Broadcasting Issues in Video Servers

Murali Mani and Richard Muntz
Department of Computer Science
University of California, Los Angel es
{mani,muntz}@cs.ucla.edu

February 9, 1999

Abstract

Periodic broadcasting for supporting near video on demand has received significant attention recently, and various schemes have been proposed. We examine schemes in which the video is partitioned into several fragments of varying sizes and each fragment is broadcast repeatedly in a logical channel allotted to it. For such schemes, we suggest partition functions that will support heterogeneous clients, and examine how the client can determine its waiting time depending on the resources that the client is ready to commit to the video. We then examine the issues in broadcasting VBR videos; previous work has all dealt with CBR videos. We suggest schemes for VBR broadcast, and evaluate the performance of these schemes for sample VBR videos.

1 Introduction

Video is a very popular means of media communication and entertainment. In addition to the popularity of video [2], video has the property that most of the demand is for a small number of videos [3, 4]. Such videos are referred to as "hot" videos in [5, 7]. The issues involved in broadcasting video are different from the issues in broadcasting traditional media such as text and image because of the size of the video and the additional constraint that video must be delivered continuously and on time. The main consideration for the server is to minimize the bandwidth required for broadcasting a video. (If all the channels have the same bandwidth, then the server tries to reduce the number of channels in which the video is broadcast.) From the client's point of view the metrics are (a) the client's storage or disk capacity (b) the bandwidth, that is the number of channels that the client will have to receive data from concurrently and (c) the latency, that is the time that the client has to wait till playout of the video can start. So the objective is to find an efficient way of transmitting the video that is economical for both the client and the server. In this paper we are concerned with the analysis that quantifies these tradeoffs. The "optimal solution" for a given set of clients will require additional considerations such as the distribution of clients with various resource levels which is beyond the scope of this paper.

A common approach for broadcasting videos is to partition each video into fragments, allocate a channel for each fragment of the video, and repeatedly transmit each fragment in the channel allotted to it. One method is to partition the video into equal size fragments and transmit these fragments repeatedly [4]. But an obvious drawback with this approach is that the maximum latency experienced by the client is inversely proportional to the number of channels in which the video is broadcast (for a given length of video), so to obtain a small maximum latency, the server will have to broadcast the video in a large number of channels. For example, in order to ensure a maximum latency of five minutes for a two hour movie, the server will have to transmit the movie in twenty four channels, and to ensure a maximum latency of two minutes, the server will have to transmit the movie in sixty channels.

For a given number of channels the maximum latency can be decreased by exploiting the client's disk capacity and bandwidth. Various approaches have been suggested for this. Viswanathan and Imielinski suggested a *pyramid broadcasting* scheme in [11, 12]. The main idea behind pyramid broadcasting is that the video is partitioned into n fragments whose sizes grow geometrically, and the frequencies of transmission of these fragments depend on their sizes. Later on, Aggarwal, Wolf and Yu suggested a permutation based pyramid broadcasting scheme [1], which required less client storage than the previous scheme.

Hua and Sheu suggested a skyscraper broadcasting method in [7]. Here the video is partitioned into a sequence of disjoint fragments of increasing size and all the fragments are broadcast at the same bandwidth. They suggested a partition function that will require that the client receive data from a maximum of only two channels at any instant. This partition function was improved by Gao, Kurose and Towsley [6]. Eager and Vernon proposed a dynamic skyscraper broadcasting scheme in [5]. In dynamic broadcasting, it is assumed that there is insufficient bandwidth to statically broadcast all the videos. Here the idea of *transmission clusters* is used, which is an extension of the static broadcasting techniques.

In [8, 9] Juhn and Tseng suggested a harmonic broadcasting strategy. This uses equal size fragments but exploits the fact that the latter fragments can be transmitted slowly. But this scheme requires a high server bandwidth to ensure a small maximum latency for the client.

In this paper the parameters that we are concerned with are the server bandwidth, the client bandwidth, the client storage, and the client latency. We first briefly review the work that has been done for determining these parameters for a CBR video. Not all clients will be identical, but none of the previous work dealt with heterogeneous clients. A good broadcast scheme should support heterogeneous clients and ensure a low maximum latency for any client. We derive the class of partition functions which satisfies the above conditions and also ensures that a sophisticated (or "fat") client experiences less latency than a simple (or "thin") client and the resource requirements for a sophisticated client will be met by a large number of clients. We also examine how a client can determine its waiting time depending on the resources that the client is willing to commit to the video.

We then extend the observations on the broadcast of CBR videos to the broadcast of VBR videos. Here we examine the various issues in broadcasting a VBR video. The burstiness in the playout of a VBR video requires variations to the solutions for CBR video. We propose *Constant SubFragment* broadcast scheme for broadcasting VBR videos, which is an extension of the CBR schemes. This scheme requires less capabilities from the client than other schemes and maintains low client latency, low network bandwidth fluctuations and low maximum network bandwidth. Finally we evaluate the various schemes for broadcasting VBR videos using sample videos and observe that the constant subfragment broadcast scheme performs better than the other schemes for the above mentioned parameters.

2 Notations and Previous Work

Let the server broadcast[1] a video in n channels according to the partition function, $F = f_1, f_2, \ldots, f_n$, where the term f_i corresponds to the ith fragment. Each fragment i will be repeatedly broadcast by the server in the channel allotted to it. We shall examine the broadcast of one video. Broadcast of more than one video follows directly. We make the following assumptions.

- The video is assumed to be a CBR encoded video unless otherwise stated. The broadcast of a VBR video is specifically analyzed in a later section.
- The server broadcast rate is the same as the client playback rate. This is a *necessary* condition if the "simplest client", that is the client which can receive from the network only one fragment at a time (or with no disk storage and no disk bandwidth), has to be supported.
- There are enough channels for the server to broadcast all the videos that it wants to broadcast. In other words, in this paper we are not concerned with the allocation of server bandwidth among the entire set of videos. Dynamic broadcast of CBR videos has been examined in [5], and it was shown that partitioning the video into fragments as for static broadcast of CBR videos can be used for dynamic broadcast of CBR videos.
- The client is a *disk conserving system*, that is the client tries to minimize the amount of video stored in its disk at any instant. This means that the client will receive data from a channel only when it is necessary, even if the client has the necessary bandwidth to listen to more channels [6].
- There is a smallest time unit with which the server broadcasts the video fragments. All times should be considered normalized, such that this smallest time unit is equal to one.[2]

The variables that we use in this paper and their definitions are given in the table below. We

[1] The use of the term broadcast rather than multicast was more a matter of personal preference. We need not distinguish between broadcast and multicast for our purpose.

[2] Note that the duration of each fragment as well as the start time of the broadcast of each fragment will be given by an integer number of time units.

Variable	Definition
n	The number of channels in which the video is broadcast.
$f_1, f_2, \ldots, f_n$	f_i is the duration of the ith fragment.
d	Duration of one time unit (in seconds)
s	Size of the video corresponding to one time unit (in Bytes)
C	The client disk capacity (in Bytes)
d_1	$f_1 \times d$ (This is the duration of the first fragment in seconds.)

will now place some constraints on the partition function. The *earliest instant* that a client with sufficient resources can start playing back the video is the next time the start of the first fragment is received. The objective of the server will be to minimize the number of channels over which the server broadcasts a video while maintaining the client latency at a low value. The fastest growing partition function will minimize the number of channels in which a video is broadcast [6]. But the growth of the partition function is limited by the "continuous effect" which the client must experience. A client is said to experience continuous effect if the client can receive each part of the video prior to its playout time. We examine partition functions where the client with sufficient resources can start playout at the earliest instant (i.e., the maximum latency for a "sophisticated" or a "resource rich" client is f_1) and experience continuous effect.

We shall examine the *synchronous broadcast* of videos. We say that the broadcasts of two fragments i and j are synchronized if there exists a time instant when both the fragments start being broadcast simultaneously in their respective channels. We call the broadcast of a video a *synchronous broadcast* if there exists a time unit when the start of all the fragments of the video will be broadcast at the same time in their respective channels. In synchronous broadcast, if the fragments i and j of a video satisfy the condition $gcd(f_i, f_j) = k$, then the start times of broadcast of these fragments always differ by a multiple of k.

Observation 1 *[6] In synchronous broadcast, the fastest growing partition function such that a client with sufficient resources can start playout at the earliest instant is given by* $F = \{f_1, f_2, \ldots, f_n\}$, *where the first term is given by* $f_1 = c$, *for an arbitrary* c, *and the remaining terms are:*
$f_i = 2 \times f_{i-1}$, $2 \leq i \leq n$

Observation 2 *[6] In synchronous broadcast, the fastest growing partition function that requires a maximum client bandwidth of* i *and the first term is* 1 *time unit, such that the client with sufficient resources can start playout at the earliest instant is given by*

$$S = 1, 2, 4, \ldots, 2^{i-1}, 2^i - 1, 2^{i+1} - 4, \ldots$$

The j*th term* f_j *where* $j > i$ *is given by*

$$f_j = s_j + f_{j-i} - (s_j \ \% \ f_{j-i})$$

where $s_j = \sum_{k=j-i+1}^{j-1} f_k$. *This expression means that* f_j *is given by the smallest number greater than* s *which is a multiple of* f_{j-i}. *This is possible when the broadcasts in the channels* j *and* $j - i$ *are synchronized.*

The synchronization between the broadcasts can be exploited to give a partition function that will require that the client listen to a maximum of only two channels at any instant, and where the client with sufficient resources can start playout at the earliest instant.

Observation 3 *[6] A partition function that requires that the client listen to a maximum of only two channels at any instant is given by*

$$1, 2, 2, 5, 5, 12, 12, 25, 25, 60, 60, \ldots$$

This partition function requires that a 120 minute video be broadcast in 20 channels, and the client has a storage of 37% of the whole video, if the duration of the first fragment is one second.[3] Another interesting partition function is the series of repeating powers of two.

$$1, 2, 2, 4, 4, 8, 8, 16, 16 \ldots$$

Here the term f_i is given by $2^{\lfloor i/2 \rfloor}$. This is the slowest growing partition function among the partition functions that we have seen till now. It requires 24 channels for a 120 minute video (if the duration of the first fragment is one second). But the storage required is only 25% of the whole video. As the fragments $2i$ and $2i + 1$ are synchronized, the client never has to read the $(2i + 1)$th stream while reading the $(2i)$th stream. Therefore the client will have to receive data from a maximum of two channels at any instant. This partition function has several other favorable characteristics as will be seen in the next section.

3 Latency observed by the client

In all the partition functions we have seen so far, the maximum client latency was equal to the duration of the first fragment, provided the client had the required storage and bandwidth. Most of the partition functions required that the client listen to a maximum of only two channels at any instant. But the storage required was high for most of the partition functions. The series which demanded the least storage from the client required that the client has a storage of at least 25% of the whole video. In this section we assume that not all clients will be able to meet the above requirement. We examine the latency experienced by diverse clients and derive partition functions that give a small latency for diverse clients, including the simplest client (the client with no disk storage or disk bandwidth). The broadcast scheme used by the server should not depend on the type of clients, the client instead will determine its waiting time depending on the resources it is willing to commit to the video.

We want to partition a video such that any client (with resources in some range) which requests the video will experience a small latency. The fastest growing partition functions are not necessarily

[3] If the client buffers 1 minute of each popular video, then the server can broadcast the remaining video using a partition function where the first term is 1 minute. In such a case, the server will need only 10 channels for a video and the client can start playout immediately.

the best for supporting diverse clients. We give partition functions that ensure that the maximum latency experienced by the simplest client is small, a sophisticated client experiences a much smaller latency than the simplest client, and the resource requirements can be tailored to accommodate a large number of clients. We do not assume any distribution of the various types of clients, knowledge of this distribution might be required for obtaining the "optimal solution".[4] Instead we concentrate on the parameters we have discussed and derive some properties that are required of the partition functions.

One way of supporting diverse clients is for the server to broadcast a video using two different partition functions. One partition function guarantees a latency of c time units for the sophisticated clients which have the required bandwidth and storage. The second partition function will divide the video into equal sized fragments which are broadcast in synchronization. If a client listens to this partition function, then it will experience a worst case latency of the duration of a fragment. Let the server broadcast a 120 minute video using two partition functions, the first one which guarantees a maximum latency of 8 seconds for the sophisticated client, and the second one which guarantees a maximum latency of 8 minutes for the simplest client. Here the server requires at least 10 channels for the first partition function and at least 15 channels for the second partition function. Therefore this scheme requires that the server broadcast a video in at least 25 channels. This scheme also requires that a sophisticated client has the bandwidth to listen to 10 channels simultaneously and it has a disk capacity of almost 50% of the video. But even if a client has no storage and no spare disk bandwidth, it can view the video continuously by listening to the second partition function.

There is another approach to solve this problem which does not require that the server broadcast the same video using two different partition functions. Also, the requirements for the sophisticated client to experience a maximum latency equal to the duration of the first fragment will be less than in the previous approach. This is based on the following observation.

Observation 4 *If the server broadcasts a video using a particular partition function,* $F = \{f_1, f_2, \ldots, f_n\}$*, and there exists a set of starting points for the broadcasts of the fragments that can support even the client with no storage,*[5] *then the simplest client will experience a maximum latency* L*, given by* $L = LCM(f_1, f_2, \ldots, f_n)$.

To have a low maximum latency for the simplest client, the partition function should be such that $\mathrm{LCM}(f_1, f_2, \ldots, f_n)$ is minimum. The easiest way of achieving this is when each term in the partition function is a multiple of the previous term. This would ensure that the maximum latency experienced by the simplest client is equal to the duration of the largest fragment.

[4]For example, if all the clients which request for a video have no storage or disk bandwidth, then partitioning the video into equal sized fragments will be the "optimal solution".

[5]This might require that the server broadcast the fragments such that there does not exist a time instant when all the fragments start being broadcast simultaneously in their respective channels (i.e., unlike in [6], the broadcast schedule is not synchronous). For example, if the partition function is $2, 3, 4$, then for the simplest client to experience continuous effect, the server should start broadcasting the third fragment one time unit after it starts broadcasting the first fragment.

Observation 5 *The fastest growing partition function where each term is a multiple of the previous term and the client has to receive data from a maximum of only two channels is given by*

$$1, 2, 2, 4, 4, 8, 8, \ldots$$

This is the series formed by the repeating powers of two.

Such a partition function can be used to circumvent the problem of the server having to broadcast a video using two different partition functions. Let the duration of the first term be 8 seconds. The largest term in the partition function can be adjusted so as to minimize the maximum latency for the simplest client. If the largest term is 512 seconds, then the simplest client would experience a maximum latency of only around eight minutes. For this partition function the storage required to guarantee a latency of 8 seconds is only 7% of the whole video. Also for a client to experience a maximum latency of 8 seconds, it will need to have the network bandwidth to listen to only two channels at any instant. So many more clients will be able to satisfy the resource requirements for experiencing a maximum delay of only 8 seconds. The number of channels required for the server to broadcast a 120 minute video using this partition function is 24.

By adjusting the largest term of the partition function, we obtain different latencies for the simplest client, different storage requirements to experience a maximum latency of f_1 and different server bandwidth requirements. The table below shows the trade-off between these parameters for different values of the largest term. The video is assumed to be a 120 minute movie and the partition function used is $1, 2, 2, 4, 4, 8, 8 \ldots$, where the first term is 8 seconds. This partition function requires that no client has to listen to more than two channels simultaneously.

Largest Term (secs)	Number of channels required by server	Storage required for sophisticated client (% of the whole video)	Maximum latency for simplest client (mins)
2048	17	25.1	34.1
1024	19	12.5	17.1
512	**24**	**6.7**	**8.5**
256	36	3.5	4.3
128	62	2.0	2.1

As we increase the client bandwidth requirement for the sophisticated client, we obtain faster growing partition functions. For example, if the client can receive data from three channels simultaneously and each term is a multiple of the previous term, then the fastest growing partition function is given by $1, 2, 4, 4, 8, 16, 16, 32, 64, 64, \ldots$. If we fix the duration of the smallest fragment as 8 seconds and the duration of the largest fragment as 512 seconds, then the server requires only 21 channels to transmit the whole video.

The following table shows the trade-off between the server bandwidth required and the client bandwidth required to guarantee a maximum latency of f_1. The video is assumed to be a 120 minute movie. The first entry corresponds to dividing the entire video into equal sized fragments, each of

8 seconds duration. The duration of the smallest term in all the remaining partition functions is 8 seconds and the duration of the largest term is 512 seconds. The partition functions are assumed to satisfy the condition in Observation 4 (each term is a multiple of the previous term), so that the simplest client will have a maximum latency of around 8 minutes.[6]

Bandwidth required for the sophisticated client (number of channels)	Server bandwidth required (number of channels for broadcast)
1	900
2	**24**
3	21
4	20
5	20

4 Determining the minimum latency

In this section we will examine how the client will determine the amount of time it should wait so that it can experience continuous effect with its available storage. We are not worried about the bandwidth because we assume that the server is broadcasting at a rate that requires that the client listen to a maximum of only two channels at any instant.[7] The latency that the client experiences is not always smaller for a more sophisticated client; a simple client could be lucky and arrive at an instant when he will not have to wait for even one time unit, but the maximum latency is less for a more sophisticated client.

Let the client have S time units of disk capacity, i.e., $S = \lfloor \frac{C}{s} \rfloor$. Let the video be broadcast according to the partition function $f_1, f_2, \ldots, f_n$ and t_r be the arrival time of the client request at the server. The server informs the client of the durations of the fragments and the nearest time that each of these fragments will be broadcast. Let the nearest times be $t_1, t_2, \ldots, t_n$ such that $t_i \geq t_r$ for $1 \leq i \leq n$. In other words, $t_r \leq t_i < t_r + f_i$. Given this information, the client will determine the minimum amount of time it has to wait before it starts listening to the broadcast. Note that t_1 gives the earliest time that a client can possibly start play back, given that the request is made at t_r. The client that satisfies the maximum storage requirement will experience a maximum delay of only f_1 time units.

The calculation of the waiting time by the client as a function of its available storage is based on the following theorem. Let S_{ii} denote the storage required for fragment i at the start of broadcast of the ith fragment. We define the *maximum storage* required for any individual fragment as $S = \max\{S_{ii} \mid 1 \leq i \leq n\}$.

[6] One can see that the choice of 24 channels is a very good one, but the optimal partition function will depend on the type of clients which we wish to satisfy.

[7] A similar approach can be used by the client to determine its waiting time depending on the bandwidth available to the client.

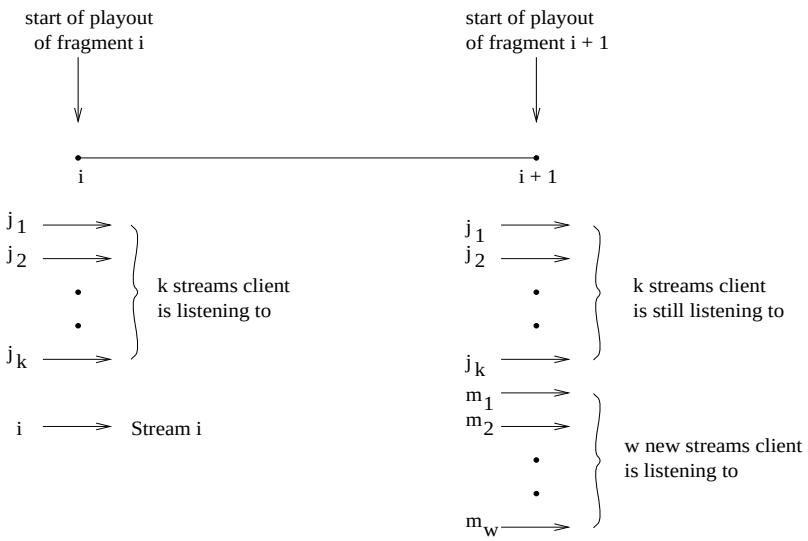

Figure 1: The listening pattern for a disk conserving client

Theorem 1 *Let S be the maximum storage (in time units of the video) required for any individual fragment given the playback start time selected by the client. Then the maximum storage required for the client to experience continuous effect for that latency is S. In other words, the maximum storage required at any instant of time (which can include pieces of different fragments) is equal to the maximum storage required for any individual fragment.*

Proof. Let the partition function be $f_1, f_2, \ldots, f_n$. Let S_{ij} denote the storage (in time units) required for fragment j just before the start of playout of fragment i. As the client is a disk conserving system the client will receive a fragment only when it is necessary for the client to listen to a fragment (this means that a client will listen to a fragment only when the next broadcast of the fragment begins after the client has to start playout of that fragment). Also once the client has started receiving the start of a fragment, it will receive the fragment continuously till the end of the fragment. Therefore if a disk conserving client listens to a fragment j just before the start of playout of the ith fragment ($j \neq i$), then the client will be listening to the jth fragment even at the end of playout of the ith fragment and the start of playout of the $(i+1)$th fragment. See Figure 1. Therefore the maximum storage required by the client will be at the start of playout of one of the fragments.

As the maximum storage required for fragment j occurs at the start of its playout and satisfies $S_{jj} \leq S$, we obtain $S_{ij} \leq \max\{0, S - \sum_{k=i}^{j-1} f_k \mid j > i\}$. The maximum storage required for fragment i before the start of its playout, S_{ii} is given by $S_{ii} = \min\{S, f_i - 1\}$. Then the storage required for all fragments at the start of playout of fragment i is given by $S_{ii} + \sum_{j=i+1}^{n} S_{ij}$.

It can be seen that this term is less than or equal to S for all i. The main idea is as follows. Let the last fragment which requires some portion of it to be stored before the start of playout of the ith fragment be f_k. No storage is required for the fragments after f_k. Let the storage required for the fragments from i to $k-1$ be S_{prev}. The client will playout S_{prev} before it starts playout of f_k.

Therefore the storage required for f_k before the start of playout of f_i is given by $S_{ik} \leq S - S_{prev}$ (this is because the maximum storage required for f_k before the start of its playout is S). Therefore the maximum storage required for the client to experience continuous effect is S. □

A client with S time units of storage can therefore determine its waiting time by determining the latency which will require that the client devote no more than S time units of storage for each fragment j just before the start of playout of that fragment. Note that if $f_i \leq S + 1$ for $1 \leq i \leq n$, then this immediately implies that the client has the required storage and it can start playback at the earliest instant irrespective of the t_i's. Let there be k terms in the partition function given by $h_1, h_2, \ldots, h_k$ such that $h_i > S + 1$, for $1 \leq i \leq k$. Let the index of these terms in the partition function be given by $m_1, m_2, \ldots, m_k$, in other words $h_i = f_{m_i}$. For each h_i the client calculates the storage required for each latency from t_1 to T where T is the maximum permissible latency.[8] We denote the storage required for h_i for a particular latency t as $s_i(t)$.

Observation 6 *For a particular latency time t, the storage required for h_i just before the start of playout of h_i denoted by $s_i(t)$ is given by $s_i(t) = (t + \sum_{l=1}^{m_i-1} f_l - t_{m_i}) \ \% \ (h_i)$*

Proof. Let the earliest instant that the client can start receiving h_i be t_{m_i}. Let the client listen to the broadcast of h_i which occurs $t_{m_i} + k \times h_i$ time units after t_r, the client request arrival time (here k is an integer). The storage required for h_i just before the start of playout of h_i when the client waits for t time units is given by $t + \sum_{l=1}^{m_i-1} f_l - t_{m_i} - kh_i$. Therefore we obtain $s_i(t) = (t + \sum_{l=1}^{m_i-1} f_l - t_{m_i}) \ \% \ (h_i)$ □

Using this expression the client determines all possible startup latencies from t_1 to T for each h_i, such that the storage requirement for h_i, given that latency, is not more than S. The smallest latency which satisfies the storage requirements for all h_i's will be chosen by the client.

We shall now consider an example of latency determination. Let a video be partitioned into five fragments and the durations of the fragments be given by $F = \{1, 2, 4, 7, 12\}$. Let a client request the video at time 0 and the values of the t_i's be $\{0, 0, 0, 0, 0\}$. Let the client have a storage of 3 time units of the video, that is $S = 3$. There are two fragments that might require greater than 3 time units of storage, they are the fragments with durations 7 and 12. Therefore the client calculates the storage required for these two fragments at various latencies. A portion of the storage requirements corresponding to latency values from 0 to 7 time units is shown in Table 1. The client determines that a storage of 3 time units is enough for it to start playback immediately, that is at time zero.

The partition function $1, 1, 2, 2, 2, 4, 4, 8, 8, 16, 16 \ldots$ has the property that the simplest client can start playout of the video when the start of all the fragments are broadcast simultaneously in their channels (at the synchronization point for all fragments). Figure 2 shows the distribution of average latencies for diverse clients for this partition function where the length of the first term is 8 seconds and the length of the largest term is 512 seconds. Here we assume that the number of

[8] The choice of T depends on the maximum time which the client is willing to wait. If unspecified, it will be the maximum latency for the simplest client.

Latency	Fragment 7	Fragment 12
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6
5	5	7
6	6	8
7	0	9

Table 1: Latency-Storage Table

clients with different levels of storage is a Poisson distribution with a mean storage of 500 MB, and the actual arrival times of each class of clients is uniform over the entire time frame. The length of the CBR video is assumed to be 120 minutes with a playout rate of 10 Mbps.

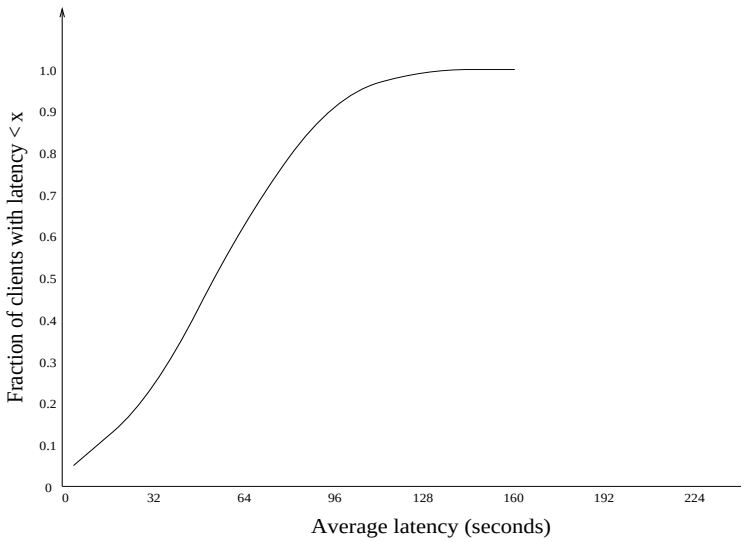

Figure 2: The distribution of the average latencies for diverse clients

5 Broadcasting VBR video

We have examined the static broadcast of CBR videos and the various parameters for the static broadcast of CBR videos. In this section we examine the static broadcast of VBR videos. What makes a VBR video different from a CBR video is the inherent burstiness in the playout of a VBR video. We examine two schemes for broadcasting VBR videos and the various trade-offs in these schemes. The additional issues in broadcasting VBR videos are the fluctuations in the network bandwidth required by the server and the minimum storage required for the simplest client to view the video (both the schemes presented here require that the client has a minimum storage in order to view the video). We aim at minimizing the network load fluctuations and at providing guarantees

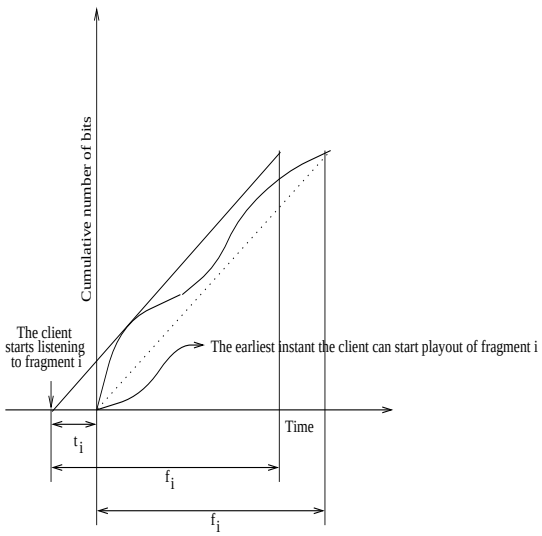

Figure 3: The broadcast and playout of a fragment i in the constant fragment broadcast scheme. The client has to buffer t_i time units of data corresponding to the fragment before it can start playout of that fragment.

for the various parameters: the client latency, client storage requirement and the client bandwidth requirement.

5.1 Constant Fragment Broadcast

In this scheme the video is partitioned into n fragments based on their playout durations. Each fragment is broadcast in its channel at the average playout rate of that fragment. Therefore the duration of broadcast of a fragment is the same as the duration of playout of that fragment. As the broadcast rate of a fragment is constant over the duration of that fragment we call this scheme the *Constant Fragment Broadcast* scheme. We examine this scheme more closely.

Let the video be partitioned into n fragments and the duration of broadcast of these fragments be given by $f_1, f_2, \ldots, f_n$. The duration of playout of a fragment is the same as the duration of broadcast of that fragment. Let the broadcast rates of these fragments be $b_1, b_2, \ldots, b_n$. In the constant fragment scheme, the client cannot always start the playout a fragment as soon as it receives the start of the fragment, instead the client might have to buffer a portion of the fragment before it can start playout of that fragment in order to avoid starvation in the middle of playout. Let the duration of the fragments that the client has to buffer before starting playout of the fragments be $t_1, t_2, \ldots, t_n$, where t_i corresponds to the duration of the fragment that has to be buffered before the client can start playout of the ith fragment. Figure 3 shows the earliest instant the client can start playout of fragment i given the broadcast of i that the client listens to.

Observation 7 *The network bandwidth required by the server to broadcast the video is* $\sum_{i=1}^{n} b_i$.

The server bandwidth required in this scheme per channel is not very much different from the average

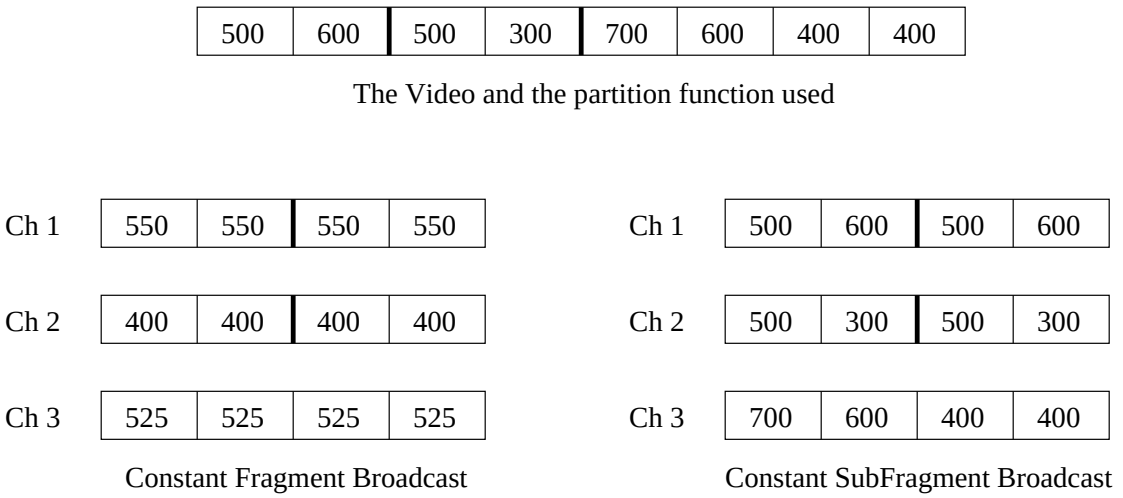

Figure 4: The different schemes for VBR broadcast. Each cell denotes a one second interval. The values shown indicate the amount of data (in kB) transmitted in that interval.

playout rate of the entire video, provided the fragment durations are determined judiciously. The main idea of a judicious fragment size is that the ratio of the number of frames of each type in all the fragments is the same. The server bandwidth required in this scheme per channel could actually be less than the average playout rate for some videos. Figure 4 shows such a video. This happens when the fragment with a broadcast rate greater than the average playout rate has a longer duration. In other words one should avoid short fragments with a high broadcast rate. In Figure 4, the fragment with broadcast rate of 525 kB/sec has a duration of 4 seconds, whereas the overall broadcast rate in the remaining 2 channels is given by 475 kB/sec per channel and the duration of these fragments are 2 seconds each. Therefore the overall server bandwidth required per channel is 492 kB/sec, whereas the average playout rate is 500 kB/sec.

The main drawback of this scheme is that the portion of the fragment which a client has to buffer before it can start playout of that fragment will vary depending on the video. Even for the same video it will be different for different fragments. Therefore accurate analysis of the growth of the partition function, the client latency and the client resource requirements will depend on the actual video. By an analysis similar to that in [6], we obtain the ith term for the fastest growing partition function in synchronous broadcast as $f_i = s_i + f_1 - (s_i \ \% \ f_1)$, where $s_i = \sum_{j=1}^{i-1} f_j + t_1 - t_i$. The actual terms which satisfy the above condition will depend on the video. But as a client has to buffer at most the entire fragment before it starts playout of that fragment, one can assume this worst case condition (i.e., $t_i = f_i, \ \forall i$) to obtain the partition functions for any video.

Observation 8 *The fastest growing partition function for the constant fragment broadcast scheme which ensures that the client with sufficient resources can start playout as soon as it has received all of the first fragment, irrespective of the video, is given by*

$f_1 = c$

$f_i = \lceil \frac{f_1 + \sum_{j=1}^{i-1} f_j}{2} \rceil$

Proof. The client will wait for at least f_1 time units before it starts playout of f_1. The server could have started broadcasting the first fragment as soon as the client requested the video. Therefore the number of time units before the client starts playout of the ith fragment is given by $f_1 + \sum_{j=1}^{i-1} f_j$. Now the client must have received all of the ith fragment by this time. Therefore the largest possible f_i is given by $f_i = \lceil \frac{f_1 + \sum_{j=1}^{i-1} f_j}{2} \rceil$ □

The above partition function satisfies the worst case criterion, that is it assumes that the client has to buffer the entire fragment before playing out that fragment. The maximum client latency is given by $2 \times c$. The first few terms in the partition function when $f_1 = 1$ are $1, 1, 2, 3, 4, 6, 9, \ldots$. This partition function requires that the server broadcast a 120 minute video in 21 channels if the duration of the first fragment is one second. The maximum client latency is 2 seconds and the client might have to listen to all 21 channels at the same time. The client storage required is almost 67% of the duration of the whole video, which will be approximately 67% of the size of the whole video if the client has to buffer the entire fragment before it starts playing out that fragment.

The series $1, 1, 2, 2, 2, 4, 4, 8, 8, 16, 16, \ldots$ satisfies the condition in Observation 11. As this partition function has several advantages over other partition functions, we use this series for evaluating the various parameters for sample videos.[9] Now we examine a scheme where the partition function and the client resource requirements will be similar to those for the broadcast of CBR videos. This scheme also provides "acceptable" fluctuations in the server network bandwidth.

5.2 Constant Subfragment Broadcast

This scheme guarantees less client latency and client resource requirements than those for the *constant fragment broadcast scheme* by introducing "acceptable" fluctuations in the server bandwidth. In this scheme, the video is first partitioned into fragments based on the playout durations (the duration of broadcast of a fragment is the same as its duration of playout). Each fragment is then subdivided into subfragments, such that the playout duration of all the subfragments in the entire video is the same. The server will broadcast each fragment in subfragments one after the other in the same channel. Each subfragment will be broadcast by the server at a constant rate which is given by its average playout rate (averaged over the duration of playout of that subfragment). In other words the duration of broadcast of a subfragment is the same as the duration of playout of the subfragment. A client buffers the entire subfragment before it starts playout of that subfragment.[10]

This scheme depends on a good subfragment duration. We denote the duration of a subfragment as t time units. Each fragment consists of an integer number of subfragments. In order to minimize the server network load fluctuations, the number of frames of each type in all the subfragments must be the same. A judicious subfragment duration should also ensure a low maximum server

[9] We therefore need not hunt for good partition functions for each video. We can use this partition function for all videos, but the analysis of the client resource requirements and the client latency will depend on the actual video.

[10] For a subfragment it may not be necessary that the client buffer the entire subfragment before it starts playout of that subfragment. But the amount of data associated with a subfragment (typically 1 to 8 seconds) is so small that almost all clients can be assumed to have that much disk capacity.

bandwidth. Figure 4 shows the constant subfragment broadcast scheme where the subfragment duration is one second.

Let the video be divided into n fragments and the duration of broadcasts of the fragments be $f_1, f_2, \ldots, f_n$. The duration of playout of these fragments is also $f_1, f_2, \ldots, f_n$. Let the duration of the subfragment be t. Each f_i consists of an integer number of subfragments.

Observation 9 *The fastest growing partition function in synchronous broadcast where the client with sufficient resources can start playout as soon as it receives all of the first subfragment of the video is given by* $F = f_1, f_2, \ldots, f_n$, *where the terms are*
$f_1 = c$
$f_i = 2 \times f_{i-1}$, $2 \leq i \leq n$.

Proof. The server could have started broadcasting the first subfragment as soon as the client made the request. In this case the client starts playout after t time units. The number of time units before the start of the ith fragment in this case is $t + \sum_{k=1}^{i-1} f_i$. The client must have received t time units of f_i by this instant. If the broadcast of the ith fragment is synchronized with the broadcast of the first fragment, then we obtain the largest f_i as $f_i = f_1 + \sum_{k=1}^{i-1} f_i$. □

Note that this partition function is similar to the fastest growing partition function for CBR video. Also if $f_1 = c = 1$, then we obtain the series of powers of 2 as the fastest growing series. The maximum latency that a client with sufficient resources will experience is given by $t + c$.

As t increases the burstiness in the traffic is likely to decrease, but it will increase the client storage requirement and the maximum client latency. On the other hand, a smaller t means less client storage and latency, but more burstiness in the server traffic. Figure 5 shows the server bandwidth fluctuations in one channel for the constant subfragment scheme for four different subfragment durations of 1, 2, 4 and 8 seconds, for a fragment duration of 8 seconds.

Let b_{max} (Bytes/second) denote the maximum broadcast rate over all subfragments. Let the video be broadcast using a partition function such as the repeating powers of two which supports the simplest client (the simplest client must have some storage and bandwidth in this scheme, unlike for CBR video). Let a sophisticated client need to listen to a maximum of only m channels at any instant.

The simplest client that can be supported will have to listen to one subfragment and playout another subfragment at any time. The guaranteed storage requirement for the simplest client irrespective of the video is $b_{max} \times 2t \times d$. This is because in the worst case, the subfragment that the client is playing out can have a burst in its playout rate at the very end and is near zero elsewhere. The network bandwidth requirement for the simplest client is given by b_{max}.

The guaranteed maximum network bandwidth required for the sophisticated client is $m \times b_{max}$. The maximum storage required for the sophisticated client is given by $(\max\{f_i\} + t) \times b_{max} \times d$. This is because the maximum storage required for other subfragments just before the end of playout of a subfragment is given by $(\max\{f_i\}) \times b_{max} \times d$ (similar to the storage analysis for CBR video).

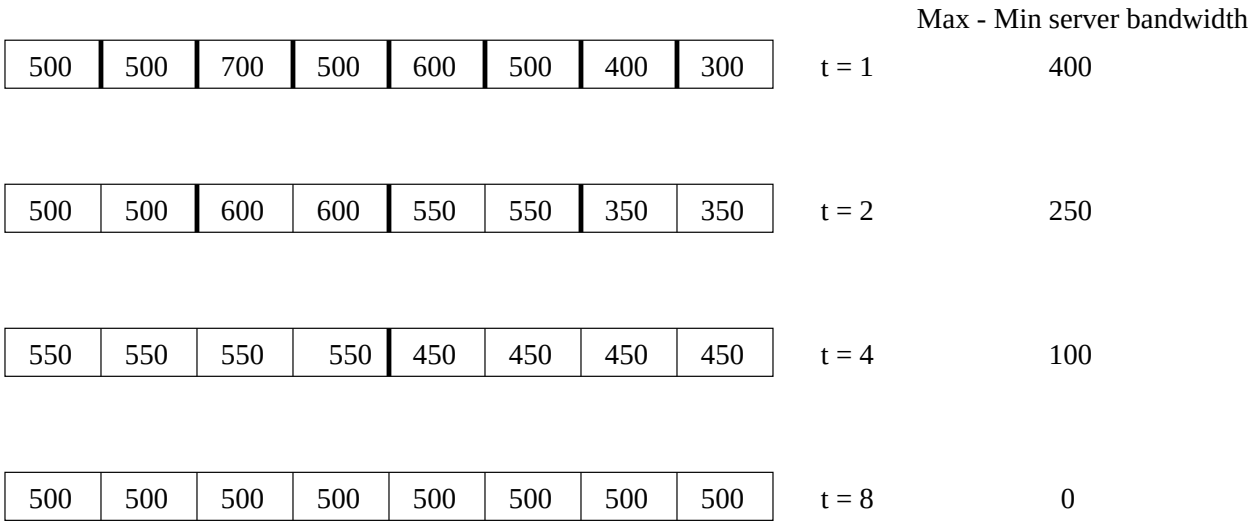

Figure 5: Server bandwidth fluctuations in one channel for the constant subfragment scheme for four different subfragment durations

The overall maximum storage required will be when the subfragment being played out currently has a burst in its playout at the very end. In this case the maximum storage required is given by $(\max\{f_i\} + t) \times b_{max} \times d$. Of course, accurate guarantees can be given while considering a particular video.

The constant subfragment scheme will perform well even if a video is being unicast to a client. The key idea here is that the initial client latency and the client storage can be exploited to decrease the server network bandwidth fluctuations inherent in a VBR video, provided we have a good subfragment duration. The scheme performs better when the video is broadcast in several channels. The main motivation behind the constant subfragment scheme is that there is not much variation between the sizes of two frames of the same type. For example, in a trace obtained from a James Bond movie, the ratio of the maximum I-frame size to the minimum I-frame size is around 10, whereas the ratio of the maximum frame size to the minimum frame size is around 125.

6 Experimental Results

In this section we analyze the schemes presented for VBR broadcast using a few sample videos. We use the partition function $1, 1, 2, 2, 2, 4, 4, \ldots$, where the first fragment duration is 8 seconds. The main parameters we are concerned with are the fluctuations in the server bandwidth, the minimum and maximum client resource requirements and the latency experienced by diverse clients.

We compute the various parameters for the subfragment durations of 1 sec, 2 sec, 4 sec and 8 sec, and also for the constant fragment broadcast scheme. One of the sample videos is a ten minute advertisement which can be downloaded from ftp://tenet.berkeley.edu/ pub/dbind/traces/. The remaining three are thirty minute videos clips of a James Bond movie, Jurassic Park and a tennis match. These were obtained from ftp-info3.informatik/uni-wuerzburg.de /pub/MPEG/ [10]. The

three videos are MPEG-1 encoded and the encoder parameters are given in Table 2.

encoder input	384 x 288 pel
color format	YUV (4:1:1, resolution of 8 bits)
quantization values	I=10, P=14, B=18
pattern	IBBPBBPBBPBB
GOP size	12
motion vector search	'Logarithmic' / 'Simple'
reference frame	'Original'
slices	1
vector/range	half pel / 10

Table 2: The encoder parameters

The 10 minute video is broadcast in twelve channels using the partition function $1, 1, 2, 2, 2, 4, 4, 8, 8, 16, 16, 16,$ where the duration of the first fragment is 8 seconds. The other 30 minute movies are broadcast using the partition function $1, 1, 2, 2, 2, 4, 4, 8, 8, 16, 16, 32, 32, 32, 32, 32$. Here again the duration of the first fragment is 8 seconds. This broadcast requires 16 channels for each video. The parameters for the different videos are shown in Table 3.

	Advertisement	Goldfinger	Jurassic Park	Tennis Match
Number of frames per second	30	24	24	24
Number of frames	16316	40000	40000	40000
Size (MB)	31.074	121.327	65.310	109.293
Max Frame size (kB)	82.560	244.592	119.632	190.856
Min Frame size (kB)	1.152	1.912	0.880	0.280

Table 3: The parameters for the four videos

Table 4 gives the values for the parameters for the various broadcast schemes. In the table, B_{min} denotes the minimum server network bandwidth and B_{max} is the maximum server network bandwidth. B_{min} and B_{max} were calculated by finding the broadcast rates of the different subfragments and determining the actual server bandwidth at all possible instances. s_{min} is the minimum storage that the client must have in order to guarantee that the client will view the video continuously ($s_{min} = b_{max} \times 2t \times d$), and s_{max} is the maximum client storage required to guarantee a maximum latency of $t + c$ time units ($s_{max} = (\max\{f_i\} + t) \times b_{max} \times d$). CSF$x$ denotes a constant sub fragment broadcast scheme where the duration of a subfragment is x seconds ($x = td$). There is not much variation in the network bandwidth requirement for both the simplest client and the sophisticated client for different subfragment durations and so we have not provided these values.

Let B_{avg} be the average playout rate of the entire video, and n be the number of channels required for broadcasting a video in both the schemes. C.b denotes the average playout rate of the entire video multiplied by the number of channels used for the other broadcast schemes (C.$b = nB_{avg}$). It is provided as a benchmark for the maximum server bandwidth required. One can notice that there is not much fluctuation in the server network bandwidth in the constant subfragment broadcast

scheme. This is due to the fact that all subfragments have the same number of frames of each type and the broadcast is done in several channels. Also the maximum server bandwidth required per channel for the constant subfragment broadcast is not very much different from the server bandwidth required per channel for the constant fragment broadcast scheme and also the average playout rate of the video. In fact, for the movie Goldfinger, the average playout rate is more than the maximum server bandwidth required per channel for any of the subfragment schemes.

	Advertisement	Goldfinger	Jurassic Park	Tennis Match
CSF8.B_{min} (Mbps)	5.448	6.201	4.889	8.219
CSF8.B_{max} (Mbps)	6.542	8.104	5.815	10.247
CSF8.s_{min} (kB)	1610	2128	1656	2139
CSF8.s_{max} (MB)	9.112	25.604	13.819	18.384
CSF4.B_{min} (Mbps)	5.185	6.172	4.774	8.135
CSF4.B_{max} (Mbps)	7.110	8.579	6.242	10.461
CSF4.s_{min} (kB)	1016	1210	850	1354
CSF4.s_{max} (MB)	8.815	25.145	13.416	17.991
CSF2.B_{min} (Mbps)	4.867	6.088	4.626	7.818
CSF2.B_{max} (Mbps)	7.295	8.660	6.516	10.477
CSF2.s_{min} (kB)	626	724	460	728
CSF2.s_{max} (MB)	8.62	24.902	13.221	17.678
CSF1.B_{min} (Mbps)	4.801	6.088	4.374	7.771
CSF1.B_{max} (Mbps)	7.466	8.680	6.968	10.484
CSF1.s_{min} (kB)	380	454	274	366
CSF1.s_{max} (MB)	8.497	24.767	13.128	17.497
CF.b (Mbps)	6.443	7.285	5.265	9.045
C.b (Mbps)	5.484	9.33	5.024	8.407

Table 4: The values for the various schemes

For comparing the different subfragment durations, we plot the server bandwidth fluctuations (Figure 6) and also the minimum storage requirements for different subfragment durations (Figure 7). One can observe that the graphs are near linear and so a good trade-off between the server network bandwidth fluctuations and the client resources and latency can be obtained.

7 Conclusions and Future work

In this paper we examined the various parameters in broadcasting videos: the server bandwidth, the client bandwidth, the client storage and the client latency. We examined the partition functions that can support heterogeneous clients. We also examined how the client can wait a judicious amount of time and then playout the video with the resources it has available. We then examined schemes for broadcasting VBR videos. The analysis of VBR videos requires another parameter, the server network bandwidth fluctuations. We proposed the constant subfragment scheme for broadcasting VBR videos in which by increasing the server network bandwidth fluctuations, the client latency

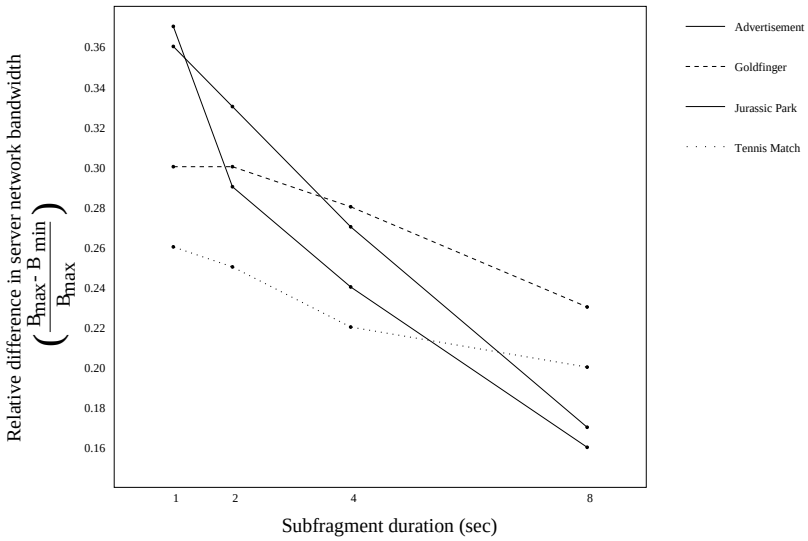

Figure 6: Server Bandwidth fluctuations for different subfragment durations

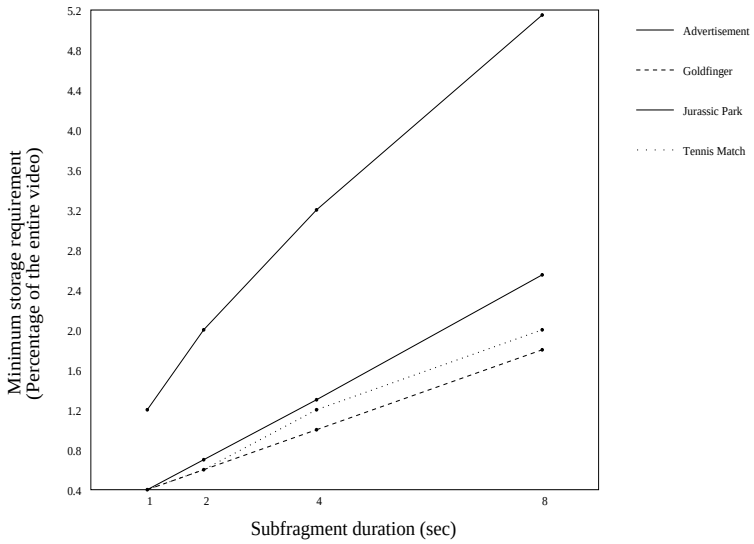

Figure 7: Minimum client storage required for different subfragment durations

and storage requirement can be decreased.

Our work supported only the play functionality for videos. Future work can examine providing VCR functionalities such as fast forward, rewind, pause and stop for videos. Also we did not consider client network bandwidth fluctuations. Future work could examine minimizing the client network bandwidth fluctuations. We also examined only clients which are disk conserving. But clients with large disk capacity need not be disk conserving. Future work can examine broadcast schemes for such clients which can receive fragments out of order or receive the second half of a fragment before receiving the first half of the fragment.

References

[1] C. C. Aggarwal, J. L. Wolf and P. S. Yu, "A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand Systems", *Proc. of the IEEE Int'l Conf. on Multimedia Systems* June 1996

[2] C. C. Aggarwal and A. S. Kellerman, "Distributed Multimedia", *Addison Wesley, ACM Press*

[3] A. Dan, D. Sitaram and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *Proc. of ACM Multimedia*, Oct 1994, pp. 15 - 23

[4] A. Dan, D. Sitaram and P. Shahabuddin, "Dynamic Scheduling Policies for an On-Demand Video Server", *Multimedia Systems*, June 1996, pp. 112 - 21

[5] D. L. Eager and M. K. Vernon, "Dynamic Skyscraper Broadcasts for Video-on-Demand Systems", *Technical Report # 1375, C.S. Dept, UW-Madison*, May 1998

[6] L. Gao, J. Kurose and D. Towsley, "Efficient Schemes for Broadcasting Popular Videos"

[7] K. A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems", *Proc. of the ACM SIGCOMM'97 Conf.*, Sep 1997, Cannes, France, pp. 89 - 100

[8] L. S. Juhn and L. M. Tseng, "Harmonic Broadcasting for Video-on-Demand Systems", *IEEE Transaction on Broadcasting*, Sep 1997, pp. 268 - 71

[9] L. S. Juhn and L. M. Tseng, "Enhanced Harmonic Data Broadcasting and Receiving Scheme for popular video services", *IEEE Transaction on Broadcasting*, May 1998, pp. 343 - 6

[10] O. Rose, "Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems", *Proc. of the 20th Annual Conf. on Local Computer Networks*, 1995, Minneapolis, MN, pp. 397 - 406

[11] S. Viswanathan and T. Imielinski, "Pyramid Broadcasting for Video-on-Demand Service", *Proc. of the SPIE Multimedia Computing and Networking Conf.*, Feb 1995, San Jose, CA, pp. 66 - 77

[12] S. Viswanathan and T. Imielinski, "Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting", *Multimedia Systems*, Aug 1996, pp. 197 - 208

[13] J. Zhang and J. Y. Hui, "Static and Dynamic Resource Allocation Algorithms for Real-Time VBR Video Transmissions in Multimedia Networks", *Intl. Conference on Information, Communications and Signal Processing*, Sep 1997