. At position x, the length of the to-be-delivered data from the cache is $\sum_{i=1}^{i=n} L_i - x$. To avoid proxy jitter, the time that the proxy

takes to prefetch S_{n+1} must not exceed the time that the proxy takes to deliver the rest of the cached data and the fetched data. That is, the following condition must be satisfied to avoid proxy jitter:

$$\frac{\sum_{i=1}^{i=n} L_i - x + L_{n+1}}{B_s} \geq \frac{L_{n+1}}{B_t}.$$

Therefore, the latest prefetching scheduling point to avoid proxy jitter is:

$$x = \sum_{i=1}^{i=n} L_i - \frac{L_{n+1} \times (B_s - B_t)}{B_t}. \quad (1)$$

Refer back to our objectives, when x is selected as the prefetching scheduling point, the buffer size required for the prefetched data reaches minimum:

$$\frac{\sum_{i=1}^{i=n} L_i - x}{B_s} \times B_t. \quad (2)$$

We now discuss the active prefetching method for two typical segment-based caching schemes by first determining the prefetching scheduling point and then discussing the prefetching scheme and resource requirements.

1) Active Prefetching for Uniformly Segmented Object: For a uniformly segmented object, $L_i = L_1$, based on Equation 1, we have the latest scheduling points x as

$$x = (n+1)L_1 - \frac{B_s}{B_t}L_1. \quad (3)$$

Equation 3 states that if $n+1 \geq \frac{B_s}{B_t}$, in-time prefetching of S_{n+1} is possible with the minimum required buffer size of

$$L_1 \times \frac{B_s - B_t}{B_s}. \quad (4)$$

However, Equation 3 also indicates that if $n+1 < \frac{B_s}{B_t}$, in-time prefetching of S_{n+1} is not possible! Therefore, when $n+1 < \frac{B_s}{B_t}$ and the segments between $n+1^{th}$ and $\lceil \frac{B_s}{B_t} \rceil^{th}$ are demanded, proxy jitter is inevitable. To minimize future proxy jitter under this situation, the proxy needs to prefetch the $\lceil \frac{B_s}{B_t} \rceil^{th}$ segment instead of the $n+1^{th}$ segment.

For uniformly segmented objects, active prefetching works as follows:

- $n = 0$: No segment is cached. Proxy jitter (in this case, startup latency) is inevitable. To avoid future proxy jitter, prefetching of the $\lceil \frac{B_s}{B_t} \rceil^{th}$ segment is necessary. The minimum buffer size required is $(1 - \frac{B_t}{B_s})L_1$.
- $n > 0$ and $n + 1 < \frac{B_s}{B_t}$: The proxy starts to prefetch the $\lceil \frac{B_s}{B_t} \rceil^{th}$ segment once the client starts to access the object. If the segments between $n+1^{th}$ and $\lceil \frac{B_s}{B_t} - 1 \rceil^{th}$ are demanded, they are fetched on demand, and proxy jitter is inevitable. The minimum buffer size required is $(1 - \frac{B_t}{B_s})L_1$.
- $n > 0$ and $n + 1 \geq \frac{B_s}{B_t}$: The prefetching of S_{n+1} is scheduled when the streaming reaches the position of $(n + 1 - \frac{B_s}{B_t})L_1$ of the first n cached segments. Proxy jitter can be completely eliminated in this case, and the minimum buffer size required is $(1 - \frac{B_t}{B_s})L_1$.

2) Active Prefetching for Exponentially Segmented Object: Through similar analysis, active prefetching for exponentially segmented objects works as follows. Here, we assume $B_s \leq 2B_t$. When $B_s \geq 2B_t$, no prefetching of the uncached segments can be in time for the exponentially segmented objects.

- $n = 0$: No segment is cached. Proxy jitter (in this case, startup latency) is inevitable. To avoid future proxy jitter, the prefetching of the $\lceil 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}}) \rceil^{th}$ segment is necessary once the client starts to access the object. The minimum buffer size required is $L_1 \times \frac{{B_t}^2}{2 \times B_s \times B_t - {B_s}^2}$.
- $n > 0$ and $n \leq log_2(\frac{1}{2 - \frac{B_s}{B_t}})$: The proxy starts to prefetch the $\lceil 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}}) \rceil^{th}$ segment once the client starts to access this object. Proxy jitter is inevitable when the client accesses data of the $n+1^{th}$ segment to the $\lceil 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}}) \rceil^{th}$ segment. The minimum buffer size is $L_i B_t / B_s$, where $i = \lceil 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}}) \rceil$.
- $n > 0$ and $n > log_2(\frac{1}{2 - \frac{B_s}{B_t}})$: The prefetching of the $n+1^{th}$ segment starts when the client accesses to the $1 - \frac{2^n}{2^n - 1} \times (\frac{B_s}{B_t} - 1)$ portion of the first n cached segment. The minimum buffer size is $L_{n+1} \times \frac{B_t}{B_s}$ and increases exponentially for later segments.

Our proposed active prefetching method gives the optimal prefetching scheduling point whenever possible with minimum resource usage. However, under certain conditions the prefetching of uncached segments may still be delayed as our analysis showed, for both uniformly and exponentially segmented objects. Furthermore, the analysis also finds that the uniformly segmented object has advantages over the exponentially segmented object: it offers enhanced capability for in-time prefetching and the in-time prefetching can always begin in a later stage.

B. Segment-based Proxy Caching and Proxy Jitter Free Strategies

The previous section shows that active prefetching can not always guarantee continuous media delivery, which is one of the most important objectives for the streaming delivery. However, for any caching strategy, if there are always enough number of segments being cached in the proxy, prefetching of the uncached segments can always be in time. To evaluate this situation, we define *prefetching length* as follows:

- *prefetching length*: the minimum length of data that must be cached in the proxy in order to guarantee the continuous delivery when $B_s > B_t$. We denote m as the number of segments with the aggregated length equal to the prefetching length.

In-time prefetching must guarantee, in the worst case, that the prefetching of the rest of segments is completed before the delivery of the whole object, that is:

$$\frac{\sum_{i=1}^{i=k} L_i}{B_s} \geq \frac{\sum_{i=1}^{i=k} L_i - \sum_{i=1}^{i=m} L_i}{B_t}. \quad (5)$$

This indicates that the following condition must be satisfied to guarantee in-time prefetching:

$$\sum_{i=1}^{i=m} L_i \geq \sum_{i=1}^{i=k} L_i(1 - \frac{B_t}{B_s}). \quad (6)$$

- In the uniform segmentation scheme, since L_b is the base segment length, the minimum m to satisfy the above condition is:

$$m = \lceil \frac{(1 - \frac{B_t}{B_s}) \sum_{i=1}^{i=k} L_i}{L_b} \rceil. \quad (7)$$

- In the exponential-segmentation scheme, [1], since $L_i = 2L_{i-1}$, the minimum m to satisfy the condition is:

$$m = \lceil log_2(\frac{(1 - \frac{B_t}{B_s}) \sum_{i=1}^{i=k} L_i}{L_b}) \rceil + 1. \quad (8)$$

Interested readers are referred to [22] for more detailed derivation.

C. Trade-off Between Low Proxy Jitter and High Byte Hit Ratio

We have calculated the minimum number of segments that must always be cached in the proxy to guarantee a continuous delivery of the streaming media object. Thus we can estimate how much cache space we need to guarantee a proxy-jitter-free delivery. However, in practice, we always have limited cache space and can not cache all these segments for each object.

In an actual segment-based proxy caching system, popular objects are always cached to reduce network traffic and server load. If an object is popular enough, all its segments can be cached in the proxy, possibly larger than its prefetching length. If an object is not popular enough, some segments may get evicted and only a few of its segments are cached. The aggregated length of these segments may be less than its prefetching length, which causes proxy jitter when the uncached segments are demanded by the client. Given a higher priority in reducing the proxy jitter, the proxy can choose to evict segments of the object whose cached data length is larger than its prefetching length. The released cache space can be used to cache more segments of the object whose cached data length is less than its prefetching length so that the prefetching of its uncached segment can always be in time. It is possible that segments of popular objects are evicted, which may reduce the byte hit ratio. However, since there are more objects with enough segments cached to avoid delayed prefetching, overall proxy jitter is reduced. From this example, we can see that the byte hit ratio can be traded for less proxy jitter.

The insights of the conflicting interests between improving the byte hit ratio and reducing proxy jitter have motivated us to revise the principle to design a highly effective proxy caching system, aiming to minimize the proxy jitter. However, in the design model, we have also observed that segment-based proxy caching strategies always perform well in the byte hit ratio, but perform not so well in the delayed startup ratio, or vice versa. The evaluation of the adaptive-lazy segmentation based scheme provides such a case study for this observation, as in [1]. We must understand these insights before we can design a correct system according to the design model. We formalize the problem and mathematically analyze this trade-off in the next section.

[1]To be consistent with the uniform segmentation case, the segments are counted from 1 instead of 0 as in [12].

III. Byte Hit Ratio vs. Delayed Startup Ratio

The observation in [1] leads us to conjecture that there are some conflicting interests between the objectives of improving the byte hit ratio and reducing the delayed startup ratio. In this section, we present our understanding and a method to effectively balance the trade-off.

A. Analytical Model

To formalize the problem, we build a general analytical model for the segment-based caching system. We assume:
(1) The popularity of the objects follows a Zipf-like distribution, which models the probability set p_i, where $p_i = \frac{f_i}{\sum_{i=1}^{i=N} f_i}$, (i =1, 2, ...,N, N is the total number of objects) and $f_i = \frac{1}{i^\theta}$, where $\theta > 0$ and is the skew factor;
(2) The request arrival interval process follows Poisson distribution with a mean arrival rate λ. The request arrival interval process to each individual object is independently sampled from the aggregate arrival interval process based on probability set p_i, where $\sum_{i=1}^{i=n} p_i = 1$;
(3) The clients view the requested objects completely. This is to simplify the analysis and does not affect the conclusion.

These assumptions indicate that the mean arrival rate for each object is:

$$\lambda_i = \lambda p_i = \lambda \times \frac{\frac{1}{i^\theta}}{\sum_{i=1}^{i=N} \frac{1}{i^\theta}}. \quad (9)$$

To evaluate the delayed startup ratio, we define the following notation:

- *startup length*: the length of the beginning part of an object. If this portion of the object is cached, no startup delay is perceived by clients when the object is accessed. We use α to denote the percentage of the startup length with respect to the full object length. [2].

Other notations used in the discussion are listed below:

- L_{obj}^i: the full length of the i^{th} object, where $1 \leq i \leq N$;
- L_{obj}^{ave}: the average length of the objects;
- C: the total cache size;
- β: the percentage of total cache space reserved for the caching of first α percent (startup length) of objects;
- C_{prefix}: the size of reserved cache space for caching startup length of objects. $C_{prefix} = \beta C$;
- C_{rest}: the size of the cache space other than the space reserved for caching of startup length of objects. $C_{rest} = C - C_{prefix} = (1 - \beta)C$.

[2]Note that instead of caching the first α percent, caching a constant length of the prefix segment for each object leads to the same results.

Consider the ideal case that the cache space is always allocated to cache the most popular objects. If we sort the objects according to their decreasing popularities, the ideal case indicates that C_{prefix} is used to cache the segments (within startup length) of the first t most popular objects. That is,

$$\sum_{i=1}^{i=t} \alpha L_{obj}^{i} \leq C_{prefix} \quad \text{and} \quad \sum_{i=1}^{i=t+1} \alpha L_{obj}^{i} > C_{prefix}.$$

Similarly, the ideal case indicates that C_{rest} is used to cache the segments (beyond startup length) of the first q most popular objects. That is,

$$\sum_{i=1}^{i=q} (1-\alpha) L_{obj}^{i} \leq C_{rest} \quad \text{and} \quad \sum_{i=1}^{i=q+1} (1-\alpha) L_{obj}^{i} > C_{rest}.$$

Under this ideal circumstance, the delayed startup ratio can be expressed as:

$$P_{delay} = \frac{\sum_{i=t+1}^{i=N} \lambda_i}{\sum_{i=1}^{i=N} \lambda_i}. \tag{10}$$

Denoting $U = \frac{C}{L_{obj}^{ave}}$, we can derive that $t = \beta U/\alpha$. The upper bound of the delayed startup ratio is obtained when $\theta \neq 1$:

$$P_{delay}^{Max} = \frac{N^{1-\theta} - (U \times \frac{\beta}{\alpha})^{1-\theta}}{(N+1)^{1-\theta} - 1}. \tag{11}$$

Now we derive an upper bound of the byte hit ratio in the ideal case. Without considering the misses when the object is accessed for the first time, the corresponding byte hit ratio is expressed as:

$$P_{hit} = 1 - \frac{\sum_{i=t+1}^{i=N} \alpha \lambda_i L_{obj}^{i} + \sum_{i=q+1}^{i=N} (1-\alpha) \lambda_i L_{obj}^{i}}{\sum_{i=1}^{i=N} \lambda_i L_{obj}^{i}}. \tag{12}$$

Again, given $t = \beta U/\alpha$, the upper bound of the byte hit ratio is obtained when $\theta \neq 1$:

$$P_{hit}^{Max} = 1 - \left(\frac{(N+1)^{1-\theta} - \alpha\left(\frac{\beta}{\alpha} \times U + 1\right)^{1-\theta}}{N^{1-\theta} - \theta} - \frac{(1-\alpha)\left(\frac{1-\beta}{1-\alpha} \times U + 1\right)^{1-\theta}}{N^{1-\theta} - \theta}\right). \tag{13}$$

B. Analytical Results

To give an intuition into the dynamic nature of the two performance objectives, an example is given in Figure 1 based on Equation 11 and Equation 13. Here, given a total of 10000 original objects, we assume a cache size 20% of the total object size. Thus, U is set as 2000 object units. Furthermore, θ and α are set as 0.47 and 5%, respectively. As shown in the figure, the decrease of the byte hit ratio is much slower than the decrease of the delayed startup ratio when β increases. Therefore, we can use a small decrease of byte hit ratio to trade for a significantly large reduction in the delayed startup ratio.

Mathematically, the partial derivative of P_{delay}^{Max} with respective to β yields $|\Delta_{Delay}|$ which denotes the change of the delayed startup ratio. The partial derivative of P_{hit}^{Max} with respective to β yields $|\Delta_{Hit}|$ which denotes the change of the byte hit ratio. Therefore, we have

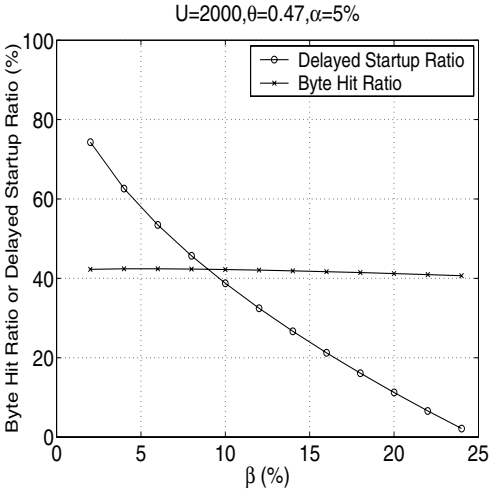

Fig. 1. Byte Hit Ratio vs. Delayed Startup Ratio

$$\frac{|\Delta_{Delay}|}{|\Delta_{Hit}|} = \frac{1}{\alpha} \times \frac{\frac{N^{1-\theta}-\theta}{(N+1)^{1-\theta}-1}}{\left(\frac{1-\beta}{1-\alpha} \times \frac{\alpha}{\beta} + \frac{\alpha}{U\beta}\right)^{-\theta} + \left(\frac{\alpha}{U\beta} + 1\right)^{-\theta}}. \tag{14}$$

It can be shown that $|\Delta_{Delay}|/|\Delta_{Hit}|$ is always greater than 1 when α and β are less than 50%. For a long but complete derivation, please refer to [14].

The above analysis provides us with a solid basis to restructure the adaptive-lazy segmentation strategy in [1] by giving a higher priority to caching the startup length of objects in the replacement policy. The objective is to effectively address the conflicting interests between improving the byte hit ratio and reducing the delayed startup ratio for the best quality of media delivery. The analysis leads to the following improved replacement policy design.

C. Improved Adaptive-Lazy Segmentation Strategy

In order to significantly reduce the startup latency with a small decrease of the byte hit ratio as suggested by our previous analysis result, a three-phase iterative replacement policy is re-designed as follows.

Based on a utility function defined similarly as in [1], upon an object admission, if there is not enough cache space, the proxy selects the object with the smallest utility value at that time as the victim, and the segment of this object is evicted in one of the two phases as follows. (1) First Phase: If the object is fully cached, the object is segmented by the lazy segmentation method [1]. The first 2 segments are kept and the remaining segments are evicted right after the segmentation is completed. (2) Second Phase: If the object is partially cached with more than 1 segment, the last cached segment of this object is evicted. (3) Third Phase: If the victim has only the first segment and is to-be-replaced, then its startup length and the base segment length, L_b, is compared. If its startup length is less than the base segment length, the startup

length is kept and the rest is replaced. Otherwise, it will be totally replaced. The utility value of the object is updated after each replacement and this process repeats iteratively until the required space is found.

This restructured adaptive and lazy segmentation strategy has shown its effectiveness in [14] by well balancing the two performance objectives. Interested reader can refer to [14].

IV. The Hyper Proxy System

Coordinating the two schemes in previous sections, we design a high quality media streaming proxy system, called Hyper Proxy system. In our design, for any media object accessed through the proxy, a data structure containing the following items in Table II is created and maintained. This data structure is called the access log of the object.

T_1	the time instance the object is firstly accessed
T_r	the last reference time of the object
T_c	the current time instance
L_{sum}	the sum of each access duration to the object
n_a	the number of accesses to the object
L_b	the length of the base segment
n	the number of the cached segments of the object
FG_{adm}	the admission flag for admitting segments
L_{thd}	the threshold length used in the replacement policy
L_{avg}	the average access duration of an object
F	the access frequency

TABLE II

Items of the Hyper Proxy Data Structure for Each Object

For each object, the L_{thd} is calculated after the segmentation (see section IV-C). It is equal to $\max$*(startup length, prefetching length,* $2L_b$) and its value varies due to the dynamic nature of B_s and B_t. In the system, two object lists (*premium list* and *basic list*) are maintained. The *basic list* contains all the objects whose length of cached segments is larger than its L_{thd} while the *premium list* contains all the objects whose cached data length is equal to or less than its L_{thd}. FG_{adm} is the flag used to indicate the priority of new segment admission. Items L_{avg} and F can be derived from the items above. They are used as measurements of access activities to each object. At time instance T_c, the access frequency F is $n_a/(T_r - T_1)$, and the average access duration L_{avg} is L_{sum}/n_a.

When an object is accessed for the first time, it is fully cached and linked to the *basic list* according to the admission policy. A fully cached object is kept in the cache until it is chosen as an eviction victim according to the replacement policy. At that time, the object is segmented and some of its segments are evicted. The object is also transferred to the *premium list*. Once the object is accessed again, the proxy uses the active prefetching method to determine when to prefetch which uncached segment. Then the segments of the object are adaptively admitted by the admission policy or adaptively replaced by the replacement policy.

We now present the detailed description of four major modules in the Hyper Proxy caching system.

A. Priority-based Admission Policy

For any media object, cache admission is considered whenever the object is accessed.

- A requested object with no access log indicates that the object is accessed for the first time. The object is then cached in full regardless of the request's accessing duration. The replacement policy (see section IV-D) is activated if there is not sufficient space. The victim is selected from objects in the *basic list*, or *premium list* when the *basic list* is empty. In the *premium list*, objects with *PRIORITY* flag are searched if no object with *NON-PRIORITY* flag is in *premium list*. The fully cached object is linked to the *basic list* and an access log is created for the object and the recording of the access history begins. If an access log exists for the object (not the first access to the object), but the log indicates that the object is fully cached, the access log is updated. No other action is necessary.
- If an access log exists for the object, and its FG_{adm} is *PRIORITY* (see section IV-B), the proxy considers the admission of the next uncached segment or segments determined by its *prefetching length*. Whether the segment(s) can be admitted or not depends on if the replacement policy can find a victim or not. Victim selection is limited to objects in the *basic list* or *premium list* with *NON-PRIORITY* flag if *basic list* is empty. Note that for this admission, the system does not need to compare the caching utility value of this object with the victim's, but only to find a victim with the smallest utility value.
- If an access log exists for the object, and its FG_{adm} is *NON-PRIORITY* (see section IV-B), the next uncached segment is considered for admission only if $L_{avg} \geq (n+1)L_b/L_{thd}$. (Note L_{avg} is changing dynamically.) The inequality indicates that the average access duration is increasing to the extent that the cached n segments can not cover most of the requests while a total of $n+1$ segments can. Whether the next uncached segment is eventually admitted or not depends on whether or not the replacement policy can find a victim whose caching utility is less than this object. The victim selection is limited to the *basic list* only.

After the admission, the object will be transferred to the *basic list* if it is in the *premium list* and its cached data length is larger than its L_{thd}.

In summary, using the priority-based admission, the object is fully admitted when it is accessed for the first time. Then the admission of this object is considered segment by segment with the higher priority given to the admissions that are necessary for in-time prefetching.

B. Active Prefetching

After the object is segmented and some of its segments are replaced (see section IV-D), the object becomes partially cached. Then, upon each subsequent access, active prefetching

is activated to determine when to prefetch which segment once the object is accessed according to the following various conditions.

- $n = 0$: No segment is cached. The prefetching of the $\lceil \frac{B_s}{B_t} \rceil^{th}$ segment is considered. The FG_{adm} of this object is set to be *PRIORITY*.
- $n > 0$ and $n + 1 < \frac{B_s}{B_t}$: The proxy starts to prefetch the $\lceil \frac{B_s}{B_t} \rceil^{th}$ segment once the client starts to access the object. If the segments between $n + 1^{th}$ and $\lceil \frac{B_s}{B_t} - 1 \rceil^{th}$ are demanded, proxy jitter is inevitable and the FG_{adm} of this object is set to be *PRIORITY*.
- $n > 0$ and $n + 1 \geq \frac{B_s}{B_t}$: The prefetching of $n + 1^{th}$ segment starts when the client accesses to the position of $(n + 1 - \frac{B_s}{B_t})L_b$ of the first n cached segments. The FG_{adm} of this object is set to be *NON-PRIORITY*.

Note that B_s and B_t are sampled when each segment is accessed. As a result, L_{thd} is also updated accordingly.

C. Lazy Segmentation Policy

The key of the lazy segmentation strategy is as follows. Once there is no cache space available and replacement is needed, the replacement policy calculates the caching utility of each cached object (see section IV-D). Subsequently, the object with the smallest utility value is chosen as the victim if it is not active (no request is accessing it). If the victim object turns out to be fully cached, the proxy segments the object as follows. The average access duration L_{avg} at current time instance is calculated. It is used as the length of the base segment, that is, $L_b = L_{avg}$. Note that the value of L_b is fixed once it is determined. The object is then segmented uniformly according to L_b. After that, the first $\lceil \frac{L_{thd}}{L_b} \rceil$ segments are kept in cache, while the rest are evicted (see section IV-D). The number of cached segments, n, is updated in the access log of the object accordingly. This lazy segmentation scheme allows better determination of L_b.

D. Differentiated Replacement Policy

The replacement policy is used to re-collect cache space by evicting selected victims. First of all, a utility function (same as in [1]) is defined below to help the victim selection process by identifying the least valuable object as the victim.

$$\frac{F \times \frac{L_{sum}}{n_a} \times \min(1, \frac{\frac{T_r - T_1}{n_a}}{T_c - T_r})}{nL_b}, \tag{15}$$

In the above equation, the caching utility value is proportional to
(1) F, which estimates the average number of future accesses;
(2) $\frac{L_{sum}}{n_a}$, which estimates the average duration of future access;
(3) $\min(1, \frac{\frac{T_r - T_1}{n_a}}{T_c - T_r})$, which estimates the possibility of future accesses[3];
and inversely proportional to
(4) nL_b , which represents the disk space required.

Corresponding to the different situations of admission, when there is not enough space, the replacement policy selects the victim with the smallest utility value from different lists in the order as designated in section IV-A. Then partially cached data of the victim is evicted as follows.

- If the victim is fully cached in the *basic list*, the object is segmented as described in section IV-C. The first $\lceil \frac{L_{thd}}{L_b} \rceil$ segments are kept and the remaining segments are evicted right after the segmentation is completed. The object is removed from the *basic list* and linked to the *premium list*.
- If the victim is partially cached in the *basic list*, the last cached segment of this object is evicted. After the eviction, the object will be linked to the *premium list* if its cached data length is less than or equal to its L_{thd}. Note this object can be selected as victim again if no sufficient space is found in this round.
- If the victim is in the *premium list*, the last cached segment of this object is evicted. If no data of this object is cached, it is removed from the *premium list*.

The utility value of the object is updated after each replacement and this process repeats iteratively until the required space is found.

The design of the differentiated replacement policy gives a higher priority for reducing proxy jitter, reduces the erroneous decision of the replacement and gives fair chances to the replaced segments so that they can be cached back into the proxy again by the aggressive admission policy if they become popular again.

Note that after an object is fully evicted, the system still keeps its access log. If not, once the object is occasionally accessed again, it should be fully cached again. Since media objects tend to have diminishing popularities as the time goes on, if the system caches the object in full again, this results in an inefficient use of the cache space. Our design enhances the resource utilization by avoiding this kind of situation. By setting a large enough time-out threshold, the proxy deletes the access logs of unpopular objects eventually.

V. Performance Evaluation

A. Workload Summary

To evaluate the performance of the Hyper Proxy system, we conduct extensive simulations based on several workloads. Both synthetic workloads and a real workload extracted from enterprise media server logs are considered. We designed two synthetic workloads. These workloads assume a Zipf-like distribution with a skew factor θ for the popularity of the

[3]The system compares the $T_c - T_r$, the time interval between now and the most recent access, and the $\frac{T_r - T_1}{n_a}$, the average time interval between accesses occurring in the past. If $T_c - T_r > \frac{T_r - T_1}{n_a}$, the possibility that a new request arrives soon for this object is small. Otherwise, it is highly possible that a request is coming soon.

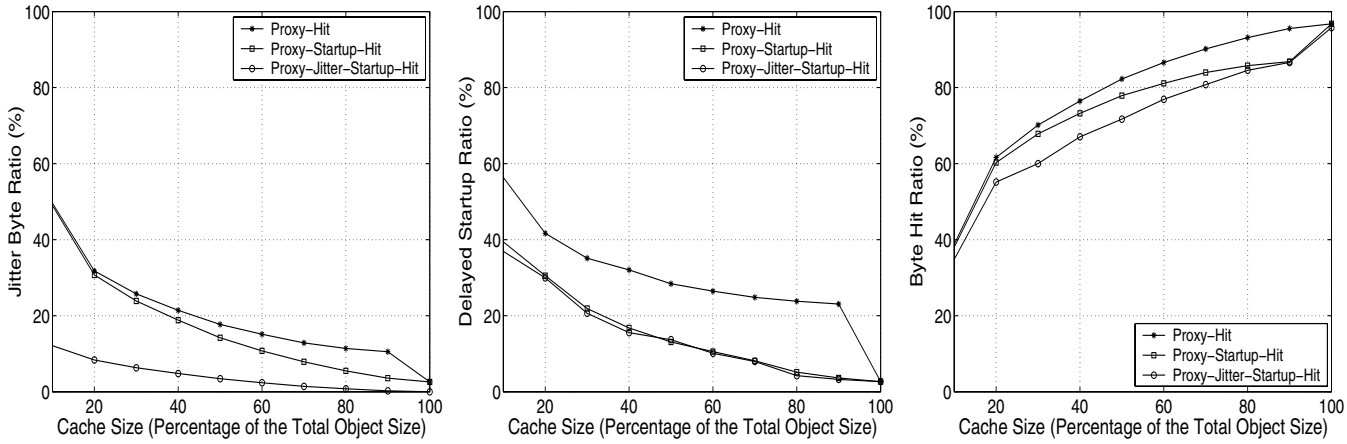

Fig. 2. WEB: (a) Proxy Jitter Bytes, (b) Delayed Startup Ratio, and (c) Byte Hit Ratio

media objects and request inter arrival follows the Poisson distribution with a mean interval λ.

The first synthetic workload simulates accesses to media object in the Web environment in which the length of the video varies from short ones to longer ones. We use WEB as the name of this workload. The second simulates the Web accesses where clients accesses to objects are incomplete, that is, a started session terminates before the full media object is delivered. We simulate this behavior by designing a partial viewing workload based on the WEB workload. We use PART as its name. In this workload, 80% of the sessions terminate before 20% of the object is delivered.

For the real workload named as REAL, we use logs from HP Corporate Media Solutions, covering the period from April 1 through April 10, 2001. There is a total of 403 objects, and the unique object size accounts to 20 GB. There is a total of 9000 requests during this period. Our analysis shows that about 83% of the requests only view the objects for less than 10 minutes and more than 56% of the requests only view less than 10% of their requested objects. About 10% of the requests view the whole objects.

Table III lists some characteristics of these workloads.

Workload Name	Num of Request	Num of Object	Size (GB)	λ	θ	Range (minute)	Duration (day)
WEB	15188	400	51	4	0.47	2-120	1
PART	15188	400	51	4	0.47	2-120	1
REAL	9000	403	20	-	-	6 - 131	10

TABLE III
THE WORKLOAD SUMMARY

B. Performance Results

In the simulation experiments, the streaming rate of accessed objects is set randomly in the range from half to four times that of the link capacity between the proxy and the server. We use the *jitter byte ratio* to evaluate the quality of the continuous streaming service provided by the proxy system. It is defined as the amount of data that is not prefetched in time by the proxy normalized by the total bytes demanded by the streaming sessions. Delayed prefetching causes potential playback jitter at the client side. A good proxy system should have small jitter byte ratio. The second metric we use is the *delayed startup ratio*, which is the number of requests that are served with a startup latency normalized by the total number of requests. The last metric we use is the *byte hit ratio*, which is the amount of data delivered to the client from the proxy cache normalized by the total bytes the clients demand.

We evaluate these three metrics in three designs of a segment-based proxy caching system. The *Proxy-Hit* represents the adaptive-lazy segmentation based proxy caching system [1] with active prefetching. This scheme aims at improving the byte hit ratio. The *Proxy-Startup-Hit* represents the improved adaptive-lazy segmentation based proxy caching system with active prefetching. This scheme is designed to reduce the delayed startup ratio subjective to improving the byte hit ratio. The *Proxy-Jitter-Startup-Hit* represents our proposed *Hyper Proxy* system in this paper, aiming at minimizing proxy jitter subjective to minimizing the delayed startup ratio while maintaining a high byte hit ratio.

For the WEB workload, figure 2(a) shows that Hyper Proxy always provides the best continuous streaming service to the client while Proxy-Hit system which aims at increasing byte hit ratio, performs worst. Specifically, when cache size is 20% of total object size, Hyper Proxy reduces proxy jitter by more than 50%.

Figure 2(b) shows that Hyper Proxy achieves the lowest delayed startup ratio. Proxy-Startup-Hit achieves results close to Hyper Proxy. This is expected as we have analyzed in the [14].

Figure 2(c) shows Hyper Proxy achieves a relatively low byte hit ratio, which indicates a smaller reduction of network traffic. This is the price to pay for less proxy jitter and the smaller delayed startup ratio as shown in Figure 2(a) and (b).

Similar results are observed for the PART workload as shown in figure 3. When cache size is 20% of total object size, Hyper Proxy reduces proxy jitter by 50% by giving up less than 5 percentage points in the byte hit ratio. Figure 3(b)

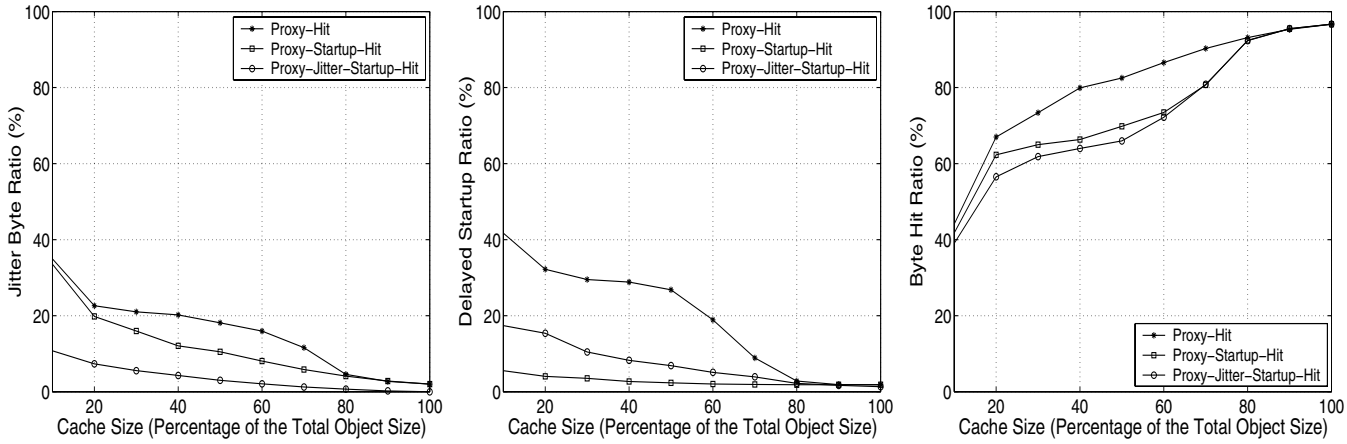

Fig. 3. PART: (a) Jitter Byte Ratio, (b) Delayed Startup Ratio, and (c) Byte Hit Ratio

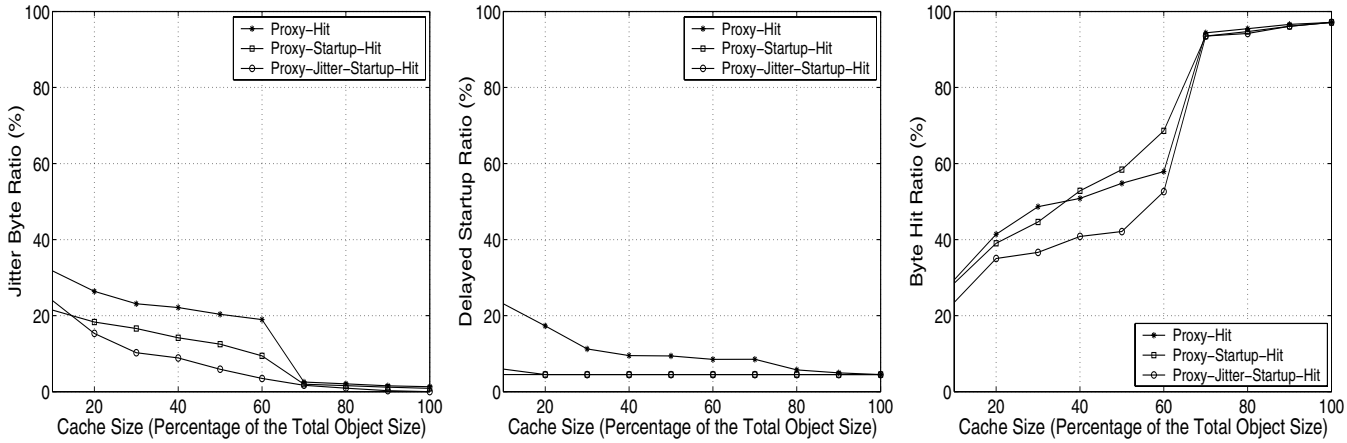

Fig. 4. REAL: (a) Jitter Byte Ratio, (b) Delayed Startup Ratio, and (c) Byte Hit Ratio

shows that Proxy-Startup-Hit achieves the best performance in reducing the delayed startup ratio. The result is expected since this scheme is specifically designed to prioritize reducing the delayed startup ratio. On the other hand, since Hyper Proxy proactively prevents proxy jitter by keeping more segments, more cache space is used for segments that may not be requested due to early termination. This in turn makes Hyper Proxy perform not as well in reducing the delayed startup ratio.

In a more realistic setup, we use the REAL workload to evaluate performance. As shown in Figure 4, Hyper Proxy performs best in reducing proxy jitter and delayed startup. The performance degradation in byte hit ratio is also acceptable.

VI. Related Work

Proxy caching of streaming media has also been studied in [4], [6], [9], [13], [23], [24]. In video staging [9], a portion of bits from the video frames whose size is larger than a predetermined threshold is cut off and prefetched to the proxy to reduce the bandwidth on the server proxy channel. So some frames are cut off into two parts: one is cached on the proxy and the other remains on the server. In [23], the algorithm attempts to select groups of consecutive frames by the selective caching algorithm, while in [5], the algorithm may select groups of non-consecutive frames for caching in the proxy. The caching problem for the layer-encoded video is studied in [4]. The cache replacement of streaming media is studied in the [6], [24].

VII. Conclusion

Proxy has been successfully used for caching text-based content. Using proxy to support media delivery is cost-effective, but challenging due to the nature of large media sizes and the low-latency and continuous streaming demand. Most existing studies target at improving the byte hit ratio that is commonly used in standard proxy caching. However, this is not the major concern for streaming media delivery, because it does not guarantee the continuous media delivery when the to-be-viewed object segments are not cached in the proxy, which causes proxy jitter. Our contributions in this paper are as follows:

- We have presented an optimization model to guide designs of highly effective media proxy caches and ensure a high delivery quality to the clients, which aims at

minimizing proxy jitter subject to reducing the startup latency and increasing the byte hit ratio.

- We have provided insights into the model by analyzing two pairs of conflicting interests and trade-offs inherent in this model.
- Guided by our optimization model, we have presented an active prefetching method to avoid proxy jitter with a low resource cost. Based on our analysis, we have further restructured the adaptive-lazy segmentation based proxy caching strategy. Since this strategy always achieves the highest byte hit ratio for delivering streaming media, we are able to minimize proxy jitter and the delayed startup ratio while still keeping a reasonably high byte hit ratio.
- Coordinating our prefetching and proxy caching schemes, we have proposed to build a new media proxy caching system called Hyper Proxy. This system addresses the interests from the perspectives of both clients and Internet resource management with a high priority given to the clients. We have shown that the Hyper Proxy system minimizes the amount of proxy jitter with a low delayed startup ratio and acceptable low network traffic compared with other existing caching schemes.

Currently we are deploying the Hyper Proxy system in a real testing environment, and making a strong effort to push this system as a model of next generation proxy caching for high quality and low cost streaming media delivery.

Acknowledgments: This work is supported by NSF grants CCR-0098055 and ACI-0129883, and a grant from Hewlett-Packard Laboratories. We thank the anonymous referees for their critical and constructive comments to our work. We appreciate Bill Bynum reading the paper and making suggestions.

REFERENCES

[1] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and lazy segmentation based proxy caching for streaming media delivery," in *Proceedings of ACM NOSSDAV*, Monterey, CA, June 2003.

[2] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the internet," in *Proceedings of IEEE Real Time Technology and Applications Symposium*, Washington, DC, May 2000.

[3] M. Y. Chiu and K. H. Yeung, "Partial video sequence caching scheme for vod systems with heteroeneous clients," in *Proceedings of the 13th International Conference on Data Engineering*, Birmingham, United Kingdom, April 1997.

[4] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," in *Proceedings of IEEE Infocom*, Anchorage, AK, April 2001.

[5] Z. Miao and A. Ortega, "Scalable proxy caching of video under storage constraints," in *IEEE Journal on Selected Areas in Communications*, September 2002.

[6] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," in *Proceedings of the First International Workshop on Intelligent Multimedia Computing and Networking*, Atlantic City, NJ, February 2000.

[7] R. Rejaie, M. Handely, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proceedings of ACM SIGCOMM*, Cambridge, MA, September 1999.

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

[9] Z. Zhang, Y. Wang, D. Du, and D. Su, "Video staging: A proxy-server based approach to end-to-end video delivery over wide-area networks," in *IEEE Transactions on Networking*, vol. 8, Aug. 2000, pp. 429–442.

[10] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy caching mechanism for multimedia playback streams in the internet," in *Proceedings of International Web Caching Workshop*, San Diego, CA, March 1999.

[11] R. Rejaie, M. H. H. Yu, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[12] K. Wu, P. S. Yu, and J. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of WWW*, Hongkong, China, May 2001.

[13] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching," in *IEEE Journal on Selected Areas in Communications*, September 2002.

[14] S. Chen, B. Shen, S. Wee, and X. Zhang, "Investigating performance insights of segment-based proxy caching of streaming media strategies," in *Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking*, San Jose, CA, January 2004.

[15] L. Cherkasova and M. Gupta, "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *Proceedings of ACM NOSSDAV*, Miami, FL, May 2002.

[16] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[17] B. Wang, S. Sen, M. Adler, and D. Towsley, "Proxy-based distribution of streaming video over unicast/multicast connections," in *Proceedings of IEEE INFOCOM*, New York City, NY, June 2002.

[18] J. I. Khan and Q. Tao, "Partial prefetch for faster surfing in composite hypermedia," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[19] J. Jung, D. Lee, and K. Chon, "Proactive web caching with cumulative prefetching for large multimedia data," in *Proceedings of WWW*, Amsterdam, Netherland, May 2000.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," http://www.ietf.org/rfc/rfc1889.txt, January 1996.

[21] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (rtsp)," http://www.ietf.org/rfc/rfc2326.txt, April 1998.

[22] S. Chen, B. Shen, S. Wee, and X. Zhang, "Streaming flow analyses for prefetching in segment-based proxy caching strategies to improve media delivery quality," in *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, Hawthorne, NY, September 2003.

[23] W. Ma and H. Du, "Reducing bandwidth requirement for dilivering video over wide area networks with proxy server," in *Proceedings of International Conferences on Multimeida and Expo.*, New York City, NY, July 2000.

[24] R. Tewari, A. D. H. Vin, and D. Sitaram, "Resource-based caching for web servers," in *Proceedings ACM/SPIE Conference on Multimeida Computing and Networking*, San Jose, CA, January 1998.