Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems

Kien A. Hua   Simon Sheu
Department of Computer Science, University of Central Florida
Orlando, FL 32816-2362, U. S. A.
E-mail: {kienhua,sheu}@cs.ucf.edu

Abstract

We investigate a novel multicast technique, called Skyscraper Broadcasting *(SB), for video-on-demand applications. We discuss the data fragmentation technique, the broadcasting strategy, and the client design. We also show the correctness of our technique, and derive mathematical equations to analyze its storage requirement. To assess its performance, we compare it to the latest designs known as* Pyramid Broadcasting *(PB) and* Permutation-Based Pyramid Broadcasting *(PPB). Our study indicates that PB offers excellent access latency. However, it requires very large storage space and disk bandwidth at the receiving end. PPB is able to address these problems. However, this is accomplished at the expense of a larger access latency and more complex synchronization. With SB, we are able to achieve the low latency of PB while using only 20% of the buffer space required by PPB.*

1 Introduction

Television is an integral part of the majority of the households throughout the world. We find ourselves turning to it daily to be entertained, educated, or simply kept informed of current events. Our desire to watch has fueled an industry eager to deliver a variety of *video-on-demand* services. With these services, a subscriber is able to start the playback of a video of his choice at a press of a button.

Typically, a large number of video files are stored in a set of centralized video servers and played through high-speed communication networks by geographically distributed clients. Due to stringent response time requirements, continuous delivery of a video stream has to be guaranteed by reserving an I/O stream and an isochronous channel needed for the delivery. In this paper, we will refer to the unit of server capacity needed to support the continuous playback of one server stream as a *channel*. To maximize the utilization of these channels, efficient scheduling techniques have been proposed by Vin and Rangan [20], Özden *et al.* [15, 16], Freedman and DeWitt [9], Keeton and Katz [12], Oyang *et al.* [14], Rotem and Zhao [18], Dan *et al.* [6], and Hua *et al.* [10], just to name a few. These techniques are sometimes referred to as *user centered* [2, 22] in the sense that the channels are allocated among the users. Although this approach simplifies the implementation, dedicating a stream for each viewer will quickly exhaust the network-I/O bandwidth at the server communication ports. In fact, the network-I/O bottleneck has been observed in many systems, such as Time Warner Cable's *Full Service Network* project in Orlando, Microsoft's *Tiger Video Fileserver* [4], and so on.

To address the network-I/O bottleneck faced by the user-centered approach, the multicast facility of modern communication networks [5, 13, 11, 17] can be used to allow users to share a server stream. For example, if two subscribers make a request for the same video separated by a small time interval, then by delaying the playback of the first request, the same server stream can be used to satisfy both requests [7, 3]. In general, requests by multiple clients for the same video arriving within a short time duration can be batched together and served using a single stream. This is referred as *batching* in [7]. We can divide batching technique into two categories:

- **Scheduled Multicast**: When a server channel becomes available, the server selects a batch to multicast according to some scheduling policy. For instance, the *Maximum Queue Length* (MQL) [7], proposed by Dan *et al.*, selects the batch with the most number of pending requests to serve first. The objective of this approach is to maximize the server throughput. Other scheduled multicast schemes are presented in [7, 8, 1, 19].
- **Periodic Broadcast**: The videos are broadcast periodically, *i.e.,* a new stream is started every B

minutes (*batching interval*) for a given video. As a result, the worst service latency experienced by any subscriber is guaranteed to be less than B minutes independent of the current number of pending requests. Such a guarantee can generally influence the reneging behavior of clients, and therefore improve the server throughput. This technique is sometimes referred as *data centered* [2, 22] because the server channels are dedicated to individual video objects rather than users. Some recent periodic broadcast techniques are presented in [7, 22, 1].

It was shown in [7, 8] that a hybrid of the two techniques offered the best performance. In this approach, a fraction of the server channels is reserved and preallocated for periodic broadcast of the popular videos. The remaining channels are used to serve the rest of the videos using some scheduled multicast technique. Nevertheless, it was also shown in [7, 8] that the popularities of movies follow the Zipf distribution with the skew factor of 0.271. That is, most of the demand (80%) is for a few (10 to 20) very popular movies. This has motivated us to focus on the popular videos. In this paper, we introduce a novel technique for doing periodic broadcast. We assume that some existing scheduled multicast scheme is used to handle the less popular videos.

One of the earlier periodic broadcast schemes was proposed by Dan, Sitaram and Shahabuddin [7]. Since this approach broadcasts a given video every batching period, the service latency can only be improved linearly with the increases in the server bandwidth. To significantly reduce the service latency, *Pyramid Broadcasting* (PB) technique was introduced by Viswanathan and Imielinski [22]. In this scheme, each video file is partitioned into K segments of geometrically increasing size; and the server capacity is evenly divided into K logical channels. The ith channel is used to broadcast the ith segments of all the videos in a sequential manner. Since the first segments are very small, they can be broadcast a lot more frequently through the first channel. This ensures a smaller wait time for every video. A drawback of this scheme is that a buffer size which is usually more than 70% of the length of the video must be used at the receiving end. As a result, it is necessary to use disks to do the buffering. Furthermore, since a very high transmission rate is used to transmit each video segment, an extremely high disk bandwidth is required to write data to disk as quickly as it receives the video. To address these issues, Aggarwal, Wolf and Yu proposed a technique called *Permutation-Based Pyramid Broadcasting* (PPB) [1]. PPB is similar to PB except that each channel multiplexes among its own segments (instead of transmitting them serially), and a new stream is started every small period for each of these segments as in [7]. This strategy allows PPB to reduce both disk space and I/O bandwidth requirements at the receiving ends. The disk size, however, is still quite significant due to the exponential nature of the data fragmentation scheme. The sizes of successive segments increase exponentially causing the size of the last segment to be very large (typically more than 50% of the video file). Since the buffer sizes are determined by the size of the largest segment, using the same data fragmentation scheme proposed for PB limits the savings can be achieved by PPB. To substantially reduce the disk costs, we introduce in this paper a new data fragmentation technique and propose a different broadcasting strategy. The proposed technique also addresses the following implementation issue. In PPB, a client needs to tune into different logical subchannels to collect its data for a given data fragment if maximum saving in disk space is desirable. This synchronization mechanism is difficult to implement because the tunings must be done at the right moment during a broadcast. To avoid this complication, we only tune to the beginning of any broadcast as in the original PB. In other words, we are able to achieve better savings using a simpler technique.

The remaining of this paper is organized as follows. We discuss PB and PPB in more detail in Section 2 to make the paper self-contained. The proposed technique, *Skyscraper Broadcasting* (SB), is introduced in Section 3. The correctness of the proposed technique and its storage requirement are analyzed in Section 4. In Section 5, we present performance study to compare SB to both PB and PPB. Finally, we give our concluding remarks in Section 5.

2 Pyramid-based broadcasting schemes

To facilitate our discussion, we need to define the following notations:

- B: The server bandwidth in Mbits/sec.
- M: The number of videos being broadcast.
- D: The length of each video in minutes.
- K: The number of data segments in each video file.
- b: The display rate of each video in Mbits/sec.

PB and PPB share the same data fragmentation technique. The idea is to partition each video into K sequential segments of geometrically increasing sizes [21, 22, 2]. Let S_i^v denote the i-th fragment of video v. Its size in minutes, D_i, is determined as follows:

$$D_i = \begin{cases} \frac{D\cdot(\alpha-1)}{\alpha^K-1} & i=1, \\ D_1\cdot\alpha^{i-1} & \text{otherwise}, \end{cases}$$

where α is a number greater than 1. We will discuss how α is determined in PB and PPB shortly. For the moment, we note that $D_1, D_2, \cdots, D_K$ of each video form a geometric series with the factor α; and $\sum_{i=1}^{K} D_i = D$.

In PB scheme, the entire bandwidth is divided into K logical channels with $\frac{B}{K}$ Mbits/sec each. The i-*th* channel (or, Channel i) will periodically broadcast S_i^1, S_i^2, ... , S_i^M in turns, where $1 \leq i \leq K$. No other segments are transmitted through this channel. On the client side, it begins downloading the first data fragment of the requested video at the first occurrence, and start playing it back concurrently. For the subsequent fragments, it downloads the next fragment at the earliest possible time after beginning to play back the current fragment. Thus, at any point, the client downloads from at most two consecutive channels and consumes the data segment from one of them in parallel. The parameter α must be chosen in such a way to ensure that the playback duration of the current fragment must eclipse the worst latency in downloading the next fragment. Mathematically, we must have the following:

$$\frac{D_i}{b} \geq \frac{D_{i+1} \cdot M}{\frac{B}{K}} .$$

Substitute $\alpha \cdot D_i$ for D_{i+1}, we have:

$$\alpha \leq \frac{B}{b \cdot M \cdot K} .$$

To determine α, PB uses two methods as follows. The first method, denoted as *PB:a* for later references, first chooses $K = \lceil \frac{B}{bMe} \rceil$ and then computes α as $\alpha = \frac{B}{bMK}$, where e is the Euler's constant ($e \approx 2.72$.) The other method, denoted as *PB:b*, lets $K = \lfloor \frac{B}{bMe} \rfloor$ and then computes α as $\alpha = \frac{B}{bMK}$.

The access time of a video is equal to the access time of the first data fragment which is broadcast on Channel 1. Thus the worst wait time can be computed as follows:

$$\begin{aligned} Worst\ Wait\ Time &= \frac{M \cdot D_1 \cdot b}{\frac{B}{K}} \\ &= \frac{DMKb(\alpha - 1)}{B(\alpha^K - 1)}\ minutes. \end{aligned}$$

By letting K increase as B does, the service latency will improve exponentially with B. Since pipelining is used at the client end, a disk bandwidth of $b + 2 \cdot \frac{B}{K}$ Mbits/sec is required for each client. The first term "b" is the bandwidth required to support the playback. The number "2" in the second term is due to the fact that one channel is used to retrieve the current fragment while another channel is used to prefetch the next one. In terms of storage requirement, PB requires each client to have a disk space at least as large as $60 \cdot b \cdot (D_K - \frac{bKD_K}{B} + D_{K-1})$ Mbits since playing back S_{K-1}^v while receiving both S_{K-1}^v and S_K^v incurs the maximum space requirement.

We note that since α is kept around e, under a large B, the disk bandwidth and storage requirements approach $b(2Me + 1) \simeq 55.36b$ Mbits/sec and $bD(1 - \frac{1}{e})(1 - \frac{1}{Me} + \frac{1}{e}) \simeq 0.84(60bD)$ Mbits if $M = 10$, respectively. Hence disk bandwidth requirement is very high. Furthermore, each client must have a disk space large enough to buffer more than 80% of the video file in order to use this technique. To reduce these requirements, PPB scheme further partitions each logical channel in PB scheme into $P \cdot M$ subchannels with $\frac{B}{KPM}$ Mbits/sec each. A "replica" of each video fragment, say i, is now broadcast on each of the P logical subchannels with a phase delay of $\frac{D_i}{P}$ minutes. On each subchannel, one segment is repeatedly broadcast in its entirety. Since the subchannels are time-multiplexed on the logical channel, using smaller bandwidths to transmit the video segments reduces the disk space requirement at the receiving ends. To further reduce this requirement, PPB occasionally pauses the incoming stream to allow the playback to catch up. This is done by allowing a client to discontinue the current stream and tune to another subchannel, which broadcasts the same fragment, at a later time to collect the remaining data. This, however, is difficult to implement since a client must be able to tune to a channel during, instead of at the beginning of, a broadcast. This is significantly more complex than in the original PB scheme.

The storage requirement for PPB at each client can be computed as

$$\frac{60 \cdot b \cdot D \cdot M \cdot K \cdot (\alpha^K - \alpha^{K-2})}{B \cdot (\alpha^K - 1)}$$

Mbits. The required rate of disk transfer is simply equal to the sum of display rate and the rate of receiving data, which is $b + \frac{B}{K \cdot P \cdot M}$ Mbits/sec. The service latency is simply the access time of the first fragment, which is $\frac{D_1 \cdot M \cdot K \cdot b}{B} = \frac{D_1}{P + \alpha}$ minutes. We note that the methods to determine the design parameters K and α are different from those of PB. K is determined as $\lfloor \frac{B}{3 \cdot M \cdot b} \rfloor$, but is limited within the range $2 \leq K \leq 7$. The first method, denoted as *PPB:a*, chooses $P = \lfloor \frac{B}{M \cdot K \cdot b} - 2 \rfloor$ and then computes $\alpha = \frac{B}{M \cdot K \cdot b} - P$. On the other hand, the second method, denoted as *PPB:b*, lets $P = \lfloor \frac{B}{M \cdot K \cdot b} - 2 \rfloor$, but limits it to be at least 2. α are then computed as $\frac{B}{M \cdot K \cdot b} - P$. Since K is limited to 7, the access latency and storage requirement will eventually improve only linearly as B increases. As a comparison, the original PB scheme does not constrain the value of K. and is able to maintain the exponential latency improvement for the increases in the very large values of B. We shall show in Section 4 that PPB actually performs worse than PB.

3 Skyscraper Broadcasting Scheme

3.1 Channel Design

We divide the server bandwidth of B Mbits/sec into $\lfloor \frac{B}{b} \rfloor$ logical channels of b Mbits/sec each. These channels are allocated evenly among the M videos such that there are K $(= \lfloor \frac{B}{bM} \rfloor)$ channels for each video. To broadcast a video over its K dedicated channels, each video file is partitioned into K fragments using the data fragmentation scheme described in the next subsection. Each of these fragments are repeatedly broadcast on its dedicated channel at its consumption rate (*i.e.*, display rate).

3.2 Data Fragmentation

Instead of fragmenting the video files according to a geometric series $[1,\ \alpha,\ \alpha^2,\ \alpha^3,\ \ldots]$ as in the pyramid-based techniques, a series generated by the following recursive function is used in SB:

$$f(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2,\ 3, \\ 2f(n-1)+1 & n \ mod\ 4 = 0, \\ f(n-1) & n \ mod\ 4 = 1, \\ 2f(n-1)+2 & n \ mod\ 4 = 2, \\ f(n-1) & n \ mod\ 4 = 3, \end{cases}$$

or,

$$f(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2,\ 3, \\ (2 + 2\lfloor \frac{n}{2} \rfloor - n) f(n-1) + & \\ \quad (1 + 2\lfloor \frac{n}{2} \rfloor - n)(1 + \lfloor \frac{n - 4\lfloor \frac{n}{4} \rfloor}{2} \rfloor) & \text{otherwise.} \end{cases}$$

For illustration, we show the materialized series in the following:

$$[1, 2, 2, 5, 5, 12, 12, 25, 25, 52, 52, \ldots]\ .$$

We will refer to such a series as a *broadcast series* in this paper. The first number in the above series signifies that the size of the first fragment is one unit, (*i.e.*, D_1.) Similarly, the size of the second one is two units (*i.e.*, $2 \cdot D_1$); the third one is also two; the fourth one is five; and so forth. Additionally, we use W to restrict the segments from becoming too large. If some segment is larger than W times the size of the first segment, we force it to be W units. The rationale is that a bigger K-th fragment will result in a larger requirement on the buffer space at the receiving end as we will discuss in more detail shortly. We call this scheme *Skyscraper Broadcasting* (SB) due to the fact that stacking up the data fragments in the order they appear in the respective video file forms the shape of a very tall skyscraper (instead of a much shorter and very wide pyramid as in the case of PB and PPB.) We note that W is the *width* of the "skyscraper," which can be controlled to achieve the desired access latency as follows. The number of videos determines the parameter K. Given K, we can control the size of the first fragment, D_1, by adjusting W. A smaller W corresponds to a larger D_1. Since the maximum access latency is D_1, we can reduce the access latency by using a larger W. The relationship between W and access latency is given in the following formula:

$$Access\ Latency = D_1 = \frac{D}{\sum_{i=1}^{K} min(f(i), W)},$$

which can be used to determine W given the desired access latency.

3.3 Transmitting and Receiving of Segments

The transmitting of data fragments at the server end is straightforward. The server multiplexes among the $K \cdot M$ logical channels; each is used to repeatedly broadcast one of the $K \cdot M$ data fragments. At the client end, reception of segments is done in terms of *transmission group*, which is defined as consecutive segments having the same sizes. For example, in our broadcast series $[1, 2, 2, 5, 5, 12, 12, 25, 25, 52, 52, \ldots]$, the first segment forms the first group; the second and third segments form the second group (*i.e.*, "$2, 2$"); the fourth and fifth form the third group (*i.e.*, "$5, 5$"); and so forth. A transmission group $(A, A, \cdots, A)$ is called an *odd group* if A is an odd number; otherwise, it is called an *even group*. We note that the odd groups and the even groups interleave in the broadcast series. To receive and playback these data fragments, a client uses three service routines, an *Odd Loader*, an *Even Loader*, and a *Video Player*. The Odd Loader and the Even Loader are responsible for tuning to the appropriate logical channels to download the odd groups and even groups, respectively. Each loader downloads its groups one at a time in its entirety, and in the order they occur in the video file. These three routines share a local buffer. As the Odd Loader and Even Loader fill the buffer with downloaded data, the Video Player consumes the data at the rate of a broadcast channel. In the next section, we will discuss the space requirement for this buffer, and show that SB is able to support jitter-free playback.

4 Correctness and Storage Analyses

We recall that the video fragments are received by a client in terms of transmission groups. To investigate the correctness and analyze the storage requirement for SB, we need to examine three possible types of group transition as follows:

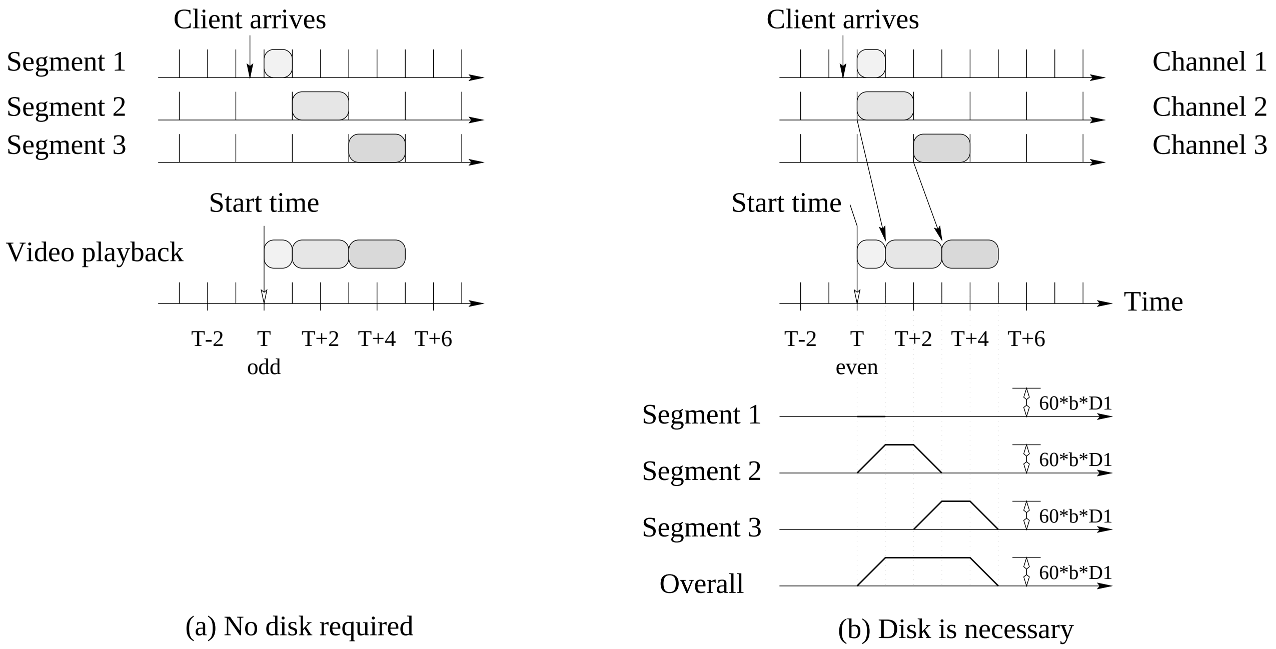


Figure 1: First transition type: $(1) \Rightarrow (2, 2)$.

1. $\underline{(1) \Rightarrow (2, 2)}$: This transition is a special case and only happens in the beginning of playing back a video.

2. $\underline{(A, A) \Rightarrow (2A + 1, 2A + 1)}$: This kind of transitions occurs when A is even. Transitions $(2, 2) \Rightarrow (5, 5)$ and $(12, 12) \Rightarrow (25, 25)$ are examples of this type.

3. $\underline{(A, A) \Rightarrow (2A + 2, 2A + 2)}$: This kind of transitions occurs when A is odd. Transitions $(5, 5) \Rightarrow (12, 12)$ and $(25, 25) \Rightarrow (52, 52)$ are examples of this type.

For the first type, since the *l.c.m.* (*least common multiple*) of 1 and 2 is 2, there can only be two possible scenarios as shown in Figure 1. Although channels 1, 2 and 3 repeatedly broadcast segments 1, 2 and 3, respectively, we show only the broadcast tuned in by some client. Without loss of generality, we use D_1 as one time unit. Let T be the start time of the video requested by the client. If T is odd, the client does not need to buffer the incoming data. In this case, the client can play back the video data as soon as they arrive. This is illustrated in Figure 1(a). Let us now focus on the other scenario shown in Figure 1(b). Since T is even, the client must start to prefetch the second group as soon as it begins to play back the first group at time T. At time $T + 2$, it must start to preload the second half of group 2, while playing back the first half of the same group. This pipelining then continues to play back the second half of group 2 while preloading the first half of group three, and so on. Obviously, the playback is jitter-free since the Video Player always finds prefeteched data in the buffer. In terms of storage requirement, the client in the second scenario has to prefetch D_1 minutes of data during every time unit. Hence the buffer size required is $60 \cdot b \cdot D_1$ Mbits as illustrated in Figure 1(b).

Possible cases of the second type, $(A, A) \Rightarrow (2A + 1, 2A + 1)$, are illustrated in Figure 2. The group (A, A) is composed of segments i and $i+1$, which are broadcast on channels i and $i + 1$, respectively. For convenience, only the first segment, segment $i+2$, of the group $(2A + 1, 2A + 1)$ is shown in the figure. Let t be the required playback time of group (A, A) or Segment i. We show in Figure 2 the various possible times for the client to start receiving group (A, A). Possible broadcast times for group $(2A + 1, 2A + 1)$ are also illustrated therein. We note that since A is even, the broadcast of group (A, A) must begin at some even time. However, since the *g.c.d.* (*greatest common divisor*) of A and $2A + 1$ is 1 (*i.e.*, mutually prime), the possible times to start receiving group $(2A + 1, 2A + 1)$ are $t, t + 1, \ldots, t + 2A$. As illustrated in the figure, the following six scenarios can happen:

- The Even Loader starts to download group (A, A) at time t, and the Video Player immediately plays back the data as soon as they arrive. The data from group $(2A + 1, 2A + 1)$ do not arrive at the Odd Loader until time $t + 2A$.
- The Even Loader starts to download group (A, A) at time t, and the Video Player immediately plays back the data as soon as they arrive. The data

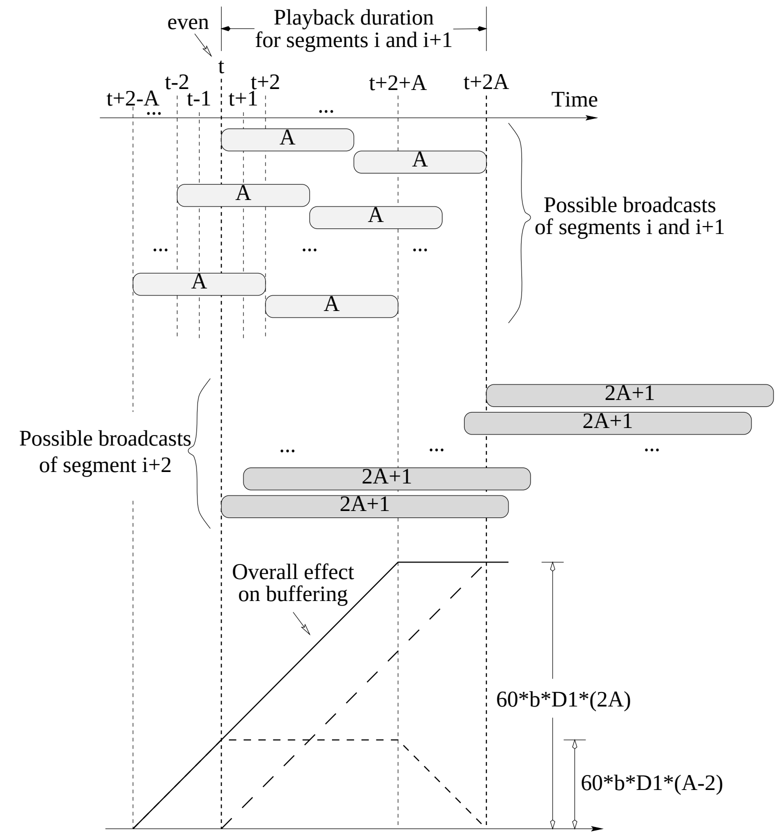


Figure 2: Second transition type: $(A, A) \Rightarrow (2A+1, 2A+1)$, A is even.

from group $(2A+1, 2A+1)$ do not arrive at the Odd Loader until time $t+2A-1$.

- The Even Loader starts to download group (A, A) at a time before t. The Video Player starts to play back (A, A) at time t. The data from group $(2A+1, 2A+1)$ do not arrive at the Odd Loader until time $t+2A$.

- The Even Loader starts to download group (A, A) at a time before t. The Video Player starts to play back (A, A) at time t. The data from group $(2A+1, 2A+1)$ do not arrive at the Odd Loader until time $t+2A-1$.

- The Even Loader starts to download group (A, A) at time t, and the Video Player immediately plays back the data as soon as they arrive. The data from group $(2A+1, 2A+1)$ arrives at the Odd Loader at time t.

- The Even Loader starts to download group (A, A) at a time before t; and the Video Player begins to play back the data as soon as they arrive. The data from group $(2A+1, 2A+1)$ arrives at the Odd Loader at time t.

We note that for the first scenario, no disk buffer is required to support the jitter-free playback. For the remaining cases, since the Video Player can always find the required data directly from the Even Loader or from the prefetched buffer, jitter-free is again guaranteed. The storage requirement for this case is illustrated in the plot shown at the bottom of Figure 2. The curves are based on the worst-case (in terms of storage requirement) scenario corresponding to the earliest possible broadcast of group (A, A), and the earliest possible broadcast of group $(2A+1, 2A+1)$. We explain the curves as follows:

- From $t+2-A$ to t, the Even Loader fills the disk buffer with data from group (A, A). As a result, the curve corresponding to group(A, A) rises during this period. The curve becomes flat for the duration from t to $t+2+A$ because the Player starts to consume the data at time t.

- The curve corresponding to group (A, A) drops after time $t+2+A$ because the Even Loader is idle while the Player continues to consume data.

- The curve corresponding to group $(2A+1, 2A+1)$ continues to rise until time $t+2A$. This is due to the fact that the Odd Loader fills the buffer with

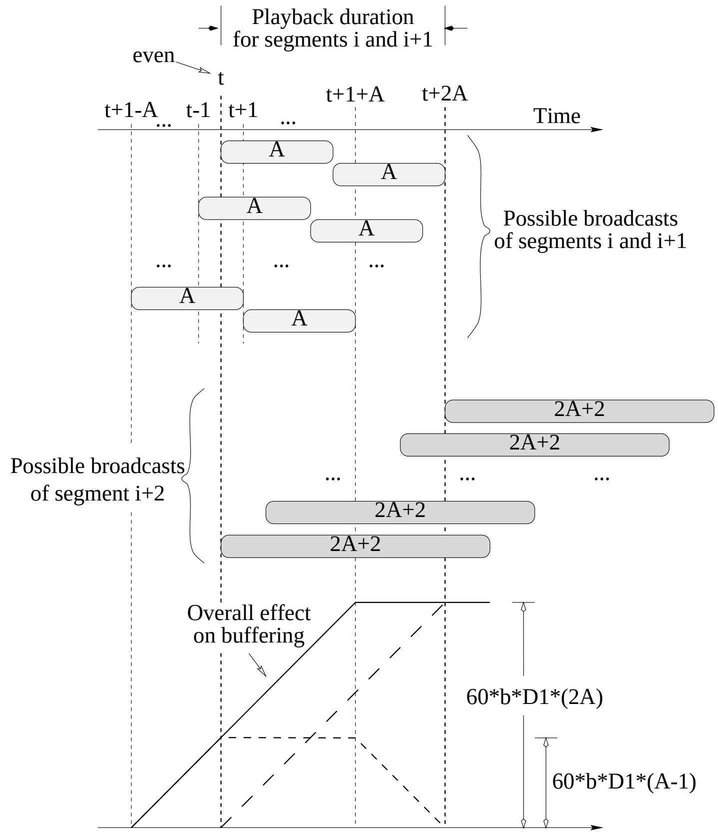


Figure 3: Third transition type: $(A, A) \Rightarrow (2A+2, 2A+2)$, A is odd and playback time of A is even.

data from segment $i+2$ during this period. The curve becomes flat after $t+2A$ because the Player starts to play back segment $i+2$ at that time.

- The curve labeled "overall effect" shows the aggregate effect of the other two curves. It shows that the storage requirement is $60 \cdot b \cdot D_1 \cdot 2A$.

Finally, Let us now examine the third type of group transition, namely $(A, A) \Rightarrow (2A+2, 2A+2)$. Since A must be odd, the broadcast of group (A, A) can start at either an odd time or an even time. The two cases are illustrated separately in Figure 3 and 4, respectively. Their interpretation is similar to that given for Figure 2. Basically, jitter-free playback is assured in either situation due to the following reasons:

- The two transmission groups are downloaded by two different loaders. Since we assume that each loader has the capability to receive data at the broadcast rate, the two download streams can occur simultaneously.

- Since the downloading of group (A, A) starts before or at t, and completes before prior to $t+2A$, the Video Player should be able to start the playback of (A, A) at time t, and continue to play back the video segment without any jitter.

- The playback of group $(2A+2, 2A+2)$ should not encounter any jitter either since the Even Loader starts to load this group no later than time $t+2A$ which is the time required to begin the playback of this group.

We note in Figure 4 that the client might be downloading both groups (A, A) and $(2A+2, 2A+2)$ simultaneously during the time period from $t-1$ to t. During this period, if the client also needs to download group $(\frac{A-1}{2}, \frac{A-1}{2})$, then SB will not work since it allows only two downloading streams at any one time. Fortunately, this can never happen since $(\frac{A-1}{2}, \frac{A-1}{2})$ is an even group, and as such its playback must not end at time t which is odd. We note that if $(\frac{A-1}{2}, \frac{A-1}{2})$ ends at an odd time, then the next broadcast of this group will necessarily start at an odd time. That is not possible for any odd groups. The playback of group $(\frac{A-1}{2}, \frac{A-1}{2})$, therefore, must terminate by time $t-1$.

The computation of storage requirements under the third type of group transition are illustrated at the bottom of Figures 4 and Figure 3. Since the explanations are similar to that discussed for Figure 2, we will not

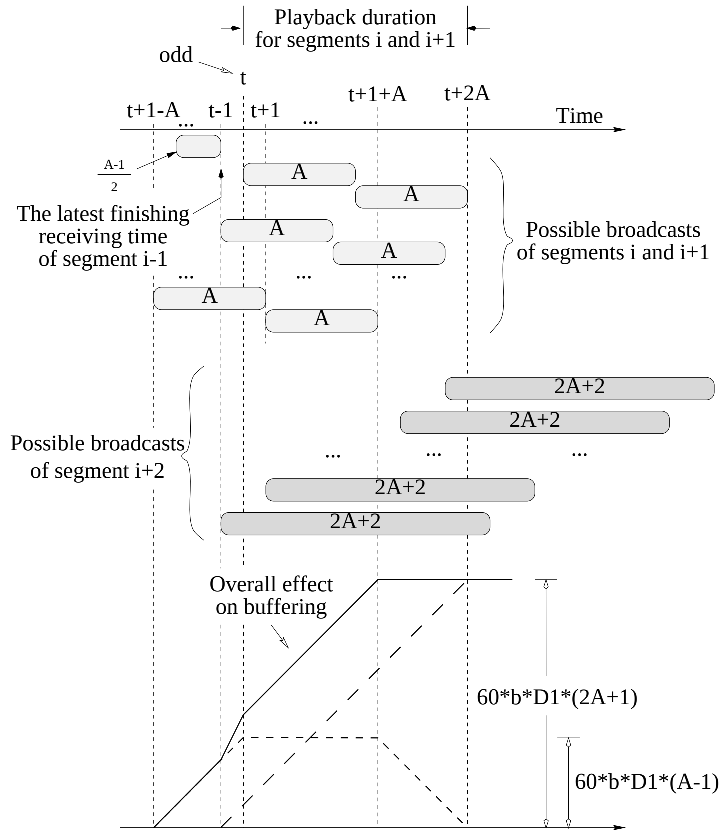


Figure 4: Third transition type: $(A, A) \Rightarrow (2A+2, 2A+2)$, A is odd and playback time of A is odd.

discuss them any further. Comparing the storage requirements under the various group transition types, we notice that the cases illustrated in Figures 2 and 4 are most demanding. Since the buffer must be large enough to accommodate the most demanding condition, we conclude that the storage requirement for SB is $60 \cdot b \cdot D_1 \cdot (W-1)$ which is obtained by applying the formula, given in Figure 2, to the last group transition of the series $(X, X) \Rightarrow (W, W, \cdots, W)$.

5 Performance study

In this section, we present the performance study for our SB scheme. For comparison, pyramid-based schemes, PB:a, PB:b, PPB:a and PPB:b as discussed in Section 2, are also investigated. To ensure the fairness, the desired values for the design parameters (i.e., K, P, and α) are determined for each technique using its own methodology. The other parameters are as follows. We assume that there are $M = 10$ popular videos requiring periodic broadcast. The playback duration of each video is $D = 120$ minutes. They are compressed using MPEG-1, so that the average playback rate is $b = 1.5$ Megabits per second. We choose *storage requirement, I/O bandwidth* and *access latency* as our performance metrics. The formulas for computing these parameters have been determined for PB and PPB in Section 2. We derive the corresponding formulas for SB in the following:

$$\begin{array}{c} I/O \\ bandwidth \\ requirement \end{array} = \begin{cases} 0 & W = 1 \text{ or } K = 1, \\ 2 \cdot b \quad Mb/s & W = 2 \text{ or } K = 2, 3, \\ 3 \cdot b \quad Mb/s & \text{otherwise;} \end{cases}$$

$$Access\ latency = D_1 = \frac{D}{\sum_{i=1}^{K} min(f(i), W)}\ minutes;$$

$$Storage\ requirement = 60 \cdot b \cdot D_1 \cdot (W-1)\ Mbits.$$

For the convenience of the reader, we repeat the formulas for each scheme in the tables (Table 1 and 2). These formulas will be used to make the plots discussed in the following subsections.

5.1 Determining the Design Parameters

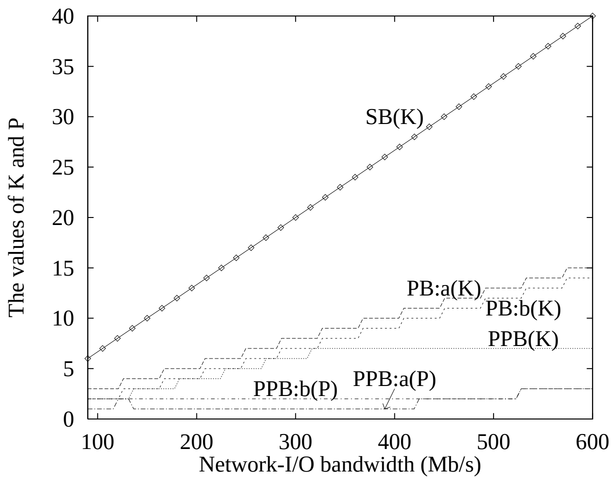
To compare the performance of the three broadcast schemes, we varied the network-I/O bandwidth from 100 Mbits/sec to 600 Mbits/sec. The rationales for choosing this range of network-I/O bandwidth in our study are as follows. First, PB and PPB do not work if the server bandwidth is less than 90 Mbits/sec (*i.e.,* α becomes less than one). Second, 600 Mbits/sec is large enough to show the trends of the various design

TECHNIQUES	I/O BANDWIDTH (Mbits/sec)	ACCESS LATENCY (minute)	BUFFER SPACE (Mbit)
PB	$b + 2\frac{B}{K}$	$\frac{DMKb(\alpha-1)}{B(\alpha^K-1)}$	$60 \cdot b(D_K - \frac{bKD_K}{B} + D_{K-1})$
PPB	$b + \frac{B}{KPM}$	$\frac{D_1MKb}{B}$	$\frac{60 \cdot bDMK(\alpha^K - \alpha^{K-2})}{B(\alpha^K-1)}$
SB	$2b$ or $3b$	$\frac{D}{\sum_{i=1}^{K} min(f(i),W)} (= D_1)$	$60 \cdot bD_1(W-1)$

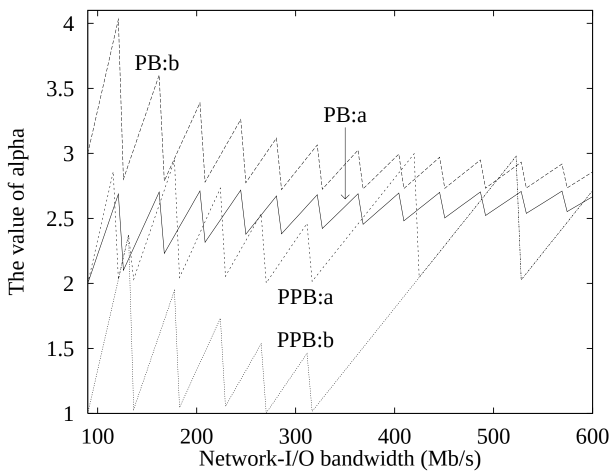
Table 1: Performance computation.

TECHNIQUES	K	P	α
PB:a	$\lceil \frac{B}{bMe} \rceil$	N/A	$\frac{B}{bMK}$
PB:b	$\lfloor \frac{B}{bMe} \rfloor$	N/A	$\frac{B}{bMK}$
PPB:a	$\lfloor \frac{B}{3Mb} \rfloor$, $2 \leq K \leq 7$	$\lfloor \frac{B}{MKb} - 2 \rfloor$	$\frac{B}{MKb} - P$
PPB:b	$\lfloor \frac{B}{3Mb} \rfloor$, $2 \leq K \leq 7$	$\lfloor \frac{B}{MKb} - 2 \rfloor$, $P \geq 2$	$\frac{B}{MKb} - P$
SB	$\lfloor \frac{B}{bM} \rfloor$	N/A	N/A

Table 2: Design parameters determination.



(a) The values of K & P



(b) The value of α

Figure 5: The values of K, P and α under different network-I/O bandwidth.

schemes.

To facilitate our studies, we first need to examine the design parameters used by each scheme. The desired values for these parameters under various network-I/O bandwidths were computed using the formulas given in the previous sections, and plotted in Figure 5. The curves for K and for P are labeled with "(K)" and "(P)", respectively. The curves for α are plotted separately in Figure 5(b). We observe that the K values are much larger for the proposed scheme under various network-I/O conditions. This means that SB uses a larger number of significantly smaller data fragments. This characteristics result in less demanding on storage bandwidth, shorter access latency and smaller storage requirement as we will see in the following subsections.

5.2 Disk bandwidth requirement

In this study, we compare the disk bandwidth requirements of the broadcasting schemes under various network-I/O bandwidths. We investigated the proposed technique under four different values of W (*i.e.*, the width of the "skyscraper"), namely 2, 52, 1705, and 54612. They are the values of the 2-nd, 10-th, 20-th and 30-th elements of the broadcast series, respectively. The reason for not considering larger elements in the series is due to the fact that we limit our study to network-I/O bandwidth less than 600 Mbits/sec. Under this condition, Figure 5(a) indicates that the desired values for W should correspond to K values less than 40. We note that these values of W can be computed using the series generating function given in Section 3.2.

The results of this study are plotted in Figure 6. For references, we also show the lines corresponding to b, $4 \cdot b$, $5 \cdot b$ and $50 \cdot b$, where b is the playback rate and is equal to 1.5 Mbits/sec or 0.1875 MBytes/sec. We observe that SB and PPB have similar disk bandwidth requirements

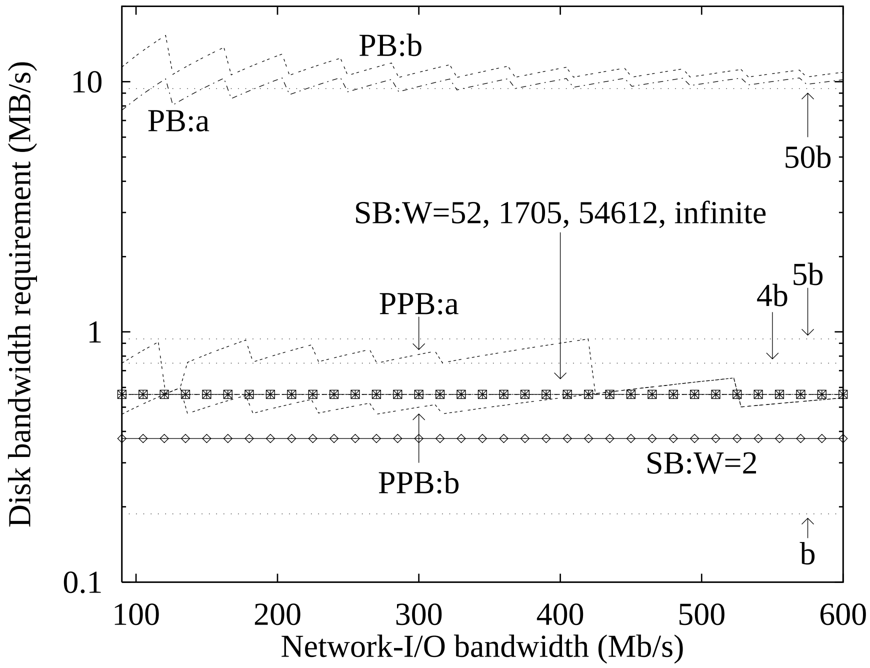


Figure 6: Disk bandwidth requirement (MBytes/sec).

at the receiving ends. The requirement for SB, however, can be lowered if we select W to be 2. As we will see later that a smaller W reduces storage bandwidth and space requirements with some sacrifice on access latency. In practice, we can control W, or the width of the skyscraper, to achieve the desired combination of storage bandwidth requirement, disk space requirement, and access latency. We note that the curves for SB are consistent with our analysis in Section 3 showing that SB requires only $3 \cdot b$ disk bandwidth to ensure jitter-free performance regardless of the W values. Our results are also consistent with those observed in [1] in that PB is very demanding on the storage-I/O bandwidth. It is shown in Figure 6 that an average bandwidth as high as 50 times the display rate (about 10 MBytes/sec) is required by PB.

5.3 Access latency analysis

The performance in terms of access latency is investigated in this study. Again, we varied the network-I/O bandwidth between 100 Mbits/sec and 600 Mbits/sec, and observe its effect on the access latency. The plots are shown in Figure 7.

We observe that the access latency of PPB can be quite significant. For instance, if the access latency is required to be less than 0.5 minutes, then we must have a network-I/O bandwidth of at least 300 Mbits/sec in order to use PPB. In terms of SB, W can be controlled to offer the best performance. More specifically, larger W values can be used to keep the access latency low. Nevertheless, improving the latency to well below 0.3 minutes is practically insignificant. We will observe in the next section that it is desirable to keep W small in order to reduce the storage costs at the receiving ends. Therefore, we must make a trade-off between access latency and buffer space requirement. We will revisit this issue in the next section after examining the storage requirement. We note that PB offers excellent access latency. However, as we have discussed, improving the latency from 0.1 minutes to 0.0001 minutes and beyond is not very interesting.

5.4 Storage requirement

The effect of the network-I/O bandwidth on the disk space requirement under various broadcasting schemes is plotted in Figure 8. As shown in the figure, PB scheme requires each client to have more than 1.0 GBytes of disk space, which is more than 75% of the length of a video. PPB scheme reduces this requirement to about 250 MBytes. The savings, however, are accomplished by sacrificing the access latency as discussed in the last subsection. For instance, when B is about 320 Mbits/sec, PPB:b requires only 150 MBytes or so of disk space. Unfortunately, its access latency in this case is as high as five minutes. Under the same situation, SB scheme with $W = 2$ has smaller access latency and requires only 33 MBytes of disk space at the receiving end.

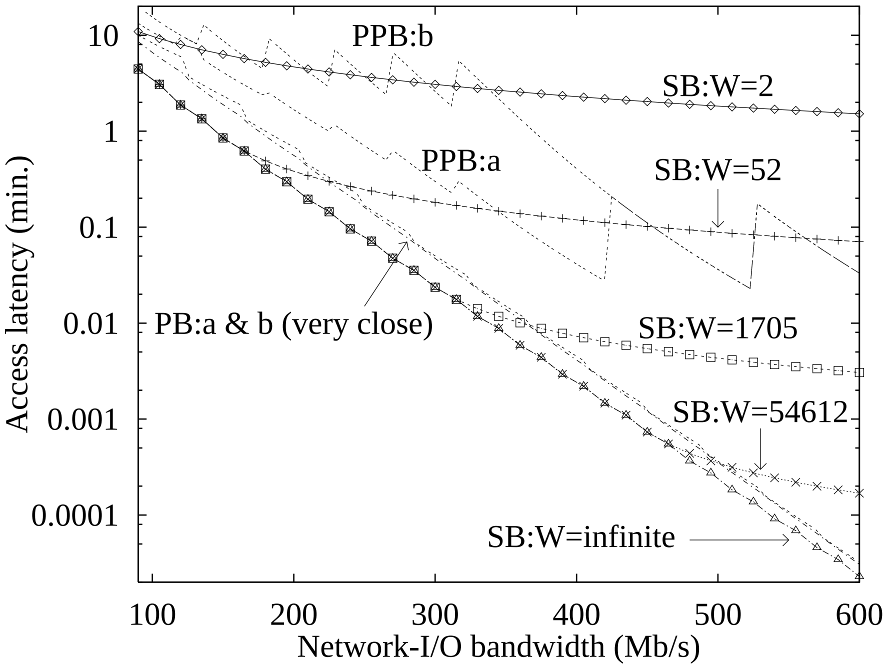


Figure 7: Access latency (minutes).

To determine a good W, we can cross-examine Figure 7 and Figure 8. If the network-I/O bandwidth of the server is greater than 200 Mbits/sec, then "$W = 52$" is a good choice for the workload. It offers an access latency of approximately 0.1 minute. This good performance can be had for a buffer space of less than 200 MBytes. For instance, if the network-I/O bandwidth is 600 Mbits/sec, each client needs only 40 MBytes of buffer space in order to enjoy an access latency of about 0.1 minutes. Such a combined benefit is many folds better than anything that can be achieved by PB or PPB. While PB and PPB must make trade-off between access latency, storage costs, and disk bandwidth requirement, the proposed scheme allows the flexibility to win on all three metrics.

6 Concluding Remarks

Network-I/O has been identified as a serious bottleneck in today's media servers. Many researchers have shown that broadcast is a good remedy for this problem. In this paper, we surveyed several broadcasting schemes. We discussed drawbacks in the current designs, and proposed an alternative approach, called *Skyscraper Broadcast* (SB), to address the problems.

SB is a generalized broadcasting technique for video-on-demand applications. Each SB scheme is characterized by a broadcast series and a design parameter called the *width* of the "skyscraper." In this paper, we focus on one broadcast series which is used as an example to illustrate the many benefits of the proposed technique. We showed that the *width* factor can be controlled to optimize the desired combination of storage costs, disk bandwidth requirement and access latency. Our performance study indicates that SB can achieve significantly better performance than the latest techniques known as *Pyramid Broadcasting*. Although the original *Pyramid Broadcasting* (PB) offers excellent access latency, it requires very large storage space and disk bandwidth at the receiving end. *Permutation-Based Pyramid Broadcasting* (PPB) is able to address these problems. However, this is accomplished at the expense of a larger access latency and more complex synchronization. With SB, we are able to better these schemes on all three metrics.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proc. of the IEEE Int'l Conf. on Multimedia Systems '96*, Hiroshima, Japan, June 1996.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proc. of the IEEE Int'l Conf. on Multimedia Systems '96*, Hiroshima, Japan, June 1996.

[3] D. P. Anderson. Metascheduling for continuous media. *ACM Trans. on Computer Systems*, 11(3):226–252, August 1993.

[4] W. J. Bolosky, J. S. Barrera, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P.

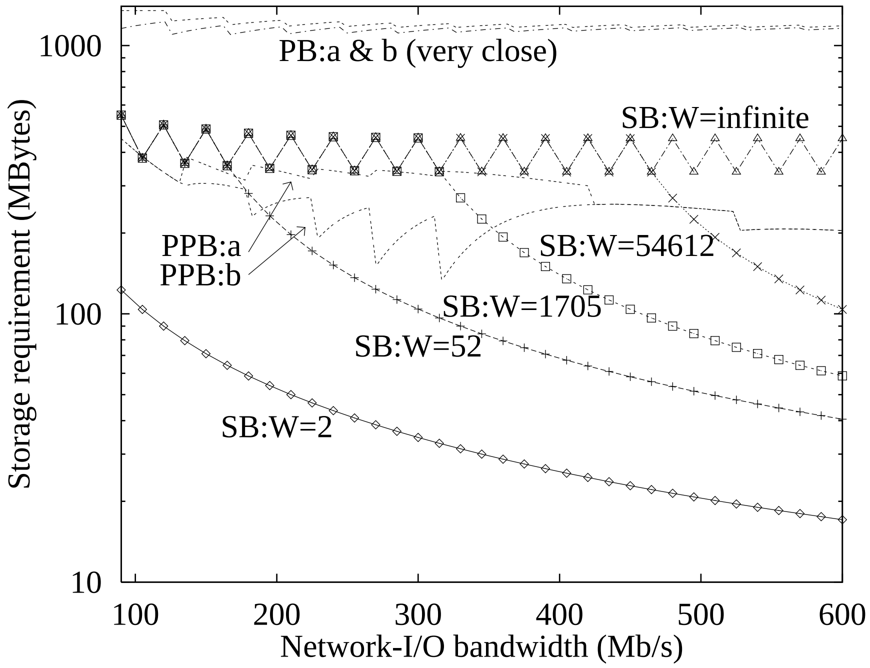


Figure 8: Storage requirement (MBytes).

Myhrvold, and R. F. Rashid. The tiger video fileserver. In *Proc. of the 6th Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, April 1996.

[5] J. Y. L. Boudec. The Asynchronous Transfer Mode: A tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[6] A. Dan, Y. Heights, and D. Sitaram. Generalized interval caching policy for mixed interactive and long video workloads. In *Proc. of SPIE's conf. on Multimedia Computing and Networking*, pages 344–351, San Jose, California, January 1996.

[7] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia*, pages 15–23, San Francisco, California, October 1994.

[8] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–121, June 1996.

[9] C. S. Freedman and D. J. DeWitt. The SPIFFI scalable video-on-demand system. In *Proc. of the 1995 ACM SIGMOD Conf.*, pages 352–363, San Jose, California, May 1995.

[10] K. A. Hua, S. Sheu, and J. Z. Wang. Earthworm: A network memory management technique for large-scale distributed multimedia applications. In *Proc. of the 16th IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[11] IEEE Standard 802.6. *Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN)*, Dec. 1990.

[12] K. Keeton and R. H. Katz. Evaluating video layout strategies for a high-performance storage server. *Multimedia Systems*, 3:43–52, 1995.

[13] D. J. Marchok, C. Rohrs, and M. R. Schafer. Multicasting in a growable packet (ATM) switch. In *IEEE INFOCOM*, pages 850–858, 1991.

[14] Y. Oyang, M. Lee, C. Wen, and C. Cheng. Design of multimedia storage systems for on-demand playback. In *Proc. of Int'l Conf. on Data Engineering*, pages 457–465, Taipei, Taiwan, March 1995.

[15] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. A low-cost storage server for movie on demand databases. In *Proc. of Int'l Conf. on VLDB*, pages 594–605, Santiago, Chile, September 1994.

[16] B. Özden, R. Rastogi, A. Silberschatz, and C. Martin. Demand paging for video-on-demand servers. In *Proc. of the IEEE Int'l Conf. on Multimedia Computing and Systems*, pages 264–272, Washington, DC, May 1995.

[17] M. A. Rodrigues. Erasure node: Performance improvements for the IEEE 802.6 MAN. In *IEEE INFOCOM*, pages 636–643, San Francisco, California, June 1990.

[18] D. Rotem and J. L. Zhao. Buffer management for video database systems. In *Proc. of Int'l Conf. on Data Engineering*, pages 439–448, Taipei, Taiwan, March 1995.

[19] S. Sheu, K. A. Hua, and T. H. Hu. Virtual Batching: A new scheduling technique for video-on-demand servers. In *Proc. of the 5th DASFAA'97*, Melbourne, Australia, April 1997.

[20] H. M. Vin and P. V. Rangan. Designing a multiuser HDTV storage server. *IEEE Journal on Selected Areas in Communications*, 11(1):152–164, Jan. 1993.

[21] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video on demaind service. In *IEEE Multimedia Computing and Networking Conference*, volume 2417, pages 66–77, San Jose, California, 1995.

[22] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Mulitmedia Systems*, 4(4):197–208, August 1996.