Multimedia Tools and Applications, 20, 67–81, 2003

An Efficient Channel Allocation Technique for Multiple Videos-on-Demand

PRASAN KUMAR SAHOO
JANG PING SHEU sheujp@csie.ncu.edu.tw
Department of Computer Science and Information Engineering, National Central University, Chung-Li 32054, Taiwan, Republic of China

Abstract. There are several broadcasting protocols for video-on-demand (VOD). Most of these protocols are tailored to handle the distribution of single video for a specific range of request arrival rates. In order to distribute multiple videos, broadcasting protocols like fast broadcasting (FB), new pagoda broadcasting (NPB) and universal distribution (UD) require more channels that are proportional to the number of hot videos to be broadcast. We present here an efficient broadcasting protocol in which, channel numbers can be reduced when multiple videos are broadcast either simultaneously or asynchronously. During the low to moderate request rates, our protocol performs best as similar to other reactive protocols and saves bandwidth and requires lesser number of channels as compared to other proactive or reactive protocols.

Keywords: broadcasting, video-on-demand, channel allocation

1. Introduction

Video-on-demand is an emerging distributed multimedia application which has recently drawn much attention from several quarters like entertainment, education, telecommunication and computer industries. In the current state of technology, though VOD is too expensive as compared to the video cassette rentals and pay-per-view, still it is widely considered that it will become an important residential service which can substitute current home entertainment and information services.

The main components of a VOD broadcasting system are the management service with a database and user desktop with the client's request interface. There are several proposals with or without enough buffer space at the client side. The first video distribution protocols have attempted to reduce bandwidth either by batching several requests together [2, 3] or by accelerating the video play back rate to allow the new requests to catch up with previous transmissions [5]. However this situation changed when Viswanathan and Imielinski [12] proposed to add to the customer set top box (STB) enough buffer space to store 40–50% of the video data. The *skyscraper broadcasting* (SB) [6] emphasizes on reducing the buffer size of STB and the number of data streams the STB has to receive at any moment. But SB requires more server bandwidth than *fast broadcasting* (FB) [8] and *new pagoda broadcasting* (NPB) protocol [9] to guarantee the same maximum waiting time. The *dynamic skyscraper broadcasting* [4] protocol has improved the *skyscraper broadcasting* by making it more dynamic, but it requires a higher server bandwidth than the *skyscraper broadcasting* protocol.

There are several proposals to reduce the bandwidth requirements of VOD services. Some of these proposals can be grouped under reactive approaches and others come under the proactive approaches with a common name of broadcasting protocols [13]. In the reactive approach, whenever several requests come for the same video in a close succession, the server will transmit only once and all the data can be shared by two or more requests. On contrary to the above case, proactive approach anticipates customer demand and distributes various segments of each video according to the deterministic schedules. Though each of these approaches has their own advantages and disadvantages, broadcasting protocol is the best technique to distribute multiple hot videos over a very large customer base. But reactive protocol performs much better than broadcasting protocol for the low request arrival rates [11] or for a smaller customer base. There are several proposals to handle the reactive approaches and *universal distribution* (UD) protocol [10] is one of them. This protocol performs very well at the low to medium request arrival rates and performance at high request arrival rates is not better than any of the broadcasting protocols. The limitations of UD protocol have been rectified by the *dynamic heuristic broadcasting* (DHB) protocol [1] which reasonably performs well at all request arrival rates. But both of the above protocols have primarily designed to handle request for a single video. In order to distribute multiple hot videos, these protocols require more channels that are proportional to the number of videos. Also, in the true VOD, subscribers may want to see various hot videos at different period of a day. Since the choice varies from customer to customer, there is chance of concurrent or asynchronous requests for different hot videos at different point of time. Several proposals have given for different arrival rates to save bandwidth, but neither of the protocols suggests how to handle requests for different hot videos by which bandwidth can be saved.

In this paper we propose a reactive approach which can distribute multiple videos at different request rates. Though our reactive approach is based on the UD protocol, we save more bandwidth as compared to any other protocol when multiple videos are distributed. Apart from saving more bandwidth, our protocol never put any additional implementation burden at the client and server side. In our protocol, we propose to divide the whole broadcasting scheme into different *blocks* and *slots* by which requests for different hot videos can be handled very efficiently to allocate more or less number of channels at different *slots* depending on the demand of a particular video.

The rest of the paper is organized as follows. In Section 2, we describe the related works. The description of our protocol has been depicted in Section 3. In Section 4, we have presented the transmitting and receiving algorithms of the both client and the server. Section 5 provides the simulation result and its discussion. Conclusions are given in Section 6.

2. Related works

In this section we focus our discussions on several protocols that are relevant to our present work. We have reviewed *fast broadcasting* (FB) scheme and *universal distribution* (UD) scheme which have relations to our protocol.

2.1. Review of the fast broadcasting scheme

Consider a video V of length D with consumption ratio b. Let there are k-channels, $C_0, C_1, C_2, \ldots, C_{k-1}$, each of bandwidth b, are assigned to V. The video server uses the following rule to broadcast the hot video V.

1. Partition the video V into N data segments $S_1, S_2, S_3, \ldots$ and S_N, where $N = 2^k - 1$. That is the concatenation $S_1 \bullet S_2 \bullet S_3 \bullet \cdots \bullet S_N = V$.
2. The length of each segment $d = D/N$.
3. Divide each channel C_i into slots of duration d for $i = 0, 1, 2, \ldots, (k-1)$. In each C_i, broadcast data segments S^i_2, S^i_{2+1}, …, and S^{i+1}_{2-1} periodically and in that order.

For example, channel C_0 broadcasts the first segment S_1 periodically; C_1 broadcasts the next two segments S_2 and S_3 periodically. C_2 broadcasts the next four segments S_4, S_5, S_6, S_7 periodically. The broadcasting scheme of FB protocol is shown in figure 1.

As shown in figure 1, suppose the video server allocates $k = 4$ channels to V. So V will be partitioned into $N = 2^4 - 1 = 15$ segments. For a client starting at time slot t_0, it will receive segments S_1, S_2, S_4 and S_8 (represented here in light gray) from channels C_0, C_1, C_2 and C_3, respectively. During time slot t_1, segment S_1 will be consumed and the other segments S_2, S_4, and S_8 will be buffered at the client's local storage for the future use. In slot t_2, the client will consume segment S_2 from its local storage. At the same time, segments S_3, S_5 and S_9 from C_1, C_2, and C_3, respectively will be buffered. In slot t_3, the client will consume S_3 from its local storage and simultaneously buffer S_6 and S_{10} from C_2 and C_3 respectively. The consumed segments have been represented in deep gray color in figure 1. This process will be repeated until the client has received $2^3 = 8$ data segments from C_3. At last the client will finish watching the video at the time $t_1 + Nd = t_1 + 15d$. The *FB scheme* allows a client to start at the beginning of any time slot by ensuring that whenever a segment is needed to be consumed, either it has been buffered previously or it is being broadcast just-in-time.

In this scheme, the client has to pay for extra buffer space to store premature segments. The maximum buffer requirement is les than half of the video size, *Db/2* [7]. In some cases it is possible for a client to play the video without buffering it, if it waits longer. For example, in figure 1 a client starting at time t_2 does not need to buffer any segment because it can continuously receive every required segment just-in-time from one of the channels. However, this may happen only once in every 2^{k-1} time slots.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
C_0	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	.
C_1	S_2	S_3	S_2	S_3	S_2	S_3	S_2	S_3	S_2	.
C_2	S_4	S_5	S_6	S_7	S_4	S_5	S_6	S_7	S_4	.
C_3	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_8	.

Figure 1. Segment scheduling of the FB scheme.

The *new pagoda broadcasting* (NPB) [9] is an improvement to the FB protocol in terms of less waiting time. But its proactive approach as similar to FB protocol requires more bandwidth. Also it gives poor performance during low to moderate request rates.

2.2. *Review of universal distribution scheme*

The UD *protocol* is based on the FB protocol whose segment-to-slot mapping is easier to manage. Let us consider a video V of length D which can be distributed in k channels. Then the UD *protocol* can be summarized as follows:

1. The video V is partitioned into $(2^k - 1)$ segments of equal duration d. These $(2^k - 1)$ segments will be broadcast in k logical channels with segments S_2^{j-1} to S_{2-1}^{j} that are assigned to channel j.
2. Each channel j has a start slot b_j whose initial value is not defined and its value depends on the slot at which the request is made.
3. When a request arrives during time slot i, the server first looks at the current segment distribution schedule channel by channel:
   (i) If the scheduled segment transmission for channel j is before slot $(i + 2^{j-1})$ then $b_j = (i + 2^{j-1})$ becomes the new start slot for the channel j.
   (ii) For any slot greater than i, if the segment S_l of channel j has not been scheduled, then the server will schedule a new transmission of S_l in slot $[b_j + (l - 2^{j-1})]$.

For example, as shown in figure 2, an incoming request arriving during slot 0 is followed by two other requests respectively arriving during slots 3 and 4. To handle the first request, we allocate segment S_1 in channel $C_1$and segments S_2 and S_3 in channel C_2. The segments S_4, S_5, S_6 and S_7 are allocated in channel C_3 as shown in figure 2. For the requests at slots 3 and 4, S_1 will be allocated in channels C_1 at slots 4 and 5. The segments S_2 and S_3 will be allocated in slots 5 and 6 respectively so that both segment transmissions can be shared with the third request. The segments S_5, S_6 and S_7 that have been allocated in channel C_3 can be shared too.

The *dynamic heuristic broadcasting* (DHB) [1] protocol is a slotted protocol and an improvement to the UD protocol. This protocol works fine during the low to moderate request rates and requires same bandwidth as in NPB during high request rates. When the customers want to view the video at any instant of time, their STB sends a request to

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
C_1	--	S_1	--	--	S_1	S_1	--	--	--
C_2	--	S_3	S_2	S_3	S_2	S_2	S_3	S_2	S_3
C_3	---	S_5	S_6	S_7	S_4	S_5	S_6	S_7	S_4

Figure 2. Segment-to-slot mapping of UD protocol.

the video server and the server transmits the schedules starting at the next slot. Though this protocol achieves minimum bandwidth by scheduling each segment on demand according to its minimum frequency, there may be heavy segment scheduling burden on the server. This protocol performs better than NPB or UD for broadcasting a single video, but for multiple hot videos, it requires more channels similar to FB, UD or NPB protocol and the segment scheduling is a complex task for the server.

3. Our protocol

The UD protocol is the reactive one that works fine at the low request rates but out performs at high request rates as compared to the NPB protocol. However the DHB protocol which is not based on any of the existing broadcasting schemes, suggests how to achieve minimum bandwidth by scheduling each segment on demand. It performs very well for the low as well as the high request rates. But the disadvantage of this protocol is its slotted technique. Each time, whenever any customer requests a video, their STB sends a request message to the video server and then the server prepares the transmission schedules starting at the next slot. This technique adds an extra burden to the server to schedule the transmission for each request. Moreover, it is not feasible at all for multiple videos on demand. It is observed that neither the UD nor the DHB protocols are suitable to handle multiple videos with minimum channel requirements at any request rates. Also, there is no such protocol which proposes for multiple videos, that achieves at the minimum bandwidth and less number of channels. From the practical point of view, different customers may request different hot videos. We can't guarantee that all customers may request only one hot video as choice varies from person to person.

So we propose here one protocol that efficiently distributes any number of hot videos and achieves the minimum bandwidth at low to moderate request rates and requires less number of channels as compared to UD protocol. Our protocol can be summarized as follows to achieve the following objectives.

- To handle requests, for multiple hot videos.
- To save bandwidth, by allocating less number of channels when request comes for different hot videos.
- To implement reactive approach, in order to distribute multiple hot videos over a large customer base.

The basic idea behind our protocol is same as that of FB and UD protocols taking an existing proactive protocol and transmitting it into reactive protocol by broadcasting required segments on demand. As similar to UD protocol, we have not chosen NPB protocol due to its precise segment-to-slot mapping which shows poor performance at low to moderate request arrival rates. Moreover, the segment-to-slot mapping is easier to manage in FB protocol. So, our broadcasting scheme is based on the FB protocol considering the segment sharing approach.

Let there are m-different videos denoted as V_1, V_2, V_3, ... , and V_m. The duration of each video is D and the consumption rate is 'b' megabits/second. (For MPEG-encoded videos b is approximately 1.5). So size of each video is $S = D * b$. Each video is equally divided in

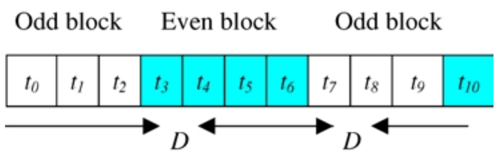

Odd block
Even block
Odd block
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
D
D

Figure 3. Partition of the duration D of the video into odd and even blocks.

to N-segments. So length of each slot for each video is $d = D/N$. Let S_i^x denote the i-th segment of the x-th video. Then the concatenation ($\bullet$) of all the segments give size of the x-th video i.e. $S^x = S_1^x \bullet S_2^x \bullet S_3^x \bullet \cdots \bullet S_N^x$ where x denotes the video number, e.g. $x = 1, 2, 3, \ldots, m$ and S^1, S^2, $S^3, \ldots,$ and S^m are size of each video and all are of equal duration D.

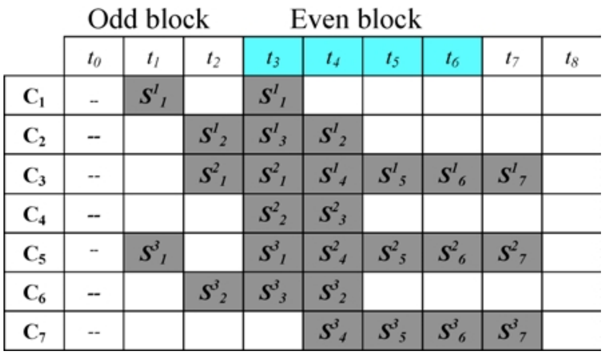
There are k channels $C_1, C_2, C_3, \ldots,$ and C_k each having bandwidth b and can be assigned to any video. We use FB protocol to distribute segments in different channels. So, each video will be divided into $(2^k - 1)$ segments which can be distributed in different channels based on our algorithm.

In our scheme, we divide the whole broadcasting period into certain *odd* and *even blocks* and each *block* into certain *slots*. The duration D of any video is equal to the duration of one *odd* and one *even* block. Also we divide each *block* into certain *slots*. If a video of duration D is divided into N-segments which can be allocated in k-channels, as our protocol, each odd block contains $(2^{k-1} - 1)$ slots and even block contains (2^{k-1}) slots. In figure 3, duration D of a video contains 7 segments and those 7 segments are partitioned into *odd* and *even blocks*. Each *slot* can be termed as a segment of duration 'd'. Based on the above rule, we have taken 3 *slots* in the *odd block* and 4 *slots* in the *even block* as shown in the figure 3.

In each *odd block* we need $(m * k - m + 1)$ channels irrespective of any concurrent or asynchronous request for any video, where m is the total number of videos to be broadcast and k is the number of channels allocated for each video based on FB protocol. Similarly, in each *even block* we need $(m * k)$ channels irrespective of any concurrent or asynchronous request for any video.

For example, at any instant of the *odd block*, if customers request for 4 different videos and we allocate 3 channels for each video in the previous protocols, then our protocol needs $(m * k - m + 1)$ channels which is equal to 9 channels as $m = 4$ and $k = 3$ here. Similarly, for the request at any instant of the *even block*, our protocol needs $(m * k)$ channels which is equal to 12 as $m = 4$ and $k = 3$ here. Generally 12 numbers of channels are required in any previous protocols like FB, DHB and UD to broadcast 4 different videos with 7 segments each.

In order to understand the scheduling of segments under the process of *block* and *slot* division, let us consider an example in which we have to broadcast three different hot videos as shown in the figure 4. We assume that all videos are of equal duration D. Let all the videos be divided into 7 segments each. In the FB *protocol*, we require 3 different channels to broadcast each video and all the segments starting from S_1 to S_7 of each video can be scheduled within these 3 channels. In our protocol, the duration D of each video can be divided into *odd block* comprising 3 *slots* namely t_0, t_1, t_2 and *even block* containing 4 *slots* t_3, t_4, t_5 and t_6. Thus the whole broadcasting scheme can be divided into *odd* and *even blocks* along with their respective *slots* as shown in the figure 4.



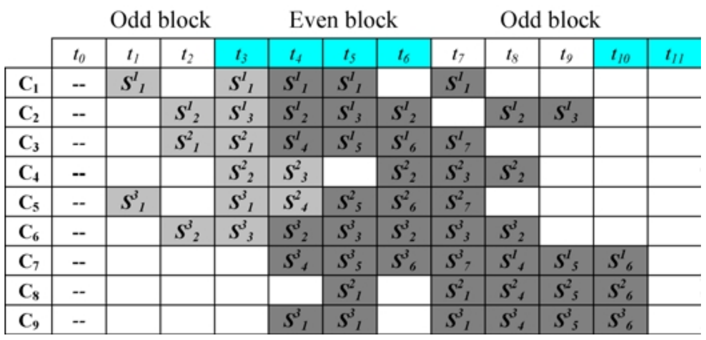
		Odd block			Even block				
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
C_1	-	S^1_1		S^1_1					
C_2	--		S^1_2	S^1_3	S^1_2				
C_3	--		S^2_1	S^2_1	S^1_4	S^1_5	S^1_6	S^1_7	
C_4	--			S^2_2	S^2_3				
C_5	-	S^3_1		S^3_1	S^2_4	S^2_5	S^2_6	S^2_7	
C_6	--		S^3_2	S^3_3	S^3_2				
C_7	--				S^3_4	S^3_5	S^3_6	S^3_7	

Figure 4. Segment-to-slot mapping during odd block of our protocol based on UD protocol.

Since, we want to broadcast 3 different videos, here number of videos, $m = 3$ and number of channels required for each video, $k = 3$. So, our protocol needs maximum $(3 * 3 - 3 + 1) = 7$ channels during *odd block* and maximum $3 * 3 = 9$ channels during *even block*. Within the first *odd block* there is probability of requesting any video concurrently or asynchronously at any *slot* t_0, t_1 or t_2. Let at slot t_0, video-1 (V_1) and video-3 (V_3) are requested concurrently. Then first segment (S^1_1) of V_1 will be allocated in C_1 and rest segments S^1_2 to S^1_7 will be allocated in the channels C_2 and C_3 in a step by step method as shown in the figure 4. Similarly the first segment (S^3_1) of V_3 will be allocated in C_5 and rest segments S^3_2 to S^3_7 of V_3 will be allocated in channels C_6 and C_7 as shown in the figure 4.

In the same way, let us suppose that a request comes for video-2 (V_2) at *slot* t_1, then the first segment (S^2_1) of V_2 can be allocated in C_3 and rest segments S^2_2 to S^2_7 will be allocated in channel C_4 and C_5. If the request comes at *slot* t_2 for all the three videos concurrently, then the first segment of V_1, V_2 and V_3 i.e., S^1_1, S^2_1, and S^3_1 will be allocated in channels C_1, C_3 and C_5, respectively. The rest segments of the first video V_1 i.e. S^1_2 to S^1_7 will be allocated in channels C_2 and C_3. Similarly the rest segments of video V_2 i.e. S^2_2 to S^2_7 will be allocated in channels C_4 and C_5 and the rest segments of video V_3 i.e. S^3_2 to S^3_7 will be allocated in channels C_6 and C_7. Segment to slot mapping of all the requested videos has clearly been shown in figure 4. The segments which are scheduled in different channels due to the request at different slots for different hot videos during an odd block have been represented in deep gray color.

Let us discuss now the segment-to-slot mapping in case of an *even block* for various request rates as shown in figure 5. This *block* contains *slots* t_3, t_4, t_5 and t_6 which continue just after *slot* t_2 of odd block. Let, the first and third videos be requested at *slot* t_3. The first segment S^1_1 of V_1 will be allocated in channels C_1 and the first segment S^3_1 of V_3 will be allocated in an additional channel C_9. The second segments of both videos have already been allocated due to the requests in the odd blocks. So the third segments of both videos will be allocated as usually in channels C_2 and C_6 for the videos V_1 and V_3, respectively. Since the segments S_4 to S_7 of video V_1 and V_3 have already been allocated in channels



	Odd block			Even block				Odd block				
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
C_1	--	S^1_1		S^1_1	S^1_1	S^1_1		S^1_1				
C_2	--		S^1_2	S^1_3	S^1_2	S^1_3	S^1_2		S^1_2	S^1_3		
C_3	--		S^2_1	S^2_1	S^1_4	S^1_5	S^1_6	S^1_7				
C_4	--			S^2_2	S^2_3		S^2_2	S^2_3	S^2_2			
C_5	--	S^3_1		S^3_1	S^2_4	S^2_5	S^2_6	S^2_7				
C_6	--		S^3_2	S^3_3	S^3_2	S^3_3	S^3_2	S^3_3	S^3_2			
C_7	--				S^3_4	S^3_5	S^3_6	S^3_7	S^1_4	S^1_5	S^1_6	
C_8	--					S^2_1		S^2_1	S^2_4	S^2_5	S^2_6	
C_9	--				S^3_1	S^3_1		S^3_1	S^3_4	S^3_5	S^3_6	

Figure 5. Segment-to-slot mapping during even block of our protocol based on UD protocol.

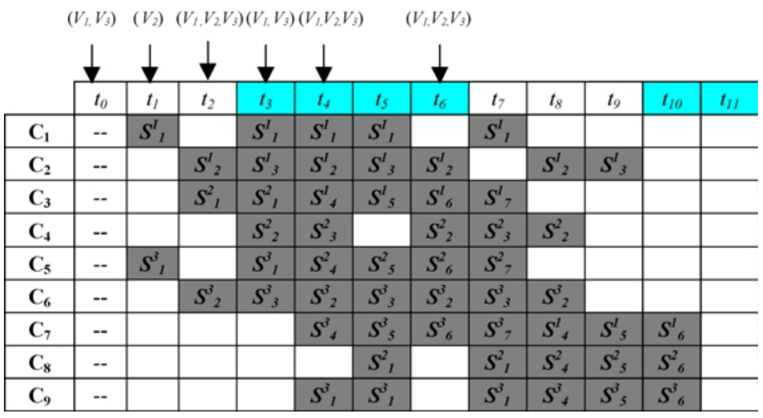
C_3 and C_7 respectively due to the request during the odd block, there is no need of any additional channels to broadcast those segments. Let all the three videos are requested at *slot* t_4. The first segment of video V_1 will be allocated in channel C_1, whereas the first segments S^2_1 and S^3_1 of videos V_2 and V_3 will be allocated in the additional channels C_8 and C_9 respectively. The allocation of segments S_2 and S_3 of all the three videos have been based on the previous continuity. The segment S_4 of all the three videos can be allocated in channels C_7, C_8, and C_9 for the videos V_1, V_2 and V_3 respectively. The rest segments S_5 to S_7 of all the three videos are already in buffer due to the requests during odd block. So these segments should not be allocated in channels.

If all the three videos are requested at *slot* t_6, S_1 of each video will be allocated in channels C_1, C_8, and C_9 for the videos 1, 2 and 3, respectively. The segments S_4 to S_6 of each video will be allocated in channels C_7, C_8, and C_9 for the videos 1, 2 and 3, respectively. The segment S_7 of all the three videos are already in the buffer, as a result of which, all the requests can be handled efficiently. In figure 5, the segments which are scheduled in different channels due to the request at different slots for different hot videos during an even block have been represented in deep gray color and the segments which have already been scheduled due to the request during odd block have been shown in light gray color.

In the figures 4 and 5, though we have taken the requests for different videos at different *slots* arbitrarily, it can work fine for any form of requests at any *slot*. Combining the requests for different videos at different slots for both *odd and even blocks*, we have shown the segment-to-slot mappings in figure 6. It clearly indicates that server needs only 7 channels during *odd block* whereas 9 channels during *even block* in order to broadcast three different hot videos.

4. Transmitting and receiving algorithms

In this section we have designed the server broadcasting and client receiving algorithms. In our protocol, we have divided the duration of each video D in to certain *blocks* and *slots* which can be summarized as follows:



	(V_1, V_3)	(V_2)	(V_1, V_2, V_3)	(V_1, V_3)	(V_1, V_2, V_3)		(V_1, V_2, V_3)					
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
C_1	--	S^1_1		S^1_1	S^1_1	S^1_1		S^1_1				
C_2	--		S^1_2	S^1_3	S^1_2	S^1_3	S^1_2		S^1_2	S^1_3		
C_3	--		S^2_1	S^2_1	S^1_4	S^1_5	S^1_6	S^1_7				
C_4	--			S^2_2	S^2_3		S^2_2	S^2_3	S^2_2			
C_5	--	S^3_1		S^3_1	S^2_4	S^2_5	S^2_6	S^2_7				
C_6	--		S^3_2	S^3_3	S^3_2	S^3_3	S^3_2	S^3_3	S^3_2			
C_7	--				S^3_4	S^3_5	S^3_6	S^3_7	S^1_4	S^1_5	S^1_6	
C_8	--					S^2_1		S^2_1	S^2_4	S^2_5	S^2_6	
C_9	--				S^3_1	S^3_1		S^3_1	S^3_4	S^3_5	S^3_6	

Figure 6. Segment-to-slot mapping during both odd and even blocks of our protocol based on UD protocol.

- Let us broadcast m-different hot videos, $V_1, V_2, \ldots,$ and V_m.
- Each video has duration D and divided into N equal segments. So $d = D/N$ is the length of each segment for each video.
- Each video $V_1, V_2, \ldots,$ and V_m is partitioned into $(2^k - 1)$ segments of equal duration d and these segments will be allocated in k-logical channels.
- The duration D of each video is divided into *odd* and *even blocks*.
- Each *odd* or *even block* or the *block* as a whole is divided into *slots*. The length of each segment is equal to magnitude of each *slot.*
- Each *odd block* contains $(2^{k-1} - 1)$ *slots* and each *even block* contains (2^{k-1}) *slots*, where k is the number of channels allocated to each video according to *FB protocol*.
- In our protocol, we need $(m * k - m + 1)$ number of channels during each *odd block* and $(m * k)$ channels during each *even block*.

4.1. *Server broadcasting algorithm*

4.1.1. Channel allocation algorithm

Step 1: Initialize m, the total number of videos to be distributed and initialize k, the number of channels required for each video.
Step 2: Set a timer in the server to fix the duration of each odd block $= d * (2^{k-1} - 1)$ and duration of each even block $= d * (2^{k-1})$.
Step 3: If requests arrive:

(a) During the odd block for xth video, the server allocates channels $C_{1+(x-1)(k-1)}$, $C_{2+(x-1)(k-1)}, \ldots, C_{1+x*(k-1)}$ to xth video.
(b) During even block for the first video, the server allocates channels $C_1, C_2, \ldots,$ C_k and $C_{m*k-m+1}$ to the first video.

For the xth video [x is not equal to 1], the server allocates channels $C_{2+(x-1)(k-1)}$, $C_{3+(x-1)(k-1)}, \ldots, C_{1+x*(k-1)}$ and $C_{m*k-m+x}$ to xth video.

4.1.2. Segment scheduling algorithm

Step 1: Initialize m, the total number of videos to be distributed and initialize k, the number of channels required for each video and $N = (2^k - 1)$.

Step 2: Set a timer in the server to fix the duration of each odd block $= d * (2^{k-1} - 1)$ and duration of each even block $= d * (2^{k-1})$.

Step 3: Initialize first odd block $A = 0$ and each time increment A when the next odd block starts. Similarly initialize first even block $B = 0$ and each time increment B when the next even block starts.

Step 4: Partition each video into $(2^k - 1)$ segments of equal duration d and these $(2^k - 1)$ segments of each video will be grouped into maximum k-logical channels.

Step 5: If the requests arrive at slot i:

(a) During odd blocks for the video x, then schedule the segments S_1^x in channel $C_{1+(x-1)(k-1)}$ in slot $(i + 1)$ and schedule the segments S_2^{xk-1} to S_{2-1}^{xk} in channel $C_{1+x*(k-1)}$ starting from slot $(A * N + 2^{k-1})$ to $(A * N + 2^k - 1)$ of xth video.

(b) During the even blocks, for the video x:

If request comes for the video $x = 1$, server schedule the segment S_1^1 in C_1 in slot $(i + 1)$.

Schedule the segments S_2^{1k-1} to S_{2-2}^{1k} in channel $C_{m*k-m+1}$ from slot $(B * N + 2^k)$ to $(B * N + 3 * 2^{k-1} - 2)$ and the segment S_{2-1}^{1k} in channel C_k in slot $(B * N + 2^k - 1)$.

If request comes for the video $x > 1$, schedule the segments S_2^{xk-1} to S_{2-2}^{xk} in channel $C_{m*k-m+x}$ from slot $(B * N + 2^k)$ to $(B * N + 3 * 2^{k-1} - 2)$ and the segment S_{2-1}^{xk} in channel $C_{1+x*(k-1)}$ in slot $(B * N + 2^k - 1)$ of video x.

Step 6: When requests arrive during slot i for the video x, the server first finds the current segment scheduling channel by channel for $[1 + (x - 1)(k - 1)] < j < [1 + x * (k - 1)]$:

(a) Assign the segments S_2^{xj-1} to S_{2-1}^{xj} of video x in channel C_j.

(b) If the last scheduled segment transmission in channel C_j is before slot $(i + 2^{j-1})$, then $(i + 2^{j-1})$, becomes the new start slot in channel C_j for video x.

(c) If the segment S_l^x of channel j, has not been scheduled for video x in slots greater than i, then the server will schedule a new transmission S_l^x in slot $(i + l)$ in channels C_j.

4.2. Client receiving algorithm

To receive the segments of any requested hot video, without any burden at clients end, we propose to set a *Hot Video Chart* (HVC) and a set top box (STB) with each client. The client has to select one of the hot videos given in the HVC and accordingly enjoy the requested video. The STB is used to buffer the premature segments when the client starts to see the requested video. The client/customer may select either the first video V_1 or any one of the video $V_2, V_3, \ldots,$ and V_m from the HVC and sends the request to the server. The server schedules the segments as the *server scheduling algorithm* which is based on the request of the client at a particular slot and a block. In order to minimize the complexity at clients end, the server sends the list of channels from which STB can download the segments. Accordingly, we have designed the algorithm to expedite the process to download segments for different videos from different channels.

Algorithm:

```
  **if** *Selection* $= V_1$
    *Download* $(2^k - 1)$ segments from the channels $(C_1, C_2, C_3, \ldots, C_{k-1})$ and $(C_k$ or $C_{m*k-m+1})$.
  **end if**
  **if** *Selection* $= V_x\ (x <> 1)$
    **for** $x := 2$ to $m$
      *Download* $(2^k - 1)$ segments from the channels $(C_{1+(x-1)(k-1)}$ or $C_{m*k-m+x})$ and $(C_{2+(x-1)(k-1)}, C_{3+(x-1)(k-1)}, \ldots, C_{1+x*(k-1)})$
    **end for loop**
**end if**
```

5. Analysis and comparisons

In our protocol, we have proposed to distribute multiple videos on demand at different request rates based on a reactive approach. We have analyzed how to reduce the number of channels at all request rates and to minimize bandwidth during low to moderate request rates for a particular video. As similar to UD protocol, our protocol requires less bandwidth and saves more channels than UD and FB protocols.

From the practical point of view, any subscriber may want to see different hot videos at different time in a day. Protocols like FB, UD and DHB, requires $(m * k)$ channels always in order to broadcast m different hot videos assigning k-channels to each one. But in our protocol, we need $(m * k - m + 1)$ channels during *odd block* and $(m * k)$ channels during each *even block*. So we save $(m - 1)$ channels in each *block* as compared to all previous protocols. Moreover we need same bandwidth that is totally identical to UD protocol.

In our study, figure 7 compares the channel requirements of our protocol for ten hot videos with 127 segments each with channel requirements of UD, FB and our protocol. The request arrival rates have been expressed in arrivals per hour and dedicated channels are expressed in multiples of the requested videos. It is to be noted that our protocol is based on reactive approach as similar to UD protocol. As the bandwidth requirement of our protocol is equal to the UD protocol, we have not compared the bandwidth with different arrival rates for UD and FB protocols.

From the simulation result for 10 different hot videos as shown in figure 7, we find that our protocol requires less number of dedicated channels than UD and FB protocols

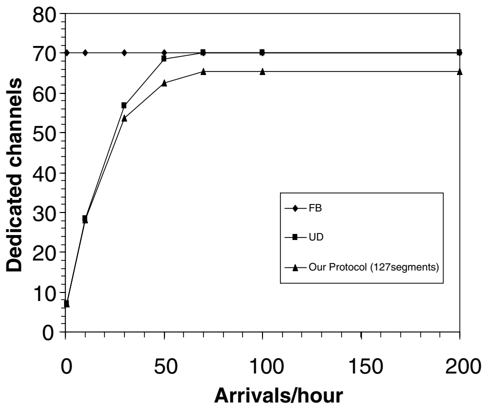


Figure 7. Compared channel requirements of FB, UD and our protocols with 127 segments for 10 hot videos.

for different request rates. Albeit, our protocol performs better at low to moderate request rates which is totally feasible for multiple hot videos with fixed number of subscribers. We assume to distribute always hot videos having equally likely demands on each of them. So we may expect moderate request rates for a particular video as all the videos are hot one and numbers of customers are fixed in any metropolitan *VOD* network; thereby our protocol saves more bandwidth as compared to FB and NPB protocols and requires less number of channels than UD and FB protocols.

For example, let there be 50 different hot videos to be distributed and each having duration 2 hours. Suppose all the videos are partitioned into 127 segments which can be broadcast in 7 different channels as *FB or UD protocol*. For 50 different videos, all other protocols need $50 * 7 = 350$ different channels, where as we need $(50 * 7 + 1 - 50) = 301$ channels and we save 49 channels in each *odd block*. It can be analyzed that we save 14% channels in each *odd block*. In general, in each *odd block*, we save $(m - 1)$ *channels and at an average* $(m - 1)/(m * k) = (1/k)(1 - 1/m)$ channels which is nearly equals to $1/k$ channels. We find that, the more number of hot videos we broadcast, the more number of channels can be saved which is highly useful from the commercial point of view.

Suppose we want to distribute m number of hot videos and we partition each video into 127 segments. In our protocol, duration of each video contains 63 *slots* in an *odd block* and 64 *slots* in an *even block*. So during an *odd block*, any number of concurrent or independent requests can be processed up to 63 *slots*, there by up to $63 * 0.95 = 1$ hour, we can save $(m - 1)$ channels [as 0.95 minute is the duration of each *slot*]. However, for any 2 hours video, we need $(m * k - m + 1)$ channels up to 50% of the duration of a video, and $(m * k)$ channels for the rest 50%. As per FB or UD protocol, if we allocate 7 channels to each 50 different hot videos, we need 301 channels up to 1 hour and in the next 1 hour if request comes, we need 350 channels. So in an average we need 325.5 channels throughout the video distribution period but both UD and FB protocols need 350 channels to distribute same number of videos.

We use reactive approach which can perfectly handle 100 or more popular videos over a very large customer base. As shown in figure 8, we require always less number of channels

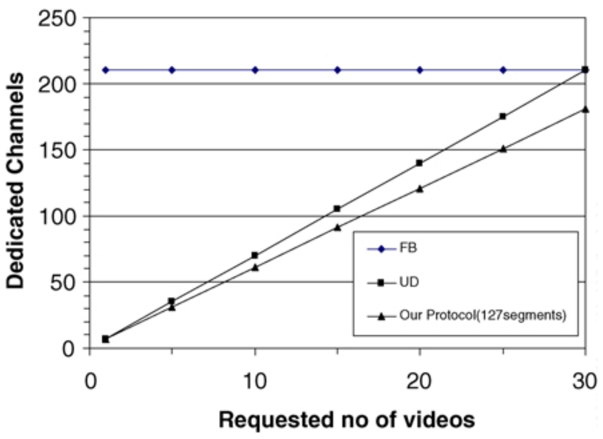


Figure 8. Compared channel requirements of FB, UD and our protocols with 127 segments for consecutive hot videos request rates.

than UD and FB protocol for consecutive requests of the hot videos. Moreover we require same bandwidth as in UD protocol and reduce the number of dedicated channels. In the given figures, though we have not compared our protocol with NPB protocol, but it has observed that our protocol out performs than FB and NPB protocols for low or moderate request rates on a particular hot video which may be possible for multiple hot videos on demand. Though we have referred to the partition technique of *FB protocol* and reactive approach of UD protocol, our segment-to-slot mapping is totally different from them. An interesting finding led us to implement the idea behind our protocol in NPB protocol to save both channel and bandwidth which we haven't discussed here to avoid the complexity.

6. Conclusions

Several protocols of reactive or proactive nature have been proposed to distribute videos on demand that can be implemented to handle different hot videos. But those protocols need channels that are multiple to the number of videos to be broadcast. In contrary to this, when we broadcast multiple videos, our protocol saves more channels and requires less bandwidth during low to moderate request rates as similar to the UD protocol. Also we save channels that are proportional to the number of distributed hot videos. Since the request rates may be equally likely for all the hot videos, our protocol can be commercially feasible to distribute multiple hot videos with a fixed customer base.

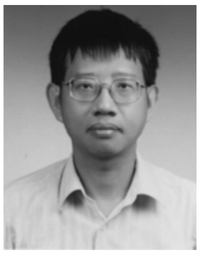
References

1. S.R. Carter, J.-F. Paris, S. Mohan, and D.D.E. Long, "A dynamic heuristic broadcasting protocol for video-on-demand," in IEEE Conference on Distributed Computing System, April 2001, pp. 657–664.
2. A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel allocation under batching and VCR control in video-on-demand systems," Journal of Parallel and Distributed Computing, Vol. 30, No. 2, pp. 168–179, 1994.

3. A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," Multimedia Systems, Vol. 4, No. 3, pp. 112–121, 1996.
4. D.L. Eager and M.K. Venon, "Dynamic skyscraper broadcasting for video-on-demand," in Proc. 4th Int. Workshop on Advances in Multimedia Information Systems, Sept. 1998, pp. 18–32.
5. L. Golubchik, J. Lui, and R. Muntz, "Adaptive Piggybacking: A novel technique for data sharing in video-on-demand storage servers," Multimedia Systems, Vol. 4, No. 3, pp. 140–155, 1996.
6. K.A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in Proc. ACM SIGCOMM'97 Conference, Sept. 1997, pp. 89–100.
7. L.-S. Juhn and L.-M. Tseng, "Fast broadcasting for hot video access," RTCSA'97, Oct. 1997, pp. 237–243.
8. L. Juhn and L. Tseng, "Fast data broadcasting and receiving schemes for popular video service," IEEE Trans. on Broadcasting, Vol. 44, No. 1, pp. 100–105, 1998.
9. J.-F. Paris, "A simple low-bandwidth broadcasting protocol for video on demand," in Proc. ICCCN'99 Conference, Oct. 1999, pp. 690–697.
10. J.-F. Paris, F.S.W. Carter, and D.D.E. Long, "A universal distribution protocol for video-on-demand," in Proc. Int. Conf. on Multimedia and Expo 2000, July 2000, pp. 657–664.
11. J.-F. Paris, D.D.E. Long, and P.E. Mantey, "A zero delay broadcasting protocol for video on demand," in Proc. 1999 ACM Multimedia Conf., Nov. 1999, pp. 189–197.
12. S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand services using pyramid broadcasting," Multimedia Systems, Vol. 4, No. 4, pp. 197–208, 1996.
13. J.W. Wong, "Broadcast delivery," in Proc. of the IEEE, Vol. 76, No. 12, pp. 1566–1577, 1998.

Prasan Kumar Sahoo received B.Sc. degree in Physics and M.Sc. degree in Mathematics from the Utkal University, Bhubaneswar, India, M.Tech degree in Computer Science from the Indian Institute of Technology, Kharagpur and Ph.D. degree in Applied Mathematics from Utkal University, India taking Advisor from IIT, Kharagpur, India.

He joined in the Software Research Center, National Central University, Taiwan in 2001 and his research interests include Ad-hoc wireless, WLAN and Bluetooth.



Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from the National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively.

He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from August 1997 to July 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing, parallel processing, and distributed computing Systems.

He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from August 1, 1996 to July 31, 2000. He was an associate editor of Journal of Information Science and Engineering from August 1, 1996 to July 31, 2002. He is an associate editor of Journal of the Chinese Institute of Engineers. He was a Guest Editor of Special Issue for Wireless Communications and Mobil Computing Journal. He was a Program Chair of IEEE ICPADS'2002. He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998.

Dr. Sheu is a senior member of the IEEE, a member of the ACM and Phi Tau Phi Society.