

Improving Server Broadcast Efficiency through Better Utilization of Client Receiving Bandwidth

Ashwin Natarajan

Ying Cai

Johnny Wong

Department of Computer Science

Iowa State University

Ames, IA 50011

E-mail: {ashwin, yingcai, wong}@cs.iastate.edu

Abstract

Periodic broadcast is a cost-effective solution for disseminating popular videos. This strategy has the potential to serve a very large community with minimal broadcast bandwidth: regardless of the number of video requests, the worst service latency to all clients is constant. Although many efficient schemes have been proposed, most of them impose some rigid requirement on client receiving bandwidth. They either demand clients to have the same bandwidth as the video server, or limit them to receive no more than two video streams at any one time. In our previous work, we addressed this problem by proposing a Client-Centric Approach (CCA). Unlike any other technique, CCA takes both server broadcast bandwidth and client receiving bandwidth into design consideration. More specifically, CCA allows clients to use all their receiving capability for prefetching broadcast data. Therefore, given a fixed broadcast bandwidth, CCA can achieve shorter broadcast period with an improved client communication capability. In this paper, we present a novel technique to further leverage client bandwidth for more efficient video broadcast. We prove the correctness of this new technique and provide analytical evaluations to show that with the same client bandwidth, it achieves significantly better performance than CCA.

KEYWORDS: *Multimedia communication, video on demand, periodic broadcast, service latency.*

1 Introduction

When a video is highly demanded, periodic broadcast is a very cost-effective dissemination strategy. By broadcasting a video periodically, say, every t time units, this approach can guarantee a constant worst service latency to clients, regardless of their number. The earliest periodic broadcast technique is *staggered*

broadcasting studied in [1, 2]. Given a broadcast bandwidth n times of a videos playback rate, this scheme broadcasts the video every $\frac{|v|}{n}$ time units, where $|v|$ is the video length. Since it requires each client to have only one-channel receiving bandwidth and does not need buffer at receiving ends, this scheme is low-cost in terms of implementation. Unfortunately, this simple approach can reduce the broadcast period only linearly with respect to the increase of broadcast bandwidth.

To improve the broadcast efficiency, many advanced techniques have been proposed, including *Pyramid Broadcasting* [3, 4], *Skyscraper Broadcasting* [5], *Pagoda Broadcasting* [6, 7, 8], just to name a few. These studies show that broadcast latency can be dramatically reduced if clients can download video data at a speed higher than video playback rate. Most of them, however, are designed with some rigid requirement on client receiving bandwidth: they either limit clients to receive no more than two video streams at any one time, or demand them to have the same bandwidth as the video server. For instances, Pyramid Broadcasting works best when clients can receive data at 2.6 times of video playback rate; Skyscraper Broadcasting assumes the receiving bandwidth at client sites is two times of video playback rate; Other techniques, such as Pagoda Broadcasting and its variations, require each client to have receiving bandwidth equal to server broadcast bandwidth. The techniques with the former limitation are intended for clients with low receiving bandwidth. They cannot perform any better in the presence of more client receiving capability. As for the techniques with the latter drawback, both server and client bandwidth must be augmented at the same pace in order to reduce broadcast latency. Such techniques are infeasible in many cases since increasing receiving bandwidth at all client sites would

require a revamp of an entire network infrastructure.

In [9, 10], we proposed a *Client-Centric Approach* (CCA) to address the aforementioned problem. Unlike any other techniques, CCA takes both server broadcast bandwidth and client receiving capability into consideration. This scheme organizes system resource into many *channels*, each can sustain one video stream at its regular playback rate. To broadcast a video over k channels, CCA partitions the video into k segments and broadcasts each segment repeatedly using one dedicated channel. Assuming client receiving bandwidth is c channels, these segments are organized into $\lceil \frac{k}{c} \rceil$ groups: the first c segments form the first group, the next c segments form the next group, ..., and so forth. Since each client can access c channels simultaneously, the sizes of video segments within one group is made to grow exponentially. To ensure that a client can download group by group continuously, the first segment in each group is made the same size as the last segment in the previous group.

Until now, CCA is the only technique that is able to leverage client receiving bandwidth for more efficient broadcast. The more channels clients can download, the less broadcast latency can be achieved. In this paper, we refine this technique and present an enhanced version, called *CCA+*. Similar to CCA, the new scheme partitions video segments into groups based on client receiving bandwidth. However, the sizes of segments within one group can now grow faster making the first segment even smaller. As a result, with the same broadcast and receiving bandwidth, the new technique is able to reduce broadcast latency more significantly than CCA. Noticeably, when client receiving bandwidth is two channels, our new technique can make broadcast latency smaller than Skyscraper Broadcasting, which is the best scheme in this setting up to date.

The remainder of this paper is organized as follows. We discuss CCA in more detail in Section 2. In Section 3, we investigate a motivation example and then provide a generalized solution in 4. We compare its performance with CCA in Section 5. Finally, we give our concluding remarks in Section 6.

2 Related Work

To broadcast a video over k channels, CCA partitions the video into k segments, $S_1, S_2, \dots,$ and S_k , each is broadcast repeatedly on its own channel. The size of each segment is determined based on client receiving capability. If we assume each client can access c channels simultaneously, then the size of the i th segment, denoted as $|S_i|$, can be calculated as follows:

$$|S_i| = \begin{cases} 1, & \text{if } i = 1, \\ 2 \cdot |S_{i-1}| & \text{if } i \bmod (c + 1) \neq 0, \\ |S_{i-1}| & \text{if } i \bmod (c + 1) = 0. \end{cases}$$

According to the above formula, these k segments can be organized into $\lceil \frac{k}{c} \rceil$ groups: the first c segments form the first group, the next c segments form the second group, ..., and the remaining segments form the last group. Such segmentation has two characteristics:

- Within one group, the sizes of segments grow exponentially. For example, for the segments in the first group, $|S_1| = 1, |S_2| = 2 \cdot |S_1|, \dots, |S_c| = 2 \cdot |S_{c-1}|$;
- The first segment in one group has the same size as the last segment in the previous group.

As an example, consider $k = 6$ and $c = 3$ (i.e., the server uses six channels to broadcast and each client can receive data from three channels simultaneously). In this case, the video is partitioned into six segments: $S_1, S_2, S_3, S_4, S_5,$ and S_6 . The sizes of them are 1, 2, 4, 4, 8, and 16, respectively. We will refer to $[1, 2, 4, 4, 8, 16]$ as a *broadcast series* corresponding to $k = 6$ and $c = 3$. Figure 1 shows a broadcast of this video.

At receiving ends, clients download the video segments group by group. To download video segments in one group, the client listens to all corresponding channels and downloads a segment as soon as a new broadcast of the segment starts. It is possible that a client may have to download all segments within one group simultaneously, in which case the client has to use all its receiving bandwidth. After the client finishes download the last segment in a group, it continues to download the segments in the next group. Since the last segment in one group is made to be the same size as the first segment in the next group, the client can always shift smoothly to download the segment in the next group.

With CCA, the more receiving bandwidth clients have, the more efficient broadcast can be achieved. Consider the example of using six channels to broadcast a video. If each client can access only one channel at any one time, CCA has the same performance as the native Stagger Broadcasting: the broadcast period is reduced to $\frac{1}{6}$ of the video length. If we assume each client can download two channels simultaneously, the video will be partitioned into six segments with sizes of 1, 2, 2, 4, 4, 8, respectively. Since the first segment is $\frac{1}{1+2+2+4+4+8}$ of the video length, the broadcast period

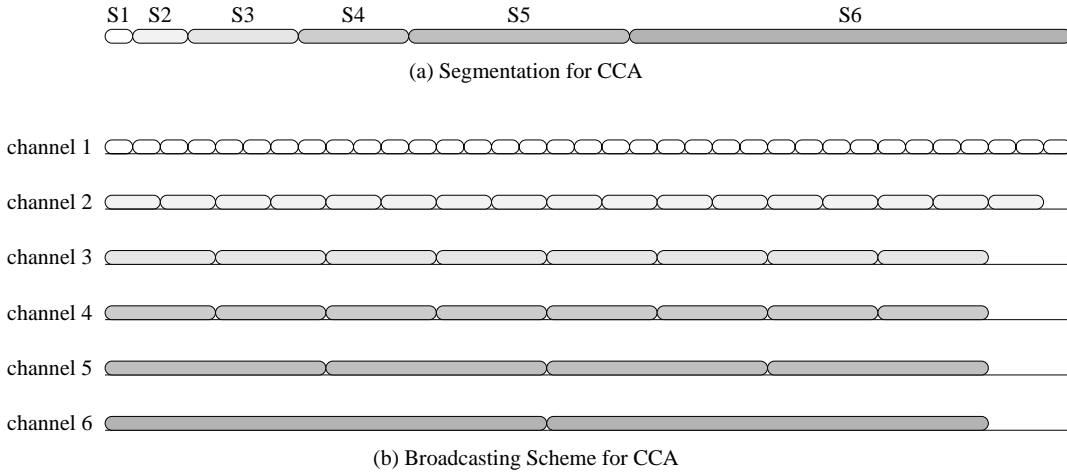


Figure 1: Client-Centric Approach ($k=6, c=3$)

is reduced to $\frac{1}{21}$ of the video length. Similarly, if $c = 3$, the broadcast period is reduced to $\frac{1}{1+2+4+4+8+16} = \frac{1}{35}$ of the video length. When the client receiving bandwidth is increased to six channels (i.e., $c = 6$), the broadcast period becomes $\frac{1}{1+2+4+8+16+32} = \frac{1}{63}$ of the video length. In this case, the broadcast latency is reduced exponentially with respect to the client receiving bandwidth.

3 Motivation Example

In CCA, a broadcast series grows faster when clients have more receiving bandwidth. Given a video and a fixed broadcast bandwidth, a faster growth of broadcast series makes the first segment smaller, resulting in a shorter broadcast latency. Thus, the key to reduce broadcast latency is to make broadcast series grow as fast as possible under the condition that client playback continuity is ensured. To investigate the limitation of growing a broadcast series, we start from a motivation example where each client has two-channels receiving capability.

Given two-channel receiving bandwidth and k broadcast channels, CCA partitions a video into k segments, say, S_1, S_2, \dots, S_k , each two forms a group. Obviously, the fastest series for the first two segments is $[1, 2]$, i.e., $|S_1| = 1$ and $|S_2| = 2$. To guarantee group continuity, we simply make the size of the first segment in one group equal to that of the last segment in the previous group. Thus, $|S_3| = |S_2| = 2$. Now the question is, what is the largest possible size for each of the remaining segments, S_4, \dots, S_k ? To solve this problem, we can try to determine the sizes of segments group by group. Once we fix the segment sizes in one group, we then use them to find out the maximum size of each segment in the next group.

Assume A and B are two consecutive segments in one group and their sizes, $|A|$ and $|B|$, are known. Let C and D be the two consecutive segments in the next group. Since C is the first segment in this group, we can fix its size, i.e., $|C| = |B|$. As for segment D , its size $|D|$ is limited and cannot be larger than $|A| + |B| + |C|$. Consider at time 0, when the broadcast of all four segments starts simultaneously. A client served at this broadcast period will download the current broadcast of A and B and then the next broadcast of C and D . Note that after A, B , and C are all downloaded, there are at least $|A|$ time units of data remaining in the buffer. This is simply because these three segments, totally $|A| + |B| + |C|$ time units of video data, are downloaded within $|B| + |C|$ time units. Thus, the client must be able to access segment D in the next $|A|$ time units. In other words, a new broadcast of D must start no later than $|A|$ time units. As a result, the size of D is limited to $|A| + |B| + |C|$.

Knowing that $|D| \leq |A| + |B| + |C|$, we try to make D as large as possible within that range while ensuring client playback continuity. Before we proceed for a solution, we introduce the following *alignment rule*: Given two segments X and Y , if their broadcast both start at time 0 and are repeated on their own channels, then the distance between any two broadcasts of X and Y must be a multiple of $HCF(|X|, |Y|)$, where HCF stands for *Highest Common Factor*. In other words, the smallest possible distance between two broadcast occurrences of X and Y is $HCF(|X|, |Y|)$. The rule can be proved easily. Since $|X|$ is a multiple of $HCF(|X|, |Y|)$, whenever it starts or finishes, the time, say T_x , must be a multiple of $HCF(|X|, |Y|)$. Similarly, if a broadcast of Y

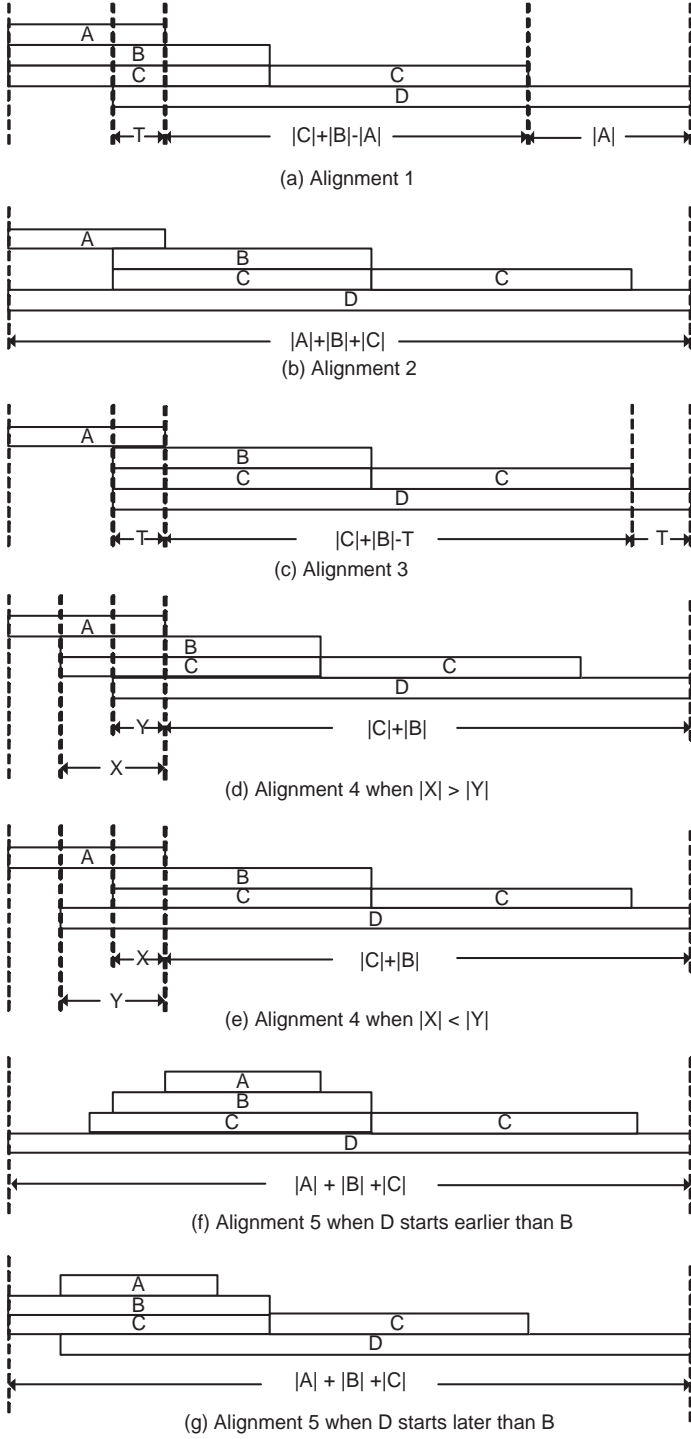


Figure 2: All Possible Broadcast Alignments of A , B , and D

starts or finishes at time T_y , T_y must be a multiple of $HCF(|X|, |Y|)$. Because the first broadcast of X and Y starts at the same time, $|T_x - T_y|$ must be a multiple of $HCF(|X|, |Y|)$.

To find out the maximum size of segment D , we consider all possible broadcast alignments of the four segments. Because B and C have the same size and their broadcasts always start at the same time, we need to consider only the broadcast alignments of A , B , and D . Figure 2 shows all possible broadcast alignments of these three segments. The type of alignment determines the largest possible value of D that can be used. We analyze them as follows:

- *Alignment 1:* A and B start at the same time. A client served in such broadcast period $|A|$ time units of data remaining in the buffer after it finishes downloading A , B , and C . This means that as long as a new broadcast of D start no later than $|A|$ time units, the playback continuity will not be broken. Thus, we can make $|D|$ at least equal to $|B| + |C|$. Assume the current broadcast of D starts T ($T > 0$) time units ahead of the current broadcast of A . According to the alignment rule, the smallest possible interval T is $HCF(|A|, |D|)$. Therefore, as long as the value of $|D|$ is not larger than $|B| + |C| + HCF(|A|, |D|)$, the playback continuity is ensured in this alignment case. Since $0 \leq |D| \leq |A| + |B| + |C|$, we can try all possible value in the range and use the largest one as the size of D , which satisfies $|D| \leq |B| + |C| + HCF(|A|, |D|)$.
- *Alignment 2:* A and D start at the same time. In this case, the size of D can be as large as $|A| + |B| + |C|$. This is due to the fact that a client downloading the current broadcast of A will need to download the next broadcast of D , which does not have to start until all A , B , and C are downloaded.
- *Alignment 3:* B and D start at the same time. We assume B and D start T time units before the current broadcast of A finishes. Since D starts with B at the same time, a client downloading the current broadcast of B will need to download the next broadcast of D . When the current broadcast of D starts, the client has consumed $|A| - |T|$ units of video data. It will then download in parallel the remaining T time units of A and the first T time units of B . After B is download, the client starts to download C . Thus, at the time it finishes downloading C , the client still has T time units of

data in its buffer. This means that a new broadcast of D must start in the next T time units. As a result, the size of D must be equal to $|C| + |B| + T$. According to the alignment rule, the smallest possible T is $HCF(|A|, |D|)$. Thus, the same as in alignment 1, we can try all possible value in between 0 and $|A| + |B| + |C|$, and use the largest one which satisfies $|D| \leq |B| + |C| + HCF(|A|, |D|)$ as the value of $|D|$.

- *Alignment 4:* Neither A nor D starts at the same time as other segments. Assume B starts X time units before A finishes and D starts Y time units before A finishes. X can be greater than Y and vice-versa as the figure shows. Note that when $|X| = |Y|$, it becomes Alignment 3. For a client downloading the current broadcast of A , it will download D in its next broadcast occurrence because the current broadcast of D starts before the current broadcast of A ends. To find the maximum value of D , we first consider the case when $|X| > |Y|$, illustrated in Figure 2(d). In this case, after a client finishes downloading the current broadcast of A , it has at least $|X|$ time units of data remaining in its buffer. The same amount of data remains the client finishes downloading B and C since they are received sequentially. Thus, the next broadcast of D must start in the next $|X|$ time unit to ensure playback continuity for this client. Since the current broadcast of D starts Y time units after that of B starts, the size of $|D|$ cannot be larger than $(|B| - X + Y) + |C| + X = |B| + |C| + Y$. We apply the same analysis on the case when $|X| < |Y|$, showed in Figure 2(e), and can find that the maximum size of $|D|$ is also $|B| + |C| + Y$. Note that in either cases, X is irrelevant to determining the size of D . Since the smallest possible value for interval Y is $HCF(|A|, |D|)$, the size of D must not be larger than $|B| + |C| + HCF(|A|, |D|)$.
- *Alignment 5:* In this case, both B and D start ahead of A . D can start either before, at the same time as, or after B . We first consider the case when D starts before or the same time as B , as illustrated in Figure 2(f). As the segments are downloaded group by group, a client downloading A cannot download D in its current broadcast. Obviously, after the client starts to download A , it will take $|A| + |B| + |C|$ time units to consume the downloaded data before it needs to access a new broadcast of D . Thus, the size of D can be as large as $|A| + |B| + |C|$. The same conclusion holds

for the case when D starts after B but before A .

The alignments discussed above include all possible cases that make the size of D smallest. Our analysis reveals that the size of D is limited by $|A| + |B| + |C|$ and $|B| + |C| + HCF(A, D)$. To find the maximum size of D , we try in the range $|A| + |B| + |C|$ to $|B| + |C|$ and choose the largest one as $|D|$ that satisfies the condition $|B| + |C| + HCF(|A|, |D|) \leq |A| + |B| + |C|$. Note that there are always some values in the range that satisfy the condition - we can always use some multiple of $|A|$ as $|D|$. Once we fix the sizes of C and D , we can use them to fix the size of segments in the next group.

The above guideline allows us to find the following broadcast series for $c = 2$:

$$[1, 2, 2, 5, 5, 12, 12, 25, 25, 60, 60, 125, 125, 300, 300, \dots]$$

It is worth mentioning the above series grows faster than that from Skyscraper Broadcasting [5], which was designed specifically for $C = 2$ and until now is the fastest one in this setting:

$$[1, 2, 2, 5, 5, 12, 12, 25, 25, 52, 52, 105, 105, 212, 212, \dots]$$

4 Proposed Technique

For the above motivation example, we develop a new generalized broadcast technique. We will call this scheme as $CCA+$ since it can be regarded as an enhanced version of CCA . Given a broadcast bandwidth of k channels, $CCA+$ also partitions the video into k segments, S_1, S_2, \dots, S_k . However, the sizes of these segments are determined using the following formula, where c is the number of channels accessible at receiving ends:

$$|S_i| = \begin{cases} 2^i - 1, & \text{if } 1 \leq i \leq c, \\ |S_{i-1}|, & \text{if } (i+1) \bmod c = 0 \text{ and } i > c, \\ x & \text{where } x \text{ is the largest number such} \\ & \text{that } x \leq \sum_{j=i-c-1}^{i-1} |S_j| \text{ and} \\ & HCF(x, |S_{i-c-1}|) = |S_{i-c-1}|, \text{ otherwise.} \end{cases}$$

Note that the broadcast series for the first $c + 1$ segments is the same as in the original CCA . Thus, a client can download and playback the first $c + 1$ segments continuously. To prove our new scheme does work, we just need to demonstrate that before a client finishes downloading and consuming $c + 1$ segments it can always start to download the next segment.

Let $A_1, A_2, \dots, A_c, A_{c+1}$, and A_{c+2} be $c + 2$ consecutive segments and the playback continuity of the first $c + 1$ segments is known to be guaranteed. Apparently, if $|A_{c+2}| = |A_{c+1}|$, the playback continuity holds. In this case, their broadcast starts and finishes at the same time. After a client finishes downloading A_{c+1} , it can always continue to download a new

broadcast of A_{c+2} . In the next, we prove playback continuity when $|A_{c+1}| \neq |A_{c+2}|$, in which case $|A_{c+2}|$ is the largest value in between 0 and $\sum_{i=1}^{c+1} |A_i|$ that satisfies equation $HCF(|A_{c+2}|, |A_1|) = |A_1|$.

Since $HCF(|A_{c+2}|, |A_1|) = |A_1|$, $|A_{c+2}|$ must be a multiple of $|A_1|$. Therefore, there are only three possible broadcast alignment cases for these two segments, as showed in Figure 3. We analyze them as follows:

- Alignment 1: A_1 and A_{c+2} start at the same time. Because the client can access only c channels simultaneously, it cannot download the current broadcast occurrence of A_{c+2} . From the time when the client starts to download A_1 , it will take $\sum_{i=1}^{c+1} |A_i|$ to playback all $c + 1$ segments. Thus, the next broadcast of $|A_{c+2}|$ must occur within $\sum_{i=1}^{c+1} |A_i|$ time units. Thus, if the size of $|A_{c+2}|$ is larger than $\sum_{i=1}^{c+1} |A_i|$, client playback continuity could be broken. Obviously, this is avoided by our segmentation formula with the condition $|A_{c+2}| \leq \sum_{i=1}^{c+1} |A_i|$.
- Alignment 2: A broadcast of $|A_{c+2}|$ starts right after a broadcast of A_1 finishes. In this case, after a client finishes downloading A_1 , it can simply use the same channel to download A_{c+2} . Thus, playback continuity will not be affected by any size of A_{c+2} .
- Alignment 3: A broadcast of A_1 starts and finishes in between a broadcast occurrence of A_{c+2} . Given $|A_{c+2}|$ is a multiple of $|A_1|$ and no larger than $\sum_{i=1}^{c+1} |A_i|$, it is trivial that in this case, the current broadcast $|A_{c+2}|$ must be finished in the next $\sum_{i=2}^{c+1} |A_i|$ time units. Thus, before the first $c + 1$ segments are downloaded and consumed completely, a new broadcast of $|A_{c+2}|$ must have already started. However, to prove playback continuity, we must show that the client is able to download these $c + 2$ segments using c channels. According to our segmentation formula, every c forms a group and the size of the last segment in one group is the same as that of the first segment in the next group. Given a number of $c + 1$ segments, they must span in two groups and at least two of them have the same size. Assume $|A_i| = |A_{i+1}|$, where $2 \leq i \leq c$ (Note that i cannot be 1; Otherwise A_1 will be the last segment in one group and A_{c+1} and A_{c+2} will be in two different group, making $|A_{c+1}| = |A_{c+2}|$). At the time a client finishes downloading A_i , it must have finishes downloading all previous segments since they are in the same group and A_i is the

last one in that group. This means that at that time point, the client has the capability to access c channels simultaneously. In other words, each of the remaining segments, A_{i+1}, \dots, A_{c+2} , can be downloaded using a distinct receiving channel.

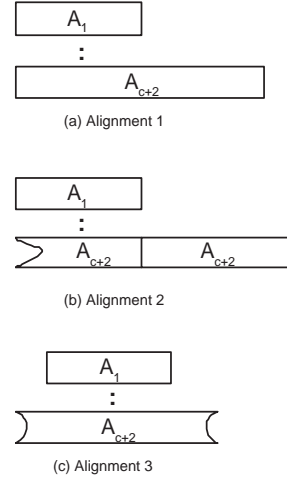


Figure 3: Three possible alignments of A_1 and A_{c+2}

We now explain how to generate a broadcast series using our segmentation formula. To determine the size of A_{c+1} from A_1, A_2, \dots, A_{c+1} , we can try the numbers one by one ranging from 0 to $\sum_{i=1}^{c+1} |A_i|$. We can start by selecting $A_{c+2} = \sum_{i=1}^{c+1} |A_i|$ first, and then proceed down. Once the condition stated in the formula is satisfied, the number is then used as $|A_{c+2}|$. After fixing $|A_{c+2}|$, we can determine the size of the next segment and so forth.

Because the broadcast series from our new scheme grows faster than that from CCA, the actual size of the first segment is smaller, making broadcast latency shorter. We note that while a faster broadcast series reduces the broadcast latency, it may require more disk buffer at client site. Fortunately, disk buffer is no longer a major concern, considering that one can hardly find a hard drive less than 10GB in today's storage market. Ultimately, it is worth making every effort to better utilize client receiving bandwidth and server broadcast bandwidth, both of which are expensive and neither one can be upgraded frequently.

5 Performance Study

We analyze the performance of the proposed technique in this section by comparing its performance to that of CCA, which until now is the only technique that can leverage broadband connection for better broadcast performance. Since we are primarily in-

interested in the relative performance of the two techniques, we assume in our study that the system has only one video. As we have discussed previously, if the system has n videos, the server bandwidth can be thought as divided evenly among n virtual servers. Each server is used to serve one of the n videos. Thus, the results reported in this section are also valid for systems with many videos.

We assume the video is assumed to be encoded using MPEG-1 with the average playback rate of 1.5 Mbits/sec. We are interested in the server bandwidth ranging from 12.0 Mbits/sec to 24.0 Mbits/sec. On the high end, we stop at 24.0 Mbit/sec since this is large enough to show the trends of the various design schemes. As for the client bandwidth, we choose the range from 3 Mbits/sec to 9 Mbits/sec because less than 3 Mbits/sec makes both techniques the same performance as stagger broadcasting, and 12 Mbits/sec is more than adequate for CCA+ to show its outstanding performance. It is not very interesting to make the access latency any smaller.

We choose *worst service latency* as our performance metric and will focus on how this is affected by client receiving bandwidth and server broadcast bandwidth. The formula for calculating the service latency under both techniques is given by $\frac{|V|}{\sum_{i=1}^k |S_i|}$. We present the performance results in the following subsections.

5.1 Effect of Client Receiving Bandwidth

In this subsection, we analyze the effect of client bandwidth on the service latency of the two broadcast techniques. We vary the client receiving bandwidth from 3.0 Mbits/sec to 9.0 Mbits/sec while the server broadcast bandwidth is fixed at 12.0 Mbits/sec. The access-latency curves for CCA and CCA+ under these conditions are plotted in Figure 4. We see that the access latency under both schemes decreases when client receiving bandwidth increases. In all cases, however, CCA+ outperforms CCA about 50%. For instance, when the client download bandwidth is 3 channels, the worst access latency guaranteed by CCA is more than 15 seconds while that under CCA+ is less than 10 seconds. We can also observe that the performance gain is more significant when the ratio of client bandwidth and broadcast bandwidth is smaller. As the client bandwidth approaches closer to the server bandwidth the performance gain by CCA+ decreases. This is due to the fact that both schemes have the same broadcast series for its first group of segments. When client bandwidth is equal to broadcast bandwidth, the two schemes essentially are the same. In reality, however, server bandwidth in general should be much higher than receiving bandwidth.

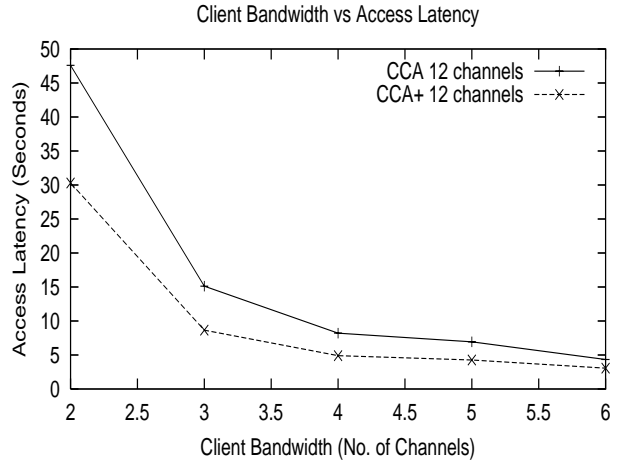


Figure 4: Access latency vs. client receiving bandwidth.

5.2 Effect of Server Broadcast Bandwidth

In this study, we investigate the effect of server broadcast bandwidth on the two broadcast schemes. We vary the server bandwidth from 8 to 16 channels and see the access latency is improved when the client receiving bandwidth is 2, 3, and 4 channels. The results of our study are plotted in Figures 5, Figures 6, Figures 7, respectively. We see that in all scenarios that CCA+ gives a significantly better performance in comparison to CCA. We also notice that the performance of both schemes is all improved significantly with the increasing of broadcast bandwidth. This is a desirable feature since adding more broadcast bandwidth is much easier than improving the bandwidth of all “last-mile” connections.

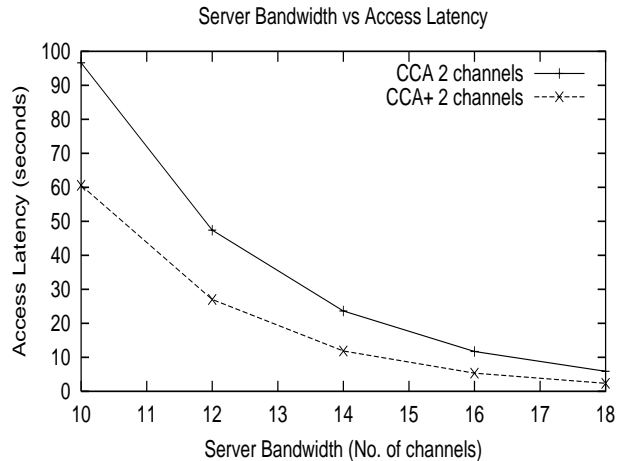


Figure 5: Access latency when $C=2$.

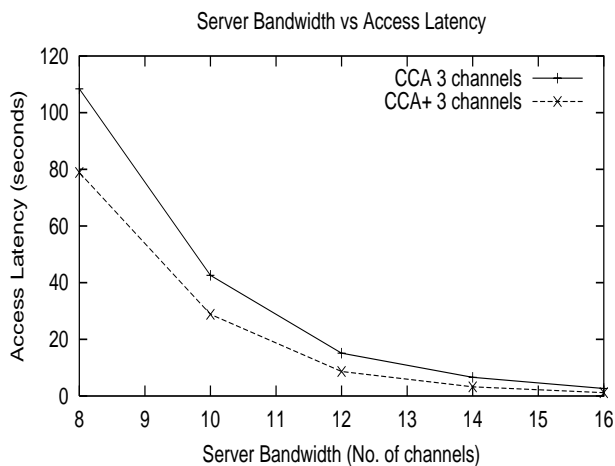


Figure 6: Access latency when $C=3$.

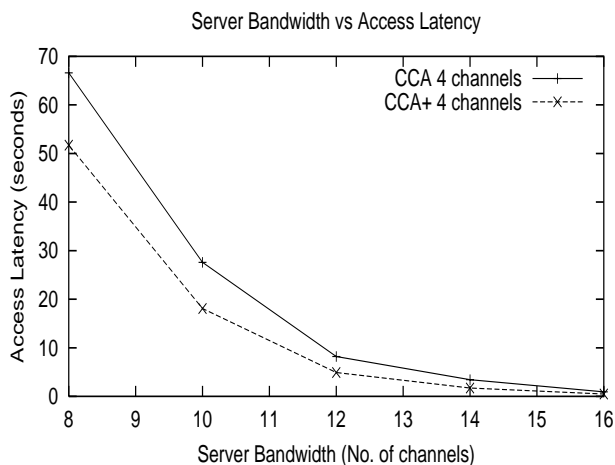


Figure 7: Access latency when $C=4$.

6 Concluding Remarks

We have presented in this paper a novel broadcast technique that can effectively utilize broadband access to minimize server broadcast latency. Unlike most existing schemes, the new approach can significantly improve the broadcast efficiency with the increase of client receiving bandwidth. This feature is highly desirable because more and more people now have broadband access. We analytically proved the correctness of our technique by showing that the continuity of the client playback is guaranteed. To substantiate its good performance, we provided analyses to compare its service latency with that of our previous CCA scheme, which was the only existing technique that can leverage client bandwidth for more efficient broadcast. Our

performance results convincingly show that the proposed technique is substantial better under the same hardware conditions.

It is worth pointing out that the new scheme works under the assumption that all clients have the same receiving bandwidth. Client heterogeneity, however, is an inherent part of today's networks. Therefore, an important future work is to extend the proposed scheme to work for heterogeneous clients.

References

- [1] 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.
- [2] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic Batching Policies for an On-Demand Video Server. *Multimedia Systems*, 4(3):112–121, June 1996.
- [3] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting. *Multimedia systems*, 4(4):179–208, August 1996.
- [4] 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.
- [5] K. A. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-On-Demand Systems. In *Proc. of the ACM SIGCOMM'97*, Cannes, France, September 1997.
- [6] J. F. Paris, S. W. Carter, and D. D. E. Long. Efficient Broadcasting Protocols for Video on Demand. In *Proc. of SPIE's Conf. on Multimedia Computing and Networking (MMCN'99)*, pages 317–326, San Jose, CA, USA, January 1999.
- [7] J.F. Paris and D.D.E. Long. Limiting the Receiving Bandwidth of Broadcasting Protocols for Videos on Demand. In *Proc. of the Euromedia 2000 Conference*, pages 107–111, May 2000.
- [8] J.F. Paris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. In *Proc. of Int'l Conference on Computer Communication and Networking*, pages 418–423, October 2001.
- [9] K. A. Hua, Y. Cai, and S. Sheu. Exploiting Client Bandwidth for More Efficient Video Broadcast. In *Proc. of Int'l Conference on Computer Communication and Networking*, pages 848–856, Louisiana, U.S.A, October 1998.
- [10] Kien A. Hua, Ying Cai, and Simon Sheu. Leverage client bandwidth to improve service latency of distributed multimedia applications. *Journal of Applied Systems Studies (JASS)*, 2(3):686–704, 2001.