Streaming Flow Analyses for Prefetching in Segment-based Proxy Caching to Improve Media Delivery Quality *

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

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

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

Abstract

Segment-based proxy caching schemes have been effectively used to deliver streaming media objects. However, this approach does not always guarantee the continuous delivery because the to-be-viewed segments may not be cached in the proxy in time. The potential consequence is the playback *jitter* at the client side due to the proxy delay in fetching these uncached segments, thus we call the problem as *proxy jitter*. Aiming at improving the media delivery quality for segment-based caching schemes, in this paper, we propose two simple and effective prefetching methods, namely, *look-ahead window based prefetching* and *active prefetching* to address the problem of proxy jitter. We focus on presenting streaming flow analyses on proxy and network resource utilizations and consumptions, performance potentials and limits of the two prefetching methods for different segment-based schemes under different network bandwidth conditions. Our study also provides some new insights into relationships between proxy caching performance and the quality of streaming. For example, we show that the objective of improving the byte hit ratio in a conventional proxy and the unique objective of minimizing the proxy jitter to deliver streaming media objects can have conflicting interests. Conducting trace-driven simulations, we show the effectiveness of our prefetching methods, and further confirm our analyses.

1 Background and Motivation

Proxy caching technique has been widely and effectively used to cache the text-based objects in Internet so that subsequent requests to the same object can be repeatedly and directly served from the proxy that is close to the requesting client. Typical proxy systems include CERN httpd [1], Harvest [2], and Squid [3]. However, with the proliferation of the multimedia objects in Internet, the existing proxy infrastructure is challenged due to their typical large sizes and continuous streaming requirement. A segment-based approach developed for proxy caching is to cache media objects in segments instead of their entirety to reduce the user perceived startup latency and to reduce the network traffic to media servers and the disk bandwidth requirement on the media server. An example of such an approach is the prefix caching [4] that segments the media object with a prefix segment and a suffix segment and always caches the prefix segments. These segment-based caching schemes can be classified into two types in terms of segment sizes. The first one is to use uniformly sized segments. For example, authors in [5] consider the caching of fixed sized segments of layer-encoded video objects. More recently, the adaptive-lazy segmentation strategies

have been proposed in [6], in which each object has itself segmented as late as possible by using an uniform segment length. We generally call them as the uniform segmentation strategy[1]. The second type is to use exponentially sized segments, where media objects are segmented in a way that the size of a segment increases exponentially from that of its preceding one [7]. We generally name them as the exponential segmentation strategy.

However, these segment-based proxy caching schemes do not always guarantee a continuous delivery because the to-be-viewed segments may not be loaded in the proxy in time when they are accessed. We call this problem *proxy jitter*. (On the client side, this problem is called "playback jitter"). A straightforward solution to eliminate the playback jitter is to store the full object by a viewing client. This special solution is also called "downloading" that requires a very large buffer size and introduces a prolonged startup delay. Unfortunately, this is not a cost-effective method. Many clients have limited buffer space and they rely on media proxies and/or servers to ensure the quality of service. Once a playback starts in a client, pausing in the middle caused by the proxy jitter is not only annoying but can also potentially drive the client away from accessing the content. The essential issue for a proxy using a segment-based caching scheme is to fetch and relay the demanded segments to the client in time.

The key to solve the proxy jitter problem is to develop prefetching schemes to preload the uncached segments in time. Some early work have studied the prefetching of multimedia objects [8, 9, 5, 10]. For layered-encoded objects [5, 10], 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 a fixed time period before its playback time. In [8], the prefetching is used to preload a certain amount of data so as to take advantage the caching power. In [9], a proactive

prefetching method is proposed to utilize any partially fetched data due to the connection abortion to improve the network bandwidth utilization. In [11], authors consider the playback restart in interactive streaming video applications by using an optimal smoothing technique to transmit data for regular playback.

To our best knowledge, prefetching methods have not been seriously studied in the context of segment-based proxy caching [12]. Specifically, neither have the previous prefetching methods considered the following unique conflicting interests in delivering streaming media objects. On one hand, *proxy jitter* is caused by the delayed prefetching of uncached segments, which clearly suggests that the proxy prefetch the uncached segments as early as possible. On the other hand, the effort of aggressively prefetching the uncached segments requires a significant increase of the buffer space for temporarily storing the prefetched data and the bandwidth to transfer the data. Even worse, the client session may terminate before such a large amount of prefetched segments are accessed. This fact suggests that the proxy prefetch the uncached segments as late as possible. Thus, an effective media streaming proxy should be able to decide when to prefetch which uncached segment aiming at minimizing the *proxy jitter* subject to minimizing the related overhead (wasted space and bandwidth).

In order to improve the media delivery quality for segment-based caching schemes, in this paper, we propose two simple and effective prefetching methods, namely, *look-ahead window based prefetching* and *active prefetching* to address the problem of proxy jitter. We focus on presenting streaming flow analyses on proxy and network resource utilizations and consumptions, performance potentials and limits of the two prefetching methods for different segment-based schemes under different network bandwidth conditions. Our study also provides some new insights into relationships between proxy caching performance and the quality of streaming. For example, we show that the objective of improving the byte hit ratio in a conventional proxy and the unique objective of minimizing the proxy jitter to deliver stream-

---

[1]The term "uniform" in this strategy only applies to each object. However, different objects may have different segment lengths.

ing media objects can have conflicting interests. Conducting trace-driven simulations, we show the effectiveness of our prefetching methods, and further confirm our analyses.

The rest of this paper is organized as follows. The look-ahead window based prefetching method and the active prefetching method are presented in Section 2 along with the related streaming flow analyses. We further discuss insights of media proxy steaming in Section 3. We evaluate the proposed methods in Section 4. We make concluding remarks in Section 5.

2 Prefetching Methods for Segment-based Proxy Caching

In this section, we present two prefetching methods, namely look-ahead window based prefetching and *active prefetching*, along with their streaming flow analyses. Both prefetching methods consider achieving the two objectives jointly: (1) to fully utilize the storage resource, and (2) to minimize *proxy jitter*.

We have made the following assumptions in our analyses.

- A media 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 a client smoothly; and

- Each segment of the object can be delivered by a media server in a unicast channel.

Considering that the prefetching is conducted segment by segment, we define the following related variables.

- B_s: the average streaming rate of object segments in bytes/second (can be extracted from the object's inherent encoding rate), which can be viewed as the average network bandwidth consumption of the proxy-client link;

- B_t: the average network bandwidth of the proxy-server link;

- k: the total number of segments of an object;

- L_i: the length of the ith segment of an object, and $L_1 = L_{base}$ is the size of the first segment, also known as the base segment size. For the uniform segmentation strategy, $L_i = L_1$; for the exponential segmentation strategy, $L_i = 2^{i-1}L_1$, where $k \geq i > 1$; (Note: in [7], $L_i = 2^{i-2}L_1$ ($i > 2$) since the segment is numbered from 0, while we number it from 1 for the consistency with the uniform segmentation strategy.)

- W: the amount of the prefetched data in bytes that are not used by the client due to an early termination (a client may only want to view a portion of the media object). The prefetching requires resources of (1) the buffer to temporarily store them; and (2) the network bandwidth used to transfer them. Both the buffer size and the network transferring cost are proportional to W. Denoting the networking cost as $f_n(W)$, the buffer cost as $f_b(W)$ and the wasted resource as as $f_w(W)$, we have $f_w(W) = f_n(W) + f_b(W)$. As long as W is known, the wasted resource cost can be determined.

2.1 Look-ahead Window Based Prefetching Method

The major action of the look-ahead window based prefetching is to prefetch the succeeding segment if it is not cached when the client starts to access the current one. The window size is thus fixed for the uniform segmentation strategy and is exponentially increasing for the exponential segmentation strategy.

Considering the streaming rate B_s and the network bandwidth B_t, we propose the look-ahead window based prefetching method under the following three different conditions.

1. $B_s = B_t$.

   Ideally, in this case, the streaming flow can be smooth from the server to proxy and then to a client without an effort of prefetching, thus, no buffer is needed. However, in

practice, we do need a buffer denoted as $Buffer_{network}$ to smooth the network jitters caused by network traffic and congestion. The size of $Buffer_{network}$ can be experimentally determined in practice. For example, it can be large enough to buffer media segments of 5 seconds. In this case, the preloading of the next uncached segment always starts 5 seconds before the streaming of the current segment is complete.

2. $B_s < B_t$.

   In this case, no prefetching is needed, but we need a buffer to hold overflowed streaming data in the proxy from a media content server, which is denoted as $Buffer_{overflow}$. The preloading of the next uncached segment does not start until the streaming of the current segment is complete. The size of $Buffer_{overflow}$ is thus at least

   $$L_i - \frac{L_i}{B_t} \times B_s = L_i \times (1 - \frac{B_s}{B_t}). \qquad (2.1)$$

   For different segmentation strategies, the buffer size can be calculated as follows:

   - For the uniform segmentation strategy, the size of $Buffer_{overflow}$ is at least $(1 - \frac{B_s}{B_t})L_1$.

   - For the exponential segmentation strategy, the size of $Buffer_{overflow}$ is at least $(1 - \frac{B_s}{B_t})L_i$, which increases exponentially.

3. $B_s > B_t$.

   In this case, both segment prefetching and buffering are needed. A buffer denoted as $Buffer_{starvation}$ is used in the proxy to prevent the streaming starvation of a client. Assume the prefetching of the next uncached segment S_{i+1} starts when the client starts to access the position x in the current segment S_i. Thus, x is the position that determines the starting time of prefetching, called the prefetching starting point. To denote y as $y = L_i - x$ and to guarantee the in-time

prefetching of the next uncached segment, we have

$$\frac{y + L_{i+1}}{B_s} \geq \frac{L_{i+1}}{B_t}, \qquad (2.2)$$

which means

$$y \geq \frac{L_{i+1} \times (S - T)}{T}. \qquad (2.3)$$

Since $y = L_i - x$, thus

$$x \leq L_i - \frac{L_{i+1} \times (B_s - B_t)}{B_t}. \qquad (2.4)$$

We can calculate the prefetching starting point as the percentage of the current segment by dividing x by L_i, which leads to

$$\frac{x}{L_i} \leq 1 - \frac{L_{i+1}}{L_i} \times (\frac{B_s}{B_t} - 1). \qquad (2.5)$$

Equation (2.5) means to prefetch the next uncached segment when the client has accessed the $1 - \frac{L_{i+1}}{L_i} \times (\frac{B_s}{B_t} - 1)$ portion of current segment. Accordingly, the size of the minimum buffer size for $Buffer_{starvation}$ is $\frac{y}{B_s} \times B_t$, which is $L_{i+1} \times (1 - \frac{B_t}{B_s})$. Once we know the minimum buffer size, we know that in the worst case, the fully buffered prefetched data may not be used by the client, which means the maximum amount of wasted prefetched data, W, that has the same size as the buffer. Thus, we always give the minimum buffer size by the following analysis.

For different segmentation strategies, the situations are as follows:

- For the uniform segmentation strategy, by Equation (2.4), we have $\frac{x}{L_i} \leq 2 - \frac{B_s}{B_t}$. It implies that B_s could not be 2 times larger than B_t. The minimum size of $Buffer_{starvation}$ is $L_1 \times (1 - \frac{B_t}{B_s})$. The prefetching of the next uncached segment starts when the client has accessed to the $2 - \frac{B_s}{B_t}$ portion of the current segment.

- For the exponential segmentation strategy, by Equation (2.4), we have $\frac{x}{L_i} \leq 3 - 2 \times \frac{B_s}{B_t}$. It implies that B_s could not be 1.5 times larger than B_t. The minimum size of $Buffer_{starvation}$ is $L_{i+1} \times (1 - \frac{B_t}{B_s})$, which increases exponentially. The prefetching of the next uncached segment starts when the client has accessed the $3 - 2 \times \frac{B_s}{B_t}$ portion of the current segment.

Above analysis shows that this look-ahead window based prefetching method does not work when $B_s > 1.5B_t$ for the exponential segmentation strategy, and it does not work when when $B_s > 2B_t$ for the uniform segmentation strategy.

In addition, since $B_s > B_t$, we have

$$B_s > B_t \Rightarrow \quad \frac{B_s}{B_t} \quad > 1$$
$$\Rightarrow \quad 2 \times \frac{B_s}{B_t} - \frac{B_s}{B_t} \quad > 1$$
$$\Rightarrow \quad -\frac{B_s}{B_t} \quad > 1 - 2 \times \frac{B_s}{B_t}$$
$$\Rightarrow \quad 2 - \frac{B_s}{B_t} \quad > 3 - 2 \times \frac{B_s}{B_t}. \quad (2.6)$$

The left side of Equation (2.6) represents the prefetching starting point for the uniform segmentation strategy, while the right side denotes that for the exponential segmentation strategy. Thus, Equation (2.6) states that the prefetching of the next uncached segment for the exponential segmentation strategy is always earlier than that for the uniform segmentation strategy, causing a higher possibility of wasted resources.

Since the condition of $B_s > B_t$ is quite common in practice, the look-ahead window based prefetching method has a limited prefetching capability in reducing the proxy jitter. Next, we will address its limit by an active prefetching method.

2.2 Active Prefetching Method

If the prefetching is conducted more aggressively, we are able to further reduce proxy jitter, and of course, which will also consume more resources. The basic idea of our second method, active prefetching, is to preload uncached segments as early as the time when the client starts to access a

media object. We define the following additional notations for this prefetching method.

- n: the number of cached segments of a media object;

- m: when the in-time prefetching of $n + 1^{th}$ segment is not possible (which will be discussed soon), the proxy should start to prefetch a later segment m once the client starts to access an object, where $m > n + 1$.

We also re-define the prefetching starting point, x, as the position in the first n cached segments (instead of a position in the nth segment for the look-ahead window based prefetching method) that is accessed by a client. As soon as this prefetching starting point is accessed, the prefetching of $n + 1^{th}$ segment must start in order to avoid the proxy jitter.

Based on Equations (2.2) to (2.5), we obtain the prefetching starting point x as

$$x \leq \sum_{i=1}^{i=n} L_i - \frac{L_{n+1} \times (B_s - B_t)}{B_t}. \quad (2.7)$$

Next, we will discuss the active prefetching alternatives for different segmentation strategies.

2.2.1 Uniform Segmentation Strategy

As we know, when $B_s > 2B_t$, it is impossible for the uniform segmentation strategy using look-ahead window based prefetching to completely avoid proxy jitter. However, the active prefetching may work this situation. Based on Equation (2.7), we obtain

$$x \leq nL_1 - \frac{\frac{B_s}{B_t} - 1}{n} \quad (2.8)$$

for the uniform segmentation strategy.

Equation (2.8) not only gives the prefetching starting point when $n + 1 \geq \frac{B_s}{B_t}$, it also implies that if $n + 1 < \frac{B_s}{B_t}$, the in-time prefetching of $n + 1^{th}$ segment is not possible! So when $n + 1 < \frac{B_s}{B_t}$ and the segments between $n + 1^{th}$ and $\frac{B_s}{B_t}^{th}$ are demanded, the *proxy jitter* is inevitable. Thus, when $n + 1 < \frac{B_s}{B_t}$, to minimize the *proxy*

jitter, the proxy should start the prefetching of a later segment, denoted as m, rather than the $n+1^{th}$ segment since it can not be prefetched in time. The prefetching of the mth segment should be faster than the streaming of the first m segments, which leads to

$$\frac{L_m}{B_t} \leq \frac{mL_1}{B_s}. \qquad (2.9)$$

We get $m \geq \frac{B_s}{B_t}$, and the corresponding minimum buffer size is thus

$$\frac{(m-1)L_i}{B_s} \times B_t = (1 - \frac{B_t}{B_s})L_1. \qquad (2.10)$$

For the uniform segmentation strategy, the active prefetching works as follows when $B_s > 2B_t$. (When $B_s \leq 2B_t$, it works the same as window-based prefetching method.)

- $n = 0$: No segment is cached. The proxy starts to prefetch the $\frac{B_s}{B_t}^{th}$ segment. Before the client accesses to this segment, the *proxy jitter* is inevitable. The minimum buffer size 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 $\frac{B_s}{B_t}^{th}$ segment once the client starts to access the object. If the segments between $n+1^{th}$ and $\frac{B_s}{B_t} - 1^{th}$ are demanded, the *proxy jitter* is inevitable. The minimum buffer size is $(1 - \frac{B_t}{B_s})L_1$.

- $n > 0$ 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 $(n + 1 - \frac{B_s}{B_t})L_1$ of the first n cached segments. The minimum buffer size is $(\frac{B_s}{B_t} - 1) \times \frac{L_1}{B_s} \times B_t$, i.e., $(1 - \frac{B_t}{B_s})L_1$. The *proxy jitter* can be totally avoided.

2.2.2 Exponential Segmentation Strategy

For the exponential segmentation strategy, when $B_s > 1.5B_t$, the calculation of the prefetching starting point based on Equation (2.7) leads to

$$x \leq \sum_{i=1}^{i=n} L_i \times (1 - \frac{2^n}{2^n - 1} \times (\frac{B_s}{B_t} - 1)). \qquad (2.11)$$

The right side of Equation (2.11) must be greater than 0, so Equation (2.11) not only determines the prefetching starting point, it also means that when $n < log_2(\frac{1}{2 - \frac{B_s}{B_t}})$, the in-time prefetching of $n + 1^{th}$ segment is not possible, and when $B_s \geq 2B_t$, the in-time prefetching of any uncached segment can never be possible!

When $n < log_2(\frac{1}{2 - \frac{B_s}{B_t}})$, to minimize the *proxy jitter*, the proxy should start the prefetching of some later segment, denoted as m since the $(n+1)^{th}$ segment can not be prefetched in time. Thus, the transferring of the mth segment must be faster than the streaming of the first m segments, which leads to

$$\frac{L_m}{B_t} \leq \frac{\sum_{i=1}^{i=m} L_i}{B_s}. \qquad (2.12)$$

Thus, we get $m \geq 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}})$ and the corresponding minimum buffer size is

$$2 \times \frac{L_{m-1}}{B_s} \times B_t = L_1 \times \frac{B_t^2}{2B_sB_t - B_s^2}. \qquad (2.13)$$

For the exponential segmentation strategy, the active prefetching works as follows when $B_s > 1.5B_t$. (When $B_s \leq 1.5B_t$, it works the same as window-based prefetching method.)

1. $B_s > 1.5B_t$ and $B_s < 2B_t$.

   - $n = 0$: No segment is cached. The proxy starts to prefetch the $1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}})^{th}$ segment once the client starts to access the object. The *proxy jitter* is inevitable before the client accesses to this segment. The minimum buffer size $L_1 \times \frac{B_t^2}{2 \times B_s \times B_t - B_s^2}$.

   - $n > 0$ and $n \leq log_2(\frac{1}{2 - \frac{B_s}{B_t}})$: The proxy starts to prefetch the $1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}})^{th}$ segment once the client starts to access this object. The minimum buffer size is $L_i \times \frac{B_t}{B_s}$, where $i = 1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}})$. The *proxy jitter* is inevitable when the client accesses segments between the $n + 1^{th}$ segment and $1 + log_2(\frac{1}{2 - \frac{B_s}{B_t}})^{th}$ segment.

- $n > 0$ 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 $L_{n+1} \times \frac{B_t}{B_s}$, which will increase exponentially.

2. $B_s \geq 2B_t$.

No prefetching of the uncached segments can be in time. Whenever a client accesses to an uncached segment, the *proxy jitter* can not be reduced.

3 Segment-based Proxy Caching Strategies with Least Proxy Jitter

We have shown that even the active prefetching can not always guarantee the continuous media delivery. However, the analysis of the active prefetching also indicates that for any strategy, with enough segments being always cached in the proxy, the prefetching of the uncached segments can always be in time whenever necessary. The number of the segments being always cached in the proxy for this purpose, which we denote as q, is dependent on the segmentation strategy, B_s and B_t.

In this section, we determine q for different segmentation strategies to guarantee the continuous delivery without considering a cache space limit when $B_s > B_t$. Then we further look into its insights with a limited cache size.

3.1 Minimum Number of Segments Cached for Proxy Jitter Free

We use the notations as before, the prefetching of the $q + 1^{th}$ segment to the kth segment must be faster than the streaming of the whole object, which means

$$\frac{\sum_{i=1}^{i=k} L_i}{B_s} \geq \frac{\sum_{i=1}^{i=k} L_i - \sum_{i=1}^{i=q} L_i}{B_t}, \qquad (3.14)$$

where $\sum_{i=1}^{i=q} L_i$ is the amount of data should be cached. Thus,

$$\sum_{i=1}^{i=q} L_i \geq (1 - \frac{B_t}{B_s}) \sum_{i=1}^{i=k} L_i. \qquad (3.15)$$

Applying Equation (3.15) to different segmentation strategies, q can be determined as follows.

- Uniform segmentation strategy

  Since the L_{base} is the base segment length, the number of cached segment must be

  $$q = \lceil \frac{(1 - \frac{B_t}{B_s}) \sum_{i=1}^{i=k} L_i}{L_{base}} \rceil. \qquad (3.16)$$

- Exponential segmentation strategy

  Since $L_i = 2 \times L_{i-1}$, the number of cached segments must be

  $$q = \lceil log_2(\frac{(1 - \frac{B_t}{B_s}) \sum_{i=1}^{i=k} L_i}{L_{base}}) \rceil + 1. \qquad (3.17)$$

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

In a practical segment-based streaming environment with a limited cache size, it is common that for some objects, they have a large number of segments cached (e.g. very popular objects), which is very likely to be larger than their q values, while for some objects (e.g. not popular objects), they may have a lower number of segments cached in the system, which is possibly to be less than their q values. If we evict some rear segments of these popular objects whose cached number of segments is larger than their q values, the byte hit ratio of the system decreases. However, the released space can be used to cache the objects whose cached number segments is less than their q values, which reduces the proxy jitter. This scenario implies that there is a trade-off between the high byte hit ratio and the low proxy jitter with a limited cache size.

With the trade-off consideration, we further change the existing uniform and exponential segmentation strategies as follows with two additional notations:

- K_{min}: the minimum number of segments that if cached, no startup latency will be perceived by the clients as in [7];

- K_{jitter}: the minimum number of segments that if cached, no proxy jitter or startup latency will be caused. It is the maximum of K_{min} and q.

In the modified segmentation strategies, instead of caching the first K_{min} segments for reducing the startup latency when the object is initially accessed, the larger value of K_{min} and the minimum number of segments for guaranteed in-time prefetching, q, is calculated, as K_{jitter}, and the first K_{jitter} segments of the object is cached. However, when the cache space is demanded, these K_{jitter} segments other than the first K_{min} need to compete for the cache space according to their caching utility value. Thus, the decrease of the byte hit ratio can be used to trade for the reduction of the proxy jitter to some extent.

4 Performance Evaluation

To evaluate the performance of these prefetching methods under different segmentation strategies, an event-driven simulator has been built. We simulate the exponential segmentation strategy as in [7]. The uniform segmentation strategy differs from the exponential segmentation strategy in that all objects are segmented in a uniform length of 1KB according to [6].

To evaluate the effectiveness of the prefetching methods, the major metric we use is *proxy jitter bytes*. It is the amount of data that are not served to the client in time by the proxy, thus causing the potential playback jitter at the client side, normalized by the total bytes the clients demand. It denotes the quality of the streaming service to the client. The *in-time prefetched* byte is used to represent how many bytes could be prefetched in time, normalized by the total bytes the clients demand. The *byte hit ratio* is defined as the amount of data delivered to the client from the proxy and normalized by the total bytes the clients demand. It is used to show the impact on caching performance when different prefetching methods are used.

Based on the previous analysis, we know that once the bandwidth of proxy-server link (or connection rate will be used in the following context) is less than half of the streaming rate, it may be impossible for the exponential segmentation strategy to prefetch any uncached segments in time. Thus, in this evaluation, the streaming rate of each object is set in the range of half to two times of the bandwidth of proxy-serve link. (Note that other than the results presented in the following context, we have also evaluated the performance with larger ranges of streaming rate and proxy-server bandwidth. The results show the similar trends with larger gaps between the uniform and exponential segmentation strategies, which we omit due to the page limit.)

4.1 Workload Summary

To evaluate the performance, we conduct simulations based on two synthetic workloads. The first, named WEB, simulates accesses to media object in the Web environment in which the length of the video varies from short ones to longer ones. Since clients' accesses to videos may be incomplete, that is, a session started may terminate before the full media object is delivered, we use the second workload, named PART, to simulate this behavior. In this workload, 80% of the session terminates before 20% of the object is delivered. Both workloads assume a Zipf-like distribution ($p_i = f_i/\sum_{i=1}^{N} f_i, f_i = 1/i^\alpha$) for the popularity of the media objects, where α is set to 0.47 for both workloads. They also assume request inter arrival to follow the Poisson distribution ($p(x, \lambda) = e^{-\lambda} \times (\lambda)^x/(x!), x = 0, 1, 2...$) with a λ of 4.

Both WEB and PART have a total number of 15188 requests in a day for 400 different media objects. The total size of all media objects is 51 GB. The viewing length of objects ranges from 2 to 120 minutes.

4.2 Exponential Segmentation Strategy

In all the following figures, notation *Window* represents the look-ahead window based prefetching
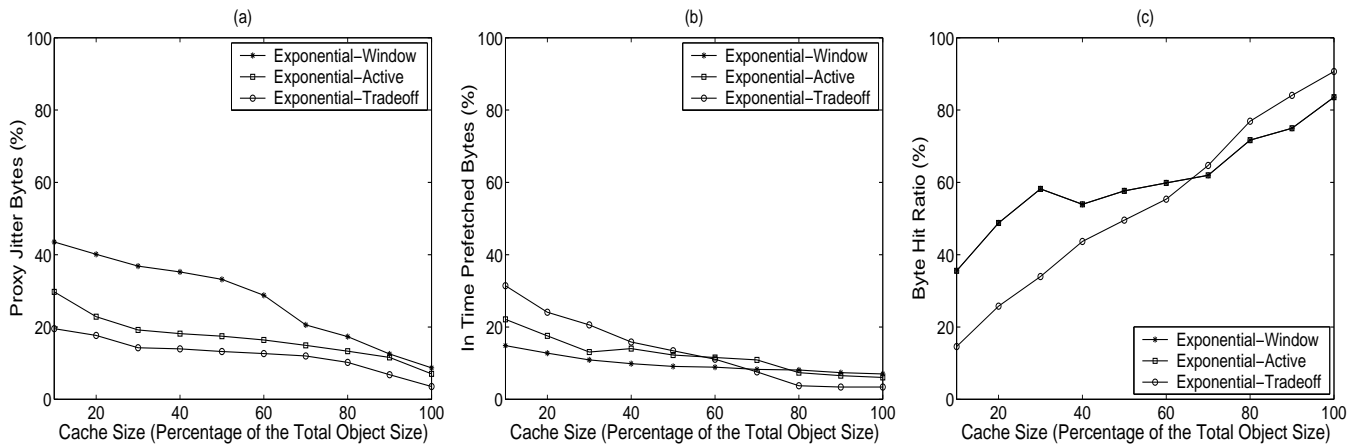
Figure 1: WEB: (a) Proxy Jitter Bytes, (b) Bytes Prefetched In Time & (c) Byte Hit Ratio
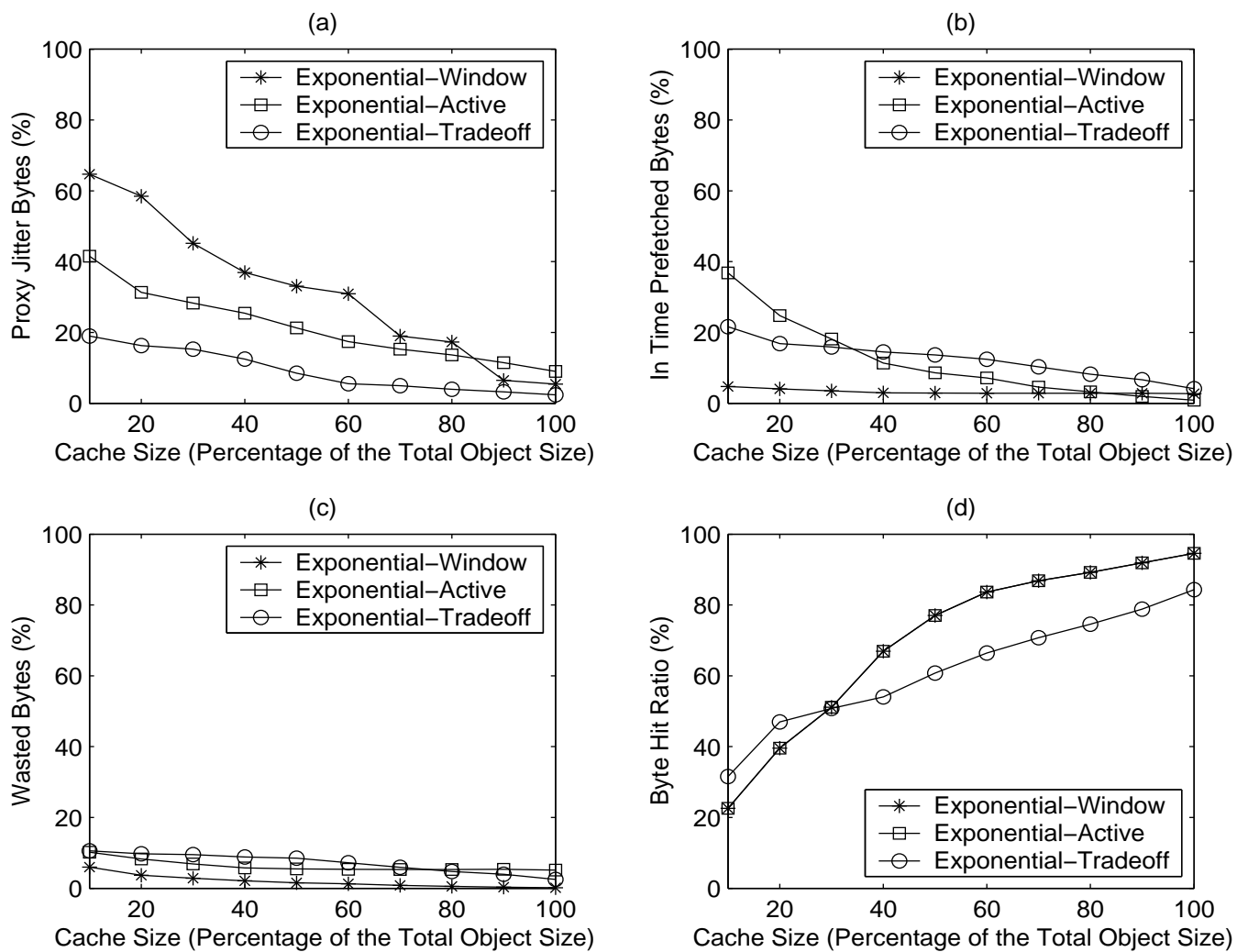

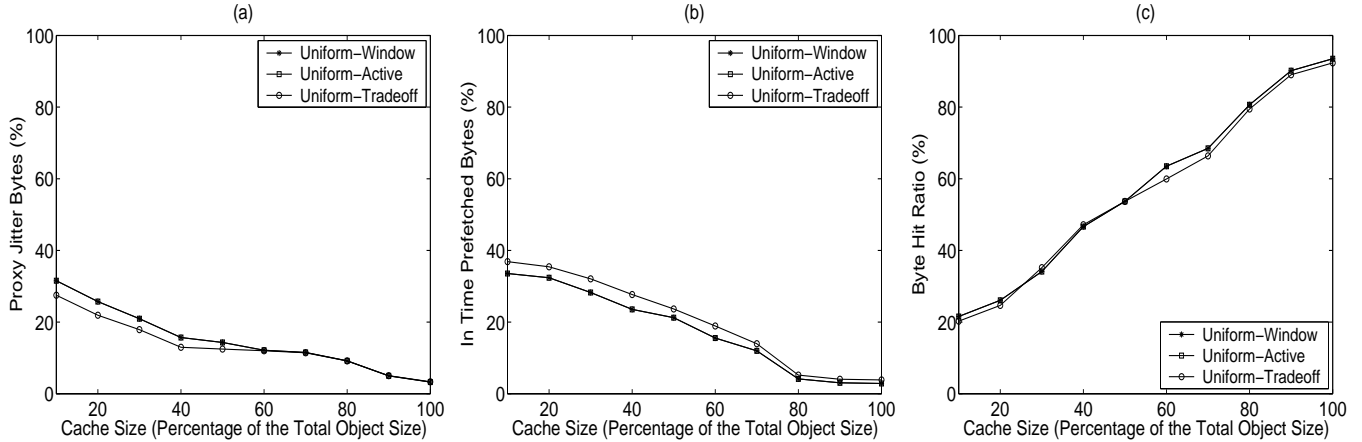Figure 2: PART: (a) Proxy Jitter Bytes, (b) Bytes Prefetched In Time, (c) Bytes Wasted & (d) Byte Hit Ratio

Figure 3: WEB: (a) Proxy Jitter Bytes, (b) Bytes Prefetched In Time & (c) Byte Hit Ratio

method, and notation *Active* represents the active prefetching method. The *Tradeoff* represents the prefetching method with the consideration of the trade-off between the byte hit ratio and the proxy jitter.

Figure 1 shows the performance results by using workload WEB for the exponential segmentation strategy. Figure 1(a) shows that *Exponential-Tradeoff* has the least proxy jitter bytes served to the clients and much less than that of *Exponential-Window* and *Exponential-Active*. As shown in Figure 1(c), although the byte hit ratio achieved by *Exponential-Tradeoff* decreases when the cache size is less than 60% of the total object size, the overall streaming service quality provided to the client is improved.

Figure 1(b) shows that *Exponential-Tradeoff* achieves the largest amount of data prefetched in time on average and decreases when the cache size increases. This is because more data has been cached in the proxy with more available cache space.

Figure 1(c) shows that the byte hit ratios for *Exponential-Window* and *Exponential-Active* are the same. This is expected as our previous analysis indicates that if the connection rate is larger than half of the streaming rate, only the amount of prefetched data for those two prefetching methods are different.

Figure 2 shows the results by using the workload PART for the exponential segmentation strategy. Figure 2(a) shows that *Exponential-Tradeoff* still achieves the least percentage of proxy jitter bytes served to the clients. Figure 2(b) shows that on average *Exponential-Tradeoff* achieves the largest amount of data prefetched in time.

For the partial viewing cases, a new metric, the *wasted* byte, is used to evaluate how much of the in-time prefetched data that is not actually used since the client terminates earlier. Figure 2(c) shows that all three prefetching methods produce some wasted bytes due to a large percentage of prematurely terminated sessions. *Exponential-Tradeoff* results in more wasted bytes than others since it gets more in-time prefetched data.

4.3 Uniform Segmentation Strategy

As aforementioned, in the experiments, the streaming rate of each object is set in the range of half to two times of the connection rate, thus the performance improvement of the uniform segmentation strategy does not look as significant as the exponential segmentation strategy. Also, for the uniform segmentation strategy, if the connection rate is larger than half of the streaming rate, the active prefetching and the look-ahead window based prefetching methods perform exactly same.

Figure 3 shows the performance results by using workload WEB. Figure 3(a) shows that *Uniform-Tradeoff* has the least *proxy jitter* bytes on average. Although Figure 3(c) shows it
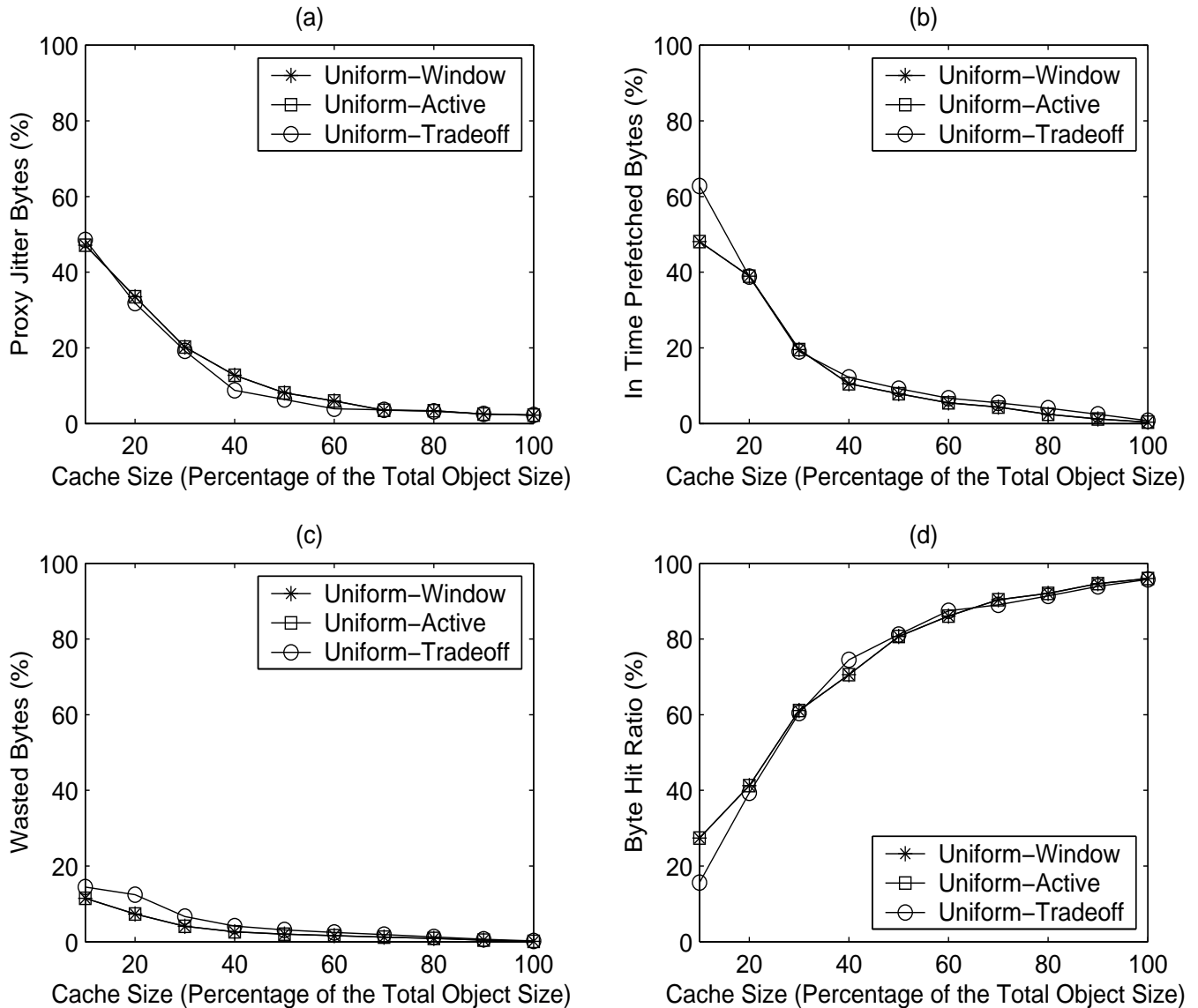
Figure 4: PART:(a) Proxy Jitter Bytes, (b) Bytes Prefetched In Time, (c) Bytes Wasted & (d) Byte Hit Ratio

achieves a lower byte hit ratio, the quality of the streaming service it provides to the client is the best. Figure 3(c) shows that the byte hit ratios achieved by *Uniform-Window* and *Uniform-Active* are the same. This is consistent to our previous analysis. On Figure 3(a) and Figure 3(b), they also achieve the same performance results. Figure 3(b) shows that *Uniform-Tradeoff* gets the largest amount of in-time prefetched data.

Figure 4 shows the results by using the partial viewing workload PART. Similar trends

have been observed as in Figure 3, but with a smaller performance gap among different methods. Figure 4(a) shows that *Uniform-Tradeoff* still achieves the least amount of proxy jitter bytes, giving the best quality of streaming delivery to the clients. Figure 4(a) shows that *Uniform-Tradeoff* also gets the largest amount of data prefetched in time, while Figure 4(c) shows that *Uniform-Tradeoff* has more wasted data due to more in-time prefetched data.

5 Conclusion

In this study, focusing on the segment-based caching strategies with a uniform segment length or an exponentially increasing segment length, we have examined the prefetching issues in the segment-based caching strategies. We have presented and evaluated the look-ahead window based prefetching method, the active prefetching method and the prefetching method with the consideration of the trade-off between improving the byte hit ratio and minimizing the proxy jitter analytically and experimentally. The prefetching method with the trade-off consideration is the most effective in reducing the proxy jitter and can provide the best quality of streaming delivery to the clients.

We are implementing the prefetching in our media surrogate testbed and performing real tests.

References

[1] A. Luotonen, H. Frystyk Nielsen, and T. Berners-Lee, "Cern httpd," http://www.w3.org/Daemon/Status.html.

[2] C.M. Bowman, P.B Danzig, D.R. Hardy, U. Manber, M.F. Schwartz, and D.P. Wessels, "Harvest: A scalable, customizable discovery and access system," in *Tech. Re. CU-CS-732-94*, University of Colorado, Boulder, USA, 1994.

[3] "http://www.squid-cache.org/," .

[4] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE INFOCOM*, 1999.

[5] 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*, Mar. 1999.

[6] 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 1-3 2003.

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

[8] 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*, 2001.

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

[10] R. Rejaie, M. Handely 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.

[11] J. K. Dey, S. Sen, J. F. Kurose, D. Towsley, and J. D. Salehi, "Playback restart in interactive streaming video applications," in *IEEE Conference on Multimedia Computing and Systems*, Ottawa, Canada, June 1997.

[12] S. Chen, B. Shen, S. Wee, and X. Zhang, "Analysis and design of segment-based proxy caching strategies for streaming media objects," in *Proceeding of ACM/SPIE Multimedia Computing and Networking*, Santa Clara, CA, Jan. 2004 (to appear).