A SIMPLE BUT EFFICIENT BROADCASTING PROTOCOL FOR VIDEO-ON-DEMAND

Jehan-François Pâris[1]

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-05-07

April 4, 2005

Keywords: video-on-demand.

Abstract

We present a simple fixed-delay broadcasting (SFDB) protocol for video-on-demand. Our protocol assumes that each video to be broadcast will be partitioned into segments of equal duration to be transmitted over a fixed number of video channels. In addition, it requires all customers to wait for the same fixed delay before watching the video they have selected. Our protocol uses time-division multiplexing to obtain the best transmission schedule for the channel that broadcasts the first segments of the video. The same multiplexing scheme is then reproduced on all the remaining channels. Despite its simplicity, our simple fixed delay broadcasting protocol achieves waiting times comparable to those of much more sophisticated broadcasting protocols. We also show how the protocol can be modified to handle set-top boxes that cannot receive data at more than two or three times the video consumption rate.

[1] Supported in part by the National Science Foundation under grant CCR-9988390.

[2]

A Simple but Efficient Broadcasting Protocol for Video-on-Demand

Jehan-François Pâris[1]
Department of Computer Science
University of Houston
Houston, TX 77204-3010

paris@cs.uh.edu

Abstract

We present a simple fixed-delay broadcasting (SFDB) protocol for video-on-demand. Our protocol assumes that each video to be broadcast will be partitioned into segments of equal duration to be transmitted over a fixed number of video channels. In addition, it requires all customers to wait for the same fixed delay before watching the video they have selected. Our protocol uses time-division multiplexing to obtain the best transmission schedule for the channel that broadcasts the first segments of the video. The same multiplexing scheme is then reproduced on all the remaining channels. Despite its simplicity, our simple fixed delay broadcasting protocol achieves waiting times comparable to those of much more sophisticated broadcasting protocols. We also show how the protocol can be modified to handle set-top boxes that cannot receive data at more than two or three times the video consumption rate.

1. INTRODUCTION

Broadcasting protocols constitute the most efficient means for distributing popular videos on demand to large metropolitan audiences. Rather than waiting for customer requests, broadcasting protocols partition each video into segments and retransmit theses segments according to a fixed schedule guaranteeing that any customer having waited for a given maximum delay will be able to watch the whole video without any interruption. As a result, the number of customers watching the video being broadcast does not affect its bandwidth requirements.

The simplest broadcasting protocol for video-on-demand is *staggered broadcasting*. It consists of broadcasting the complete contents of each video on several channels at equal offsets. Hence, it requires k dedicated channels per video to achieve a customer waiting time equal to $1/k$ of the duration the video. More recent—and more complex—broadcasting protocols have achieved better customer waiting times at much lower bandwidth costs. For instance, one of the most recent protocols only requires six video channels to achieve a customer waiting time of thirty seconds for a two-hour video [7].

To achieve these excellent results, all recent broadcasting protocols for video-on-demand utilize complex transmission schedules that attempt to minimize the amount of bandwidth required to broadcast each segment of each video. We propose a different approach: rather than attempting to minimize the amount of bandwidth required to transmit all video segments, our *Simple Fixed-Delay Broadcasting* (SFDB) protocol focuses its optimization efforts on the first segments of each video and constructs the most efficient transmission schedule for these segments. Similar segment-to-channel mappings are then used for the remaining segments of the video ensuring that these segments are transmitted in an efficient, if not optimal, manner. The outcome of this procedure is a much simpler broadcasting protocol that nevertheless achieves customer waiting times comparable to those achieved by much more sophisticated protocols.

2. PREVIOUS WORK

Earlier video distribution protocols attempted to reduce server bandwidth either by batching together several requests [1] or by accelerating the video playback rate of new requests to let them catch up with previous transmissions [3]. Viswanathan and Imielinski [10] proposed in 1996 a better solution. Their *Pyramid Broadcasting* protocol required special customer set-top boxes (STBs) (a) capable of receiving data at rates exceeding the video consumption rate and (b) having enough buffer space to store one hour of video data. This allowed the server to distribute the different segments of each popular video according to a deterministic schedule ensuring that no customer would have to wait more than a few minutes. Their original proposal has been followed by several more recent schemes requiring less server bandwidth to achieve the same customer waiting times. We will only mention those protocols that are directly relevant to our work.

[1] Supported in part by the National Science Foundation under grant CCR-9988390.

First Channel	S_1	S_1	S_1	S_1
Second Channel	S_2	S_3	S_2	S_3
Third Channel	S_4	S_5	S_6	S_7

Figure 1. The first three channels for fast broadcasting

First Channel	S_1	S_1	S_1	S_1	S_1	S_1
Second Channel	S_2	S_4	S_2	S_5	S_2	S_4
Third Channel	S_3	S_6	S_8	S_3	S_7	S_9

Figure 2. The first three channels for the PB protocol.

Juhn and Tseng's *Fast Broadcasting* (FB) protocol [4] allocates to each video k data channels whose bandwidths are all equal to the video consumption rate b. It then partitions each video into 2^{k-1} segments, S_1 to $S_{2^{k-1}}$, of equal duration d. As Figure 1 indicates, the first channel continuously rebroadcasts segment S_1, the second channel transmits segments S_2 and S_3, and the third channel transmits segments S_4 to S_7. More generally, channel j with $1 \le j \le k$ transmits segments $S_{2^{j-1}}$ to S_{2^j-1}.

When customers want to watch a video, they wait until the beginning of the next transmission of segment S_1. They then start watching that segment while their STB starts downloading data from all other channels. By the time the customer has finished watching segment S_1, segment S_2 will either be already downloaded or ready to be downloaded. More generally, any given segment S_i will either be already downloaded or ready to be downloaded by the time the customer has finished watching segment S_{i-1}.

The *Pagoda Broadcasting* (PB) [5] protocol improves upon the FB protocol by using a more complex segment-to-channel mapping. As seen in Figure 2, the PB protocol can pack nine segments into three channels while the FB protocol can only pack seven segments. Hence the segment size will be equal to one ninth of the duration of the video and no customer would ever have to wait more than 14 minutes for a two-hour video. Improved versions of the protocol, among which, the *New Pagoda Broadcasting* (NPB) [6] and the *Recursive Frequency-Splitting* (RFS) [9] protocols, use more sophisticated schedules to outperform the PB protocol. As a result, the RFS can map 26 segments into four channels and achieve a maximum customer waiting time equal to 1/26 of the duration of the video, that is, slightly more than four minutes and half for a two-hour video. Adding a fifth channel would allow the server to partition the video into 73 segments and achieve a waiting time of 99 seconds for a two-hour video [9].

None of these protocols require customers to wait for any minimum amount of time before watching the video of their choice. As a result, there is no point in requiring customer STBs to start downloading data while customers are still waiting for the beginning of the video. The *Fixed-Delay Pagoda Broadcasting* (FDPB) protocol [7] requires all users to wait for a fixed delay w before watching the video they have selected. This waiting time is normally a multiple m of the segment duration d. The FDPB protocol uses this delay to stretch the reception of the n segments of the video over a longer time interval. Previous Pagoda protocols required segment S_i to be repeated at least once every i slots to ensure the continuity of the video. With the FDBP protocol, segment S_1 has to be transmitted at least once every m slots to be always received before the customer starts watching the video. More generally, segment S_i has to be transmitted at least once every $m + i - 1$ slots.

Subchannel 0	S_1			S_2			S_3			S_1			S_2		
Subchannel 1		S_4			S_5			S_6			S_7			S_4	
Subchannel 2			S_8			S_9			S_{10}			S_{11}			S_{12}

Figure 3. The first channel for an FDPB protocol with m = 9 and 3 subchannels.

Figure 3 details the organization of the first channel of a FDPB protocol requiring customers to wait for exactly nine times the duration of a segment (m = 9). As we can see, the channel is partitioned through time-division multiplexing into 3 subchannels, each occupying 1/3 of the available slots. The first of these subchannels broadcasts segments S_1 to S_3 ensuring that each segment is repeated every 9 slots. The second subchannel broadcasts segments S_4 to S_7 ensuring that each segment is repeated every 12 slots. The third subchannel broadcasts segments S_8 to S_{12} ensuring that each segment is repeated every 15 slots. The same allocation process is repeated for the subsequent channels, selecting each the optimal number of subchannels. This allows the protocol to map 302 segments into four channels and achieve a deterministic waiting time of 9/302 of the duration of the video, that is, slightly less than four minutes for a two-hour video. Adding a fifth channel would allow the server to broadcast 802 segments and achieve a waiting time of 80 seconds for a two-hour video.

3. OUR PROTOCOL

We can draw two major conclusions from this brief review of broadcasting protocols for video-on-demand. First, fixed-delay protocols require significantly less bandwidth than protocols only asking customers to wait for the next occurrence of the first segment of the video to achieve equal maximum waiting times. Second, the first few segments of each video require much more bandwidth than their successors. Consider for instance the case of a FDP protocol with m = 9 broadcasting a video over five channels. The first channel will broadcast the first 12 segments of the video leaving the four remaining channels to broadcast the remaining 802 –12 = 790 segments of the

video. Hence 20% of the total bandwidth allocated to the video will be used to broadcast less than 1.5 percent of the video. This situation is not specific to the FDPB protocol. Consider the much simpler FB protocol. It can broadcast $2^5 - 1 = 31$ video segments over 5 channels, with segments S_1 to S_7 being transmitted by the first 3 channels. Hence, the first 7 segments of the video will occupy 60 percent of the total bandwidth, leaving only 40 percent of the bandwidth to the remaining 25 segments even though these segments represent 81 percent of the video.

These two observations suggest two important directions for the design of new broadcasting protocols for video-on-demand. First, fixed-delay broadcasting protocols should be preferred to protocols only asking customers to wait for the next occurrence of the first segment of the video. In addition to utilizing the server bandwidth better, fixed-delay protocols offer the advantage of being better suited to the broadcasting of MPEG videos. Since these protocols require that each and every segment of a video must be completely received by the STB before the customers start to watch it, they provide implicit forward buffering, which will eliminate most of the bandwidth fluctuations inherent to compressed video signal. Second, most, if not all, our efforts should be dedicated to the optimization of the segment-to-slot mapping of the first few video channels.

The Simple Fixed-Delay Broadcasting (SFPD) takes these two principles to the limit. It partitions each video into n segments of equal duration $d = D/n$ where D is the video length and requires all customers to wait for the same fixed delay $w = md$ before watching the video of their choice. To optimize the segment-to-slot mapping of the first video channel, it partitions it into $\sqrt{m}$ subchannels each occupying $1/\sqrt{m}$ of the channel bandwidth. Thus its segment-to-slot mapping for the first channel is identical to that of an FDPB protocol with the same m parameter. Unlike the FDPB protocol, our new protocol partitions all other video channels into the same number of subchannels as the first channel instead of trying to find the optimal number of subchannels for each channel.

Figure 4 describes in detail how our SFDB protocol allocates video segments. The two input parameters of the algorithm are the number m of segments the customer has to wait and the number k of channels allocated to the video. We first compute the number s of subchannels per channel, which we round to the nearest integer. We then start allocating segments to channels starting with the first segment. Given that this segment must be repeated at least once every m slots to be guaranteed to always arrive in time, we figure that the first subchannel of the first channel can broadcast at most $\lfloor m/s \rfloor$ segments and allocate segments S_1 to $S_{\lfloor m/s \rfloor}$ to that subchannel. We then continue the same process with the remaining $ks - 1$ subchannels observing that the number of segments that can be allocated to a given subchannel is limited by the maximum interval at which the lowest-numbered segment

Assumptions:

m is the number of segments customer has to wait
k is the number of video channels allocated to the video
s is the number of subchannels per channel
first [i, j] is the lowest-numbered segment broadcast by subchannel j of channel i
last [i, j] is the highest-numbered segment broadcast by subchannel j of channel i
n is the total number of segments into which the video will be partitioned
n_a is the number of segments already assigned to a subchannel

Algorithm:

```
s ← round(√m)
n_a ← 0
for i from 1 to k  begin
        for j from 1 to s  begin
                first [i, j ] ← n_a + 1
                last [i, j ] ← n_a + ⌊(first [i, j ] + m – 1)/s⌋
                n_a ← last [i, j ]
        end
end
n ← n_a
```

Figure 4. How the SFDB protocol allocates video segments.

allocated to the channel must be repeated to guarantee it will always arrive on time. So, if S_f is the lowest-numbered segment to be broadcast by a given subchannel, that subchannel will be able to broadcast $\lfloor (f + m - 1)/s \rfloor$ segments. The customer waiting time will then be equal to mD/n, where D is the duration of the video.

Table I details how an SFDB protocol with $m = 9$ allocates its first six channels. As one can see, each channel is subdivided into $\sqrt{9} = 3$ subchannels. Segments are allocated to subchannels in a purely sequential fashion starting with the first subchannel of the first channel, which has to broadcast segments S_1 to S_3 ensuring that these three segments will be broadcast once every nine slots.

There are several advantages to this simpler approach. First, it greatly simplifies the protocol. Second, it makes all subchannels interchangeable since they now have the same bandwidth and are multiplexed in the same fashion. This greatly simplifies the sharing of channels among videos. Rather than having an integer number of channels allocated to each video, we can now allocate some but not all of the subchannels of a channel to a specific video. We could have a given video broadcast on 4⅓ channels and another slightly longer one on 4⅔ channels. As we will see, the more regular structure of the protocol also makes it easier to develop variants of the protocol limiting the client bandwidth. The obvious disadvantage of our new approach is that the SFDB protocol cannot map as many segments in the same number of channels as the FDPB.

Table I. The first six channels for an SFDB protocol with $m = 9$

Channel	*Subchannel*	*First Segment*	*Last Segment*
C_1	1	S_1	S_3
	2	S_4	S_7
	3	S_8	S_{12}
C_2	1	S_{13}	S_{19}
	2	S_{20}	S_{28}
	3	S_{29}	S_{40}
C_3	1	S_{41}	S_{56}
	2	S_{57}	S_{77}
	3	S_{78}	S_{105}
C_4	1	S_{106}	S_{143}
	2	S_{144}	S_{193}
	3	S_{194}	S_{260}
C_5	1	S_{261}	S_{349}
	2	S_{350}	S_{468}
	3	S_{469}	S_{627}
C_6	1	S_{628}	S_{839}
	2	S_{840}	S_{1121}
	3	S_{1122}	S_{1497}

Recall that a FDPB protocol with $m = 9$ can broadcast 802 segments over 5 channels and achieve a waiting time of 80 seconds for a two-hour video. As we can see in Table I, an SFDB protocol with same value of m can only broadcast 627 segments over the same number of channels. It will thus only achieve a waiting time equal to 9/627 of the video duration, that is, 103 seconds for the same two-hour video.

Figure 5 compares the customer waiting times achieved by the SFDB and the FDPB protocols with 4 to 7 channels and selected values of m. All customer waiting times are expressed in fractions of the video duration. Hence a customer waiting time of 0.05 corresponds to a wait of two minutes for a two-hour video. As we can see, the gap between the performances of the two protocols narrows when m increases from 2 to 100. In addition, an SFDB protocol with a large value of m achieves lower customer response times than a FDPB protocol with a small value of m.

It would thus be tempting to assume that we could achieve even lower customer waiting times by using even larger values of m. This is not true as we would quickly approach the theoretical lower bound for a fixed-delay protocol using k video channels.

Consider a video of duration D and assume that all customers are willing to wait w time units between the time they have ordered the video and the time they can start watching it. Let b represent the video consumption rate and Δt a small time interval at a location t within the video. Assuming that each customer STB starts downloading video data from the moment the video is ordered, the contents of this time interval will have to be broadcast at a minimum bandwidth $b/(t+w)$ where b is the video consumption rate. Passing to the limit when Δt goes to 0, we see that the minimum bandwidth required to transmit the video is be given by

$$B_{\min} = \int_0^D \frac{b}{t+w} dt = b \log \frac{D+w}{w} \qquad (1)$$

From this equation, we can also derive the minimum waiting time $w_{\min}$ that can be achieved when the broadcasting bandwidth is equal to k video channels, which is

$$w_{\min} = \frac{D}{e^k - 1} \qquad (2)$$

Figure 6 compares the customer waiting times achieved by the SFDB protocol with those achieved by the Fast Broadcasting (FB) and the Recursive Frequency-Splitting (RFS) protocols. We selected the first protocol for its simplicity and the second for its excellent performance. In addition, the solid curve at the bottom represents the theoretical minimum waiting time that we have just derived. As in Figure 5, all customer waiting times are expressed in fractions of the video duration. We can see that our SFDB protocol always achieves lower customer waiting times than the FB protocol. In addition it outperforms the RFS protocol for sufficiently large values of its parameter m. The actual threshold was found to be $m = 36$.

Comparing the waiting times achieved by our SFDB protocol with the minimum customer waiting times derived from Equation 2, we can also see that we will never be able to derive a protocol that would achieve much lower waiting times than the SFDB protocol with a sufficiently large value of m.

There is one last aspect of the SFDB protocol we have to address, that is, its client storage requirements. To derive those, we need to observe that a STB downloading a video broadcast by the SFDB protocol will go through three phases, namely, one during which it receives more data than it consumes by displaying the video, a second during which the data arrival rate is exactly equal to the video consumption rate and a third during which the data arrival rate will be lower than the video consumption rate. To estimate the client storage requirements of the protocol, we need to measure the number of video segments stored in the STB at any moment when the data arrival rate is exactly equal to the video consumption rate. The STB will enter that state when it has just terminated receiving data from the first $k-1$ channels and leave that state when it stops receiving data from the first subchannel of the last channel.

Consider the contents of the STB at the time it has just terminated receiving data from the first $k-1$ channels. Let m_{last} designate the number of slots elapsed since the

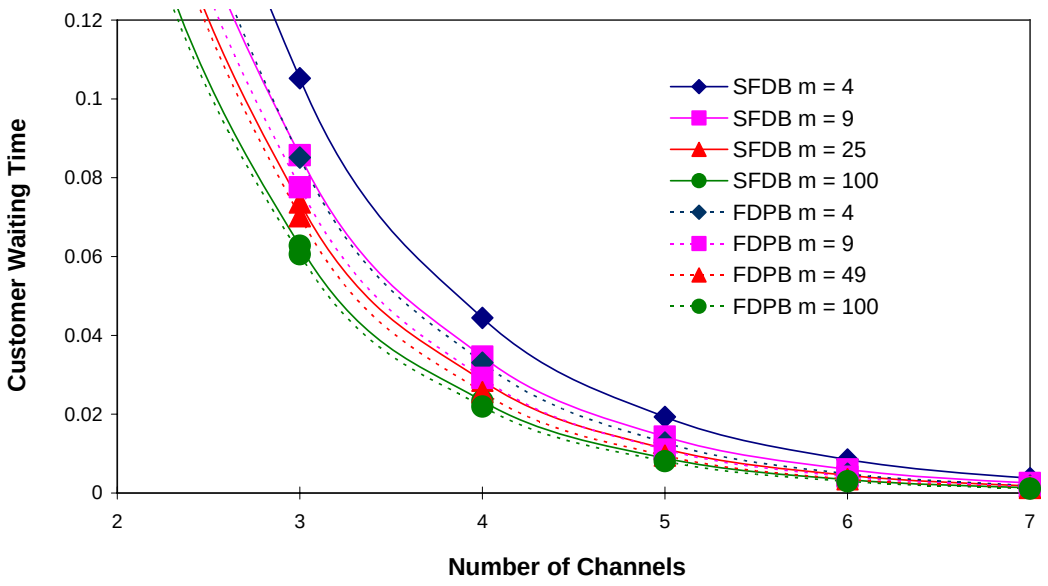

Figure 5. Compared customer waiting times of the SFDB and the FDPB protocols for different values of *m*.

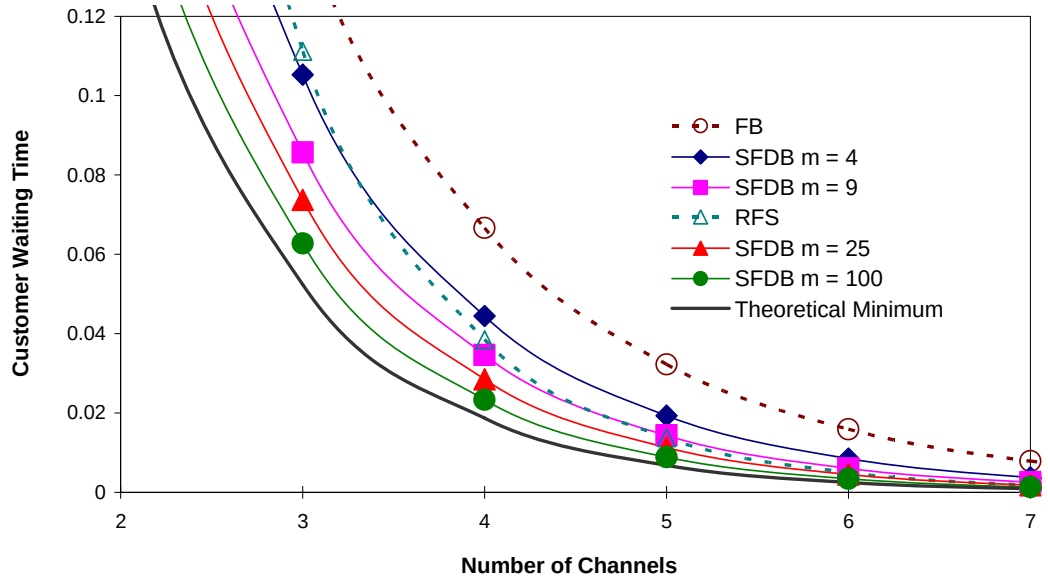

Figure 6. Compared customer waiting times of the SFDB, the FB and the RFS protocols.

time the customers ordered the video and s_{last} represent the index of the highest-numbered segment broadcast by channel C_{k-1}. At that time, the customer STB will contain all the $s_{last} - (m_{last} - m)$ segments it has received from the first $k - 1$ channels but not yet played and all the m_{last} segments it has already received from the last channel as none of them has already been played. The total number of segments in the customer STB will thus be equal to

$$N_{max} = s_{last} - (m_{last} - m) + m_{last} = s_{last} + m$$

Returning to Table I, we can see that the highest numbered segment broadcast by channel C_5 is segment S_{627}. We can infer that a SFDB protocol broadcasting a

video over 7 channels with a customer waiting times equal to 9 times the duration of a segment will store in the customer STB up to 636 segments, that is, 636/1497 or 42.5 percent of the video size.

We found that the client storage requirements of the protocol were a decreasing function of both the number k of channels assigned to the video and the number m of segments the customers had to wait before watching the video they had ordered. The maximum customer storage requirements we observed were 60 percent of the video for $k = m = 2$, a combination that we are not likely to encounter as it provided a waiting time equal to 27 percent of the video duration. The client storage requirements for more reasonable values of k, that is, values of k providing waiting times not exceeding 5 percent of the duration of the video, remained below 47 percent of the video size. These values are comparable to those achieved by other broadcasting protocols for video-on-demand.

4. LIMITING THE CLIENT BANDWIDTH

Like most other broadcasting protocols, our SFDB protocol assumes that all customer STB can and will simultaneously receive data from the k channels on which the various segments of the video are broadcast. This requirement complicates the design of the STB and increases its cost.

One possible approach to this problem is to restrict the STB receiving bandwidth to a given multiple $k' < k$ of the video consumption rate. For instance, the *Skyscraper Broadcasting* protocol [2] never requires the customer STB to receive data from more than two channels at the same time. This approach has a major drawback, namely a very significant increase in the server bandwidth required to distribute the videos. Hence, the potential savings in STB costs achieved by skyscraper broadcasting cannot be achieved without bigger video servers and a costlier network infrastructure.

We propose here a less radical implementation of the same concept, namely, reducing the client bandwidth requirements of an existing protocol to two or three concurrent channels. As we will see, this approach will result in very moderate increases of the server bandwidth.

Consider a modified FDPB protocol with $m = 9$ and 3 subchannels per channel that restricts its client bandwidth to 2 channels. As shown in Table II, the segment-to-subchannel mappings of the two first channels are unchanged. The first mappings to be affected are those of channel C_3, because the STB must now wait until it has received all data from the first subchannel of channel C_1 before starting to receive data from the first subchannel of channel C_3. Since the first subchannel of channel C_1 broadcasts 3 segments and occupies 1/3 of the slots of its channel, it will repeat itself every 9 slots. Hence the STB will have to wait exactly 9 slots before starting to receive data from the first subchannel of channel C_3. The lowest-numbered segment broadcast by that subchannel is segment S_{41}. With the original SFDB protocol, it had to be broadcast at least once every $41 + 9 – 1 = 49$ slots. Since the STB will now have to wait 9 slots before receiving data from the subchannel broadcasting segment S_{41}, that segment will now have to be broadcast at least once every $49– 9 = 40$ slots. Similarly segment S_{42} will now have to be broadcast at least once every 41 slots instead of every 50 slots and so on. Since all subchannels occupy exactly 1/3 of the slots of their channel, the first subchannel of channel C_3 will be able to broadcast exactly $\lfloor 40/3 \rfloor = 13$ segments, that is, segments S_{41} to S_{53}. The same process will be applied to the second subchannel of channel C_3, observing that the STB will not be able to receive data from this subchannel until it has finished receiving data from the second subchannel of channel C_1. After that, it will be repeated for the third subchannel of channel C_3, then to all subchannels of channel C_4 and so on. The outcome of this procedure is summarized in Table II.

Table II. The first six channels for a modified SFDB protocol with $m = 9$ limiting its client bandwidth to 2 channels

Channel	*Subchannel*	*First Segment*	*Last Segment*
C_1	1	S_1	S_3
	2	S_4	S_7
	3	S_8	S_{12}
C_2	1	S_{13}	S_{19}
	2	S_{20}	S_{28}
	3	S_{29}	S_{40}
C_3	1	S_{41}	S_{53}
	2	S_{54}	S_{69}
	3	S_{70}	S_{90}
C_4	1	S_{91}	S_{116}
	2	S_{117}	S_{148}
	3	S_{149}	S_{188}
C_5	1	S_{189}	S_{237}
	2	S_{238}	S_{299}
	3	S_{300}	S_{375}
C_6	1	S_{376}	S_{470}
	2	S_{471}	S_{588}
	3	S_{589}	S_{735}

Figure 7 presents a more general description of the algorithm used to map the segments into subchannels. The algorithm has three inputs, namely, the number m of segments the customer has to wait before starting to watch the video, the number k of video channels allocated to the video, and the number k' of video channels the client STB can receive at the same time.

Assumptions:

m is the number of segments customer has to wait
k is the number of video channels allocated to the video
k' is the number of channels the client STB can receive at the same time
s is the number of subchannels per channel
delay[*i, j*] is the number of slots the client must wait before receiving data from| subchannel *j* of channel *i*
first [*i, j*] is the lowest-numbered segment broadcast by subchannel *j* of channel *i*
number[*i, j*] is the number *of* segments broadcast by subchannel *j* of channel *i*
last [*i, j*] is the highest-numbered segment broadcast by subchannel *j* of channel *i*
n is the total number of segments into which the video will be partitioned
n_a is the number of segments already assigned to a subchannel

Algorithm:

$s \leftarrow \text{round}(\sqrt{n})$
$n_a \leftarrow 0$
for *i* from 1 to *k* begin
  for *j* from 1 to *s* begin
    delay[*i, j*] ← 0
  end
end
for *i* from 1 to *k* begin
  for *j* from 1 to *s* begin
    first [*i, j*] ← $n_a + 1$
    number[*i, j*] ← $\lfloor$(*first* [*i, j*] + *m* – 1 – *delay*[*i,* j])/*s*$\rfloor$
    last [*i, j*] ← n_a + *number*[*i,* j]
    $n_a \leftarrow$ *last* [*i, j*]
    delay[*i*+*k', j*] ← *delay*[*i, j*] + *s*×*number*[*i, j*]
  end
end
$n \leftarrow n_a$

Figure 7. How a modified SFDB protocol limiting its client bandwidth to *k'* video channels allocates video segments.

Such a simple algorithm would not have been possible with the FDPB protocol because FDPB partitions each channel into a different number of subchannels. One possible solution [7] is to require the STB to wait until it has received all the data transmitted by channel C_1 before allowing it to receive any data from channel C_3. This introduced additional delays and produced less than optimum segment to subchannel mappings. A more recent algorithm [8] achieved better segment to subchannel mappings but required complex adjustments in the number and bandwidths of the subchannels of all high-numbered channels, starting with channel C_3. These adjustments are not required with our SFDB protocol because all subchannels have the same bandwidth.

Figure 8 compares the customer waiting time achieved by a modified SFDB protocol limiting its client bandwidth to two video channels with those achieved by the original SFDB protocol, the FDPB and the Skyscraper Broadcasting protocol. We selected an "unconstrained" version of the Skyscraper Broadcasting that does not place any restriction on the number of segments that are broadcast by each channel because it achieves shorter customer waiting times than versions of the protocol restricting that number to a maximum width *W*. As on previous graphs, customer waiting times are expressed in fractions of the video duration while bandwidths are expressed in video channels. We can immediately see that the modified SFDB protocol achieves much lower customer waiting times than a Skyscraper Broadcasting protocol using the same number of video channels. For instance, a modified SFDB protocol with $m = 9$ that limits its client bandwidth to two channels can achieve a lower customer waiting time with 5 channels than a Skyscraper Broadcasting protocol requiring 7 channels. Increasing the parameter *m* of the SFDB protocol results in even lower waiting times.

We should mention than the Skyscraper Broadcasting protocol has the dual objective of limiting both the client bandwidth and the client storage requirements of the protocol while our modified SFDB protocol only limits its client bandwidth. It should be relatively easy to limit the storage requirements of any SFDB protocol by limiting the number of segments transmitted by each individual subchannel. We did not pursue that avenue as the continuous increase of memory and disk drive storage capacities make that objective less important today than it was when the Skyscraper Broadcasting protocol was introduced.

5. CONCLUSION

We have presented a simple broadcasting protocol for video-on-demand that performs as well as much more sophisticated protocols. This excellent performance was due to two factors. First, we selected a fixed-delay policy requiring all customers to wait for the some amount of time. Second, we partitioned the first video channel into the optimal number of subchannels for each customer waiting time to segment duration ratio *m*. To keep the protocol as simple as possible, we did not attempt to optimize the remaining video segments in a similar fashion and decided instead to partition all channels into the same number of subchannels. Despite its simplicity, our *Simple Fixed-Delay Broadcasting* (SFDB) protocol achieves waiting times comparable to those of much more sophisticated broadcasting protocols, such as the Fixed-Delay Pagoda Broadcasting protocol. We have also shown how the more regular structure of the protocol made it much easier to develop variants of the protocol limiting the client bandwidth.

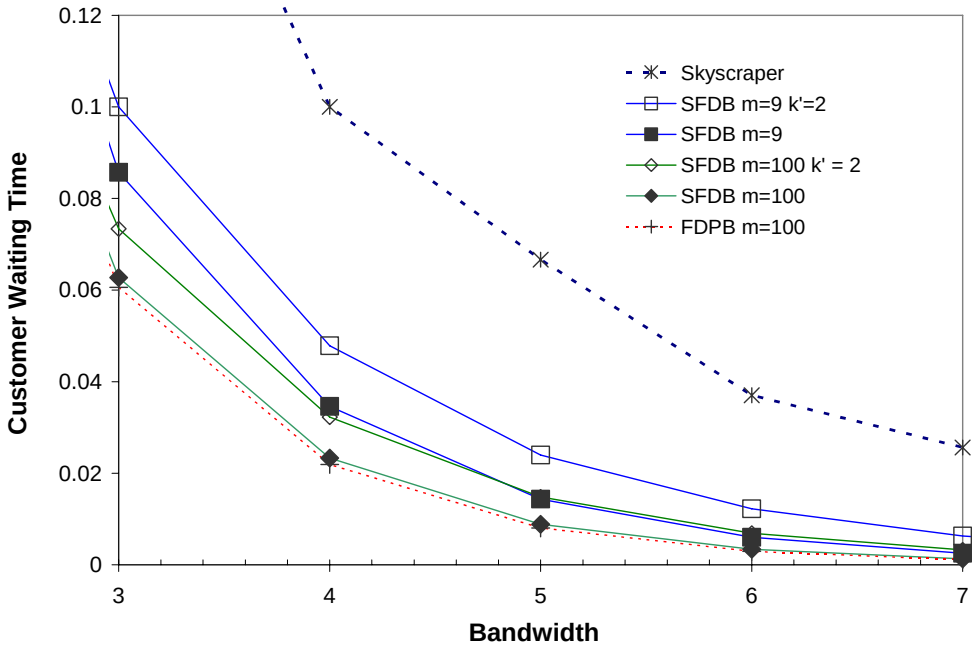

Figure 8. Compared customer waiting times of a modified SFDB protocol limiting the client bandwidth to two channels with those achieved by, the original SFDB protocol, the FDPB and the Skyscraper Broadcasting protocols

.

We believe we can derive two general conclusions from this study. First, broadcasting protocols that require their customers to wait for a fixed delay before watching the video of their choice are inherently more efficient than protocols that only require their customers to wait for the next broadcast of the first segment of the video. They should thus be our first choice. Second, all protocol optimization efforts should focus on reducing the bandwidth required to distribute the first few segments of the video as these segments are the most costly to distribute. Conversely, there is little incentive for developing better techniques for distributing the remaining segments of the video in the most efficient fashion as any reasonable solution will reach satisfactory results.

More work is still needed to evaluate how our SFDB protocol could handle video s in MPEG format and how the minimum frequencies at which each individual segment of the video should be adjusted to reflect the bit-rate fluctuations inherent to any compressed video signal.

REFERENCES

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

[2] K. A. Hua and S. Sheu, Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. *Proc. SIGCOMM 97 Conf.*, pp. 89–100, Sept. 1997.

[3] L. Golubchik, J. Lui, and R. Muntz. Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers. *Multimedia Systems*, 4(3): 140–155, 1996.

[4] L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Trans. on Broadcasting*, 44(1):100–105, March 1998.

[5] J.-F. Pâris, S. W. Carter and D. D. E. Long. A hybrid broadcasting protocol for video on demand. *Proc. 1999 Multimedia Computing and Networking Conf.*, pp. 317–326, Jan. 1999.

[6] J.-F. Pâris. A simple low-bandwidth broadcasting protocol for video-on-demand, *Proc. 8th Int. Conf. on Computer Communications and Networks*, pp. 690–697, Oct. 1999.

[7] J.-F. Pâris. A fixed-delay broadcasting protocol for video-on-demand, *Proc. 10th Int. Conf. on Computer Communications and Networks*, pp. 418–423, Oct. 2001.

[8] K. Thirumalai, J.-F. Pâris and D. D. E. Long. Tabbycat: an inexpensive scalable server for video-on-demand. *Proc. 2003 IEEE Int. Conf. on Communications*, pp. 896–900, May 2003.

[9] Y.-C. Tseng, M.-H. Yang and C.-H. Chang. A recursive frequency-splitting scheme for broadcasting hot videos in VOD service. *IEEE Trans. on Communications*, 50(8):1348–1355, Aug. 2002.

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