Communication and Network, 2010, 2, 79-85
doi:10.4236/cn.2010.21013 Published Online February 2010 (http://www.scirp.org/journal/cn)

Scientific Research

Live Video Services Using Fast Broadcasting Scheme

Satish Chand
Computer Engineering Division, Netaji Subhas Institute of Technology, Dwarka, New Delhi, India
E-mail: *schand86@hotmail.com*
Received November 17, 2009; *accepted December* 29, 2009

Abstract: The Fast Broadcasting scheme is one of the simplest schemes that provide video services. In this scheme, the video is divided into equal-sized segments depending upon the bandwidth allocated by the video server. If the video length is not known, then this scheme cannot be applied as the number of video segments cannot be determined. In a live video wherein the video size is unknown, especially the ending time of the live broadcast, e.g., cricket match, this scheme cannot be applied. In this paper, we propose a model that helps the Fast Broadcasting scheme to support live video broadcasting. The basic architecture of the system consists of a live system with one video channel that broadcasts the live video and a video server that broadcasts the already broadcast live video to users.

Keywords: fast broadcasting scheme, live video channel, channel transition

1. Introduction

Video-on-Demand (VOD) services are one of the important classes that has several potential applications, such as entertainment, advertisement, distance education, etc. In spite of vast usability, these applications could not gain much attention because the earlier network technologies were not sufficient enough to support high data rate and same was with the storage technologies. These technologies have been developed significantly and are further being enhanced. New applications are also being explored that again put high demand on these resources. Therefore, efficient utilization of the resources especially the bandwidth is must. Several schemes exist in literature, but all are meant for the stored videos. These schemes require the video length to construct its segments. If the video length is not known in advance, which is generally the case for live videos, these schemes can be applied. To develop a broadcasting scheme for live videos is the motivation of this work. In this paper, we propose a mechanism that helps the Fast Broadcasting scheme to support the live video services. The Fast Broadcasting scheme is one of the simplest schemes that provides the video services. For broadcasting a video, there is a need to have a video server to transmit the video data. Designing a video server has many issues, e.g., efficient system design [1–3], storage management [4–7], broadcasting techniques. The broadcasting techniques are very important as they help utilizing the bandwidth efficiently. Some of the important broadcasting techniques are discussed in [8–9].

The basic model in this paper consists of a system (we call it as a live system) with a video channel that is called as the live channel. This system is active for the duration of the live video and broadcasts the live video data only once using its channel. There is another system that stores the video data from the live channel in its buffer and then broadcasts. This system simultaneously stores the new data from the live channel into its buffer and broadcasts the already stored data by using its channels. This process continues for the duration of the live video transmission. When the live video is over, the data is broadcast by the video server only. If the live video broadcast is still there and the video server has no free channel to broadcast the newly downloaded data, then the last channel of the video server is made free by transferring its data to other channels and the last channel can broadcast the newly downloaded data. This mechanism is called the channel transition mechanism. The important issue in channel transition is that the current as well as future users should get continuous data delivery. In literature, there does not seem to appear any work that discusses live video broadcasting. It is perhaps that the broadcasting schemes existing in literature require the video in terms of prespecified number of segments depending on the allocated bandwidth. To determine the number of video segments of a live video is not possible because the video length is not known. In this paper, we develop a mechanism so that the Fast Broadcasting scheme can support the live videos. The remaining of the paper is organized as follows. Section 2 reviews the previous work. Section 3 proposes a system model to sup-

port the live video transmission. The main concept in this paper is channel transition, which is of two types intermediate and final channel transition. In intermediate channel transition the live video is not over, whereas in the final channel transition the live video is over. Section 4 presents the results and finally Section 5 concludes the paper.

2. Previous Work

The broadcasting schemes are an important class of protocols supporting the video-on-demand services. Some schemes use a segment as the basic data unit for transmission and a video channel a basic transmission unit. Some schemes further divide the segments into subsegments and the video channels into subchannels. A subchannel transmits all the subsegments of a particular single segment. Taking subsegment as a basic data unit and subchannel as a basic transmission unit reduce the resources requirement. This however increases the complexity. Some of the important broadcasting schemes include harmonic scheme [11], geometrico-harmonic scheme with continuous delivery [12], Pyramid Broadcasting scheme [13], Fast Broadcasting scheme [14]. The harmonic and geometrico-harmonic schemes have their basic data and transmission units as subsegments and subchannels, respectively. The pyramid and Fast Broadcasting schemes employ the segments and video channels as their basic data and transmission units, respectively. A video channel is a logical channel that has bandwidth equal to the consumption rate of the video. The Fast Broadcasting scheme is one of the simplest schemes for providing the video services. In this scheme, the bandwidth is equally-divided into channels, each having bandwidth equal to the video viewing rate. The video is also uniformly divided into segments. The first video channel transmits the first segment, repeatedly, and the second video channel transmits the second & third segments, alternately and periodically. The *i*th video channel transmits 2^{i-1} segments from $S_{2^{i-1}}$ to S_{2^i-1}, sequentially and periodically. The segment size determines the user's waiting time. The video transmission time is also divided into fixed time units, called time slots. In a time slot, a segment can be exactly viewed at the viewing rate. The number of segments of the video of length D for allocating K video channels is 2^K-1. Figure 1 shows transmission of the video segments over four video channels, denoted by C_1, C_2, C_3, and C_4, in the Fast Broadcasting scheme.

3. Live Fast Broadcasting Scheme

In live video broadcasting, the video size is not known in the beginning and hence its number of segments cannot be determined. The Fast Broadcasting scheme needs the number of segments in advance, so this scheme cannot

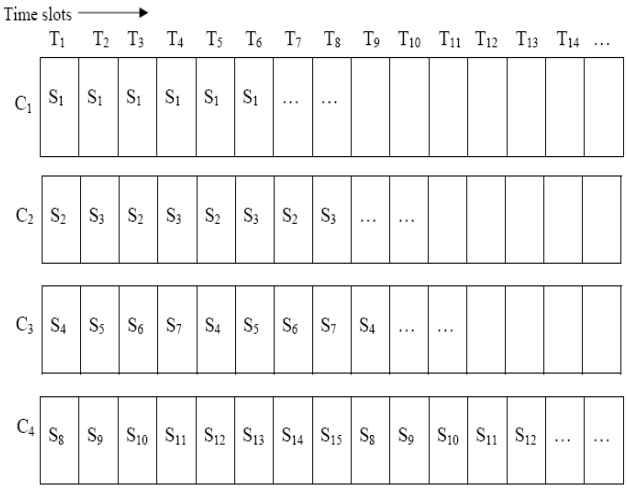

Figure 1. Data transmission in Fast Broadcasting scheme

be applied. In order to use this scheme for live video transmission, we make a simple modification in its ar chitecture. We transmit N number of segments using K video channels, which are related by

$$N = 2^K - 2 \quad (1)$$

The basic system model for supporting the live videos consists of a system, called the live system. This system broadcasts the live video by using its video channel (live channel). There is another system; we call it as the video server. The video server downloads the data from the live channel into its buffer in terms of fixed time durations, which are time slots. The data stored in a time slot is referred to as a video segment. The video server performs two activities. It stores new segments from the live channel into its buffer and simultaneously broadcasts the already stored segments. This server supports the video data to any user request, which misses the initial portion of the live video. A user request received after the live video has been started gets future data from the live channel and the initial missing data from the video server. The video server is always tuned to the live channel for new segments to store into its buffer. The stored segments are broadcast by the video server according to the following broadcasting procedure.

3.1 Broadcasting Procedure

The number of video segments for allocating K video channels is 2^K - 2. Denote the video segments by S_i ($i=1,2,..,N$). The first video channel transmits the first segment S_1, repeatedly, and the second video channel transmits the segments S_2 and S_3, alternately and repeatedly. The *i*th video channel transmits the segments $S_{2^{i-1}}$, $S_{2^{i-1}+1}$, $S_{2^{i-1}+2}$, …, S_{2^i-1}, sequentially and periodically, provided this *i*th channel is not the last video

channel. The last video channel, i.e., *K*th video channel transmits the segments $S_{2^{K-1}}$, $S_{2^{K-1}+1}$, $S_{2^{K-1}+2}$, …, S_{2^K-2}, sequentially and periodically.

The video server is always connected to the live channel for downloading the new segments. When a segment has been downloaded, the video server broadcasts that segment by using its channels along with other segments. This process continues till there is a free channel with the video server and the live video transmission is going on. Since the bandwidth allocated to the video by the video server is of fixed amount, this will get exhausted at some point of time. In that case, the video server would not be able to broadcast the newly downloaded segments. One solution to this problem is to increase the bandwidth, which may not always be possible. A better solution is to make the last channel free by transferring its segments to other video channels so that this video channel can broadcast the new segments. This process is called channel transition. The important issue while doing a channel transition is that all (present or future) user requests must get timely video data delivery. There are two types of channel transition: intermediate channel transition in which the live video is not over and the final channel transition in which the live video is over. We first discuss the intermediate channel transition.

3.2 Intermediate Channel Transition

The intermediate channel transition is performed when the video server is downloading new segments from the live system, but it does not have a free channel to broadcast them. We transmit (2^K-2) segments using K channels, not (2^K-1) as required in the Fast Broadcasting scheme. This is done because the capacity of the last channel is equal to total capacity of all other channels. Here the 'capacity' of a channel refers to the maximum number of segments transmitted by that channel. While transferring segments of the last video channel to other channels to make it free, we simply need to double the segments' size. If $S_1, S_2,\ldots,S_k,\ldots$ are the old segments, then the new segments, denoted by $S^1_1, S^1_2,\ldots$, are given by $S^1_1 = S_1 + S_2$, $S^1_2 = S_3 + S_4,\ldots$, $S^1_k = S_{2k-1} + S_{2k},\ldots$, and so forth. After a channel transition the segment size (i.e., new waiting time) becomes twice of the old one (old waiting time). To avoid the increase in the waiting time, we should delay the channel transition as much as possible, which can be delayed till all channels have their capacity full. It is not difficult to show that by delaying a channel transition maximally, all user requests (before and after the channel transition) get the video data in time.

Figure 2 shows the first channel transition at thick black line for allocating four video channels to the video by the video server. The gray-colored channel signifies the live channel and remaining are the video server's channels. The problem of data availability may be for a request that begins viewing the video just before the channel transition. This request, denoted by R_{13} in Figure 2, begins downloading the data from the time slot TS_{14}. This request R_{13} downloads the segments S_{15} onward from the live channel and S_1 to S_{14} from the video server. The segments available for downloading from the video server using the 1st, 2nd, 3rd, and 4th video channels in the time slot TS_{14} are S_1, S_2, S_6, and S_{14}, respectively. The segment S_1 can be viewed while downloading from the video server and does not requires storage space. Just

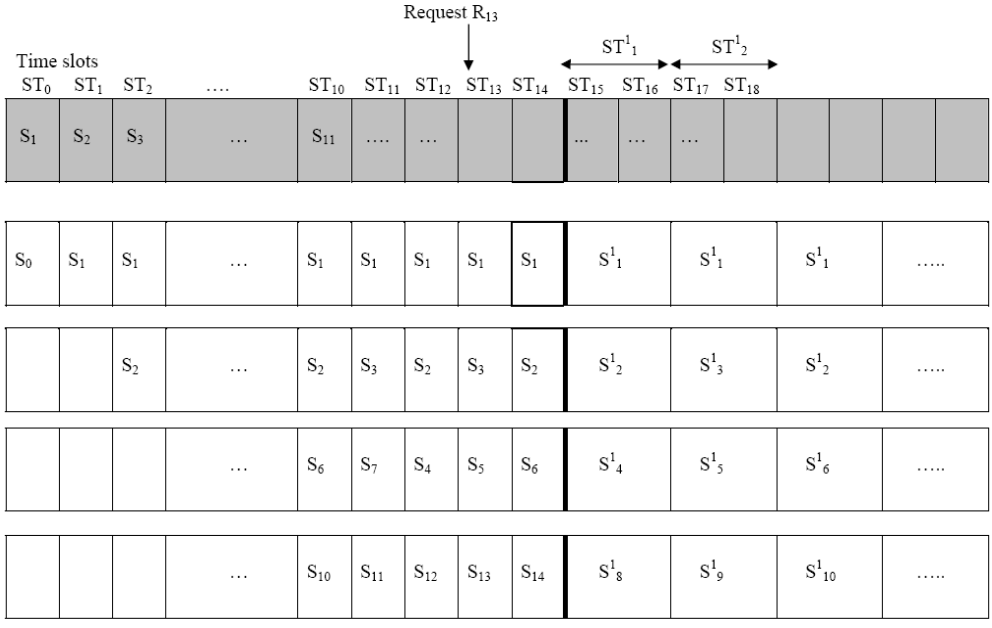

Figure 2. First channel transition

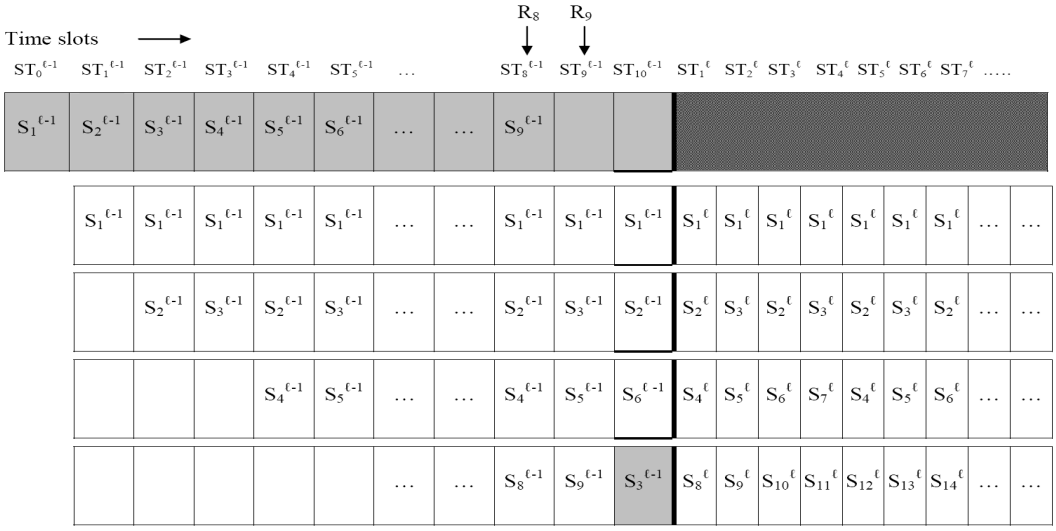

Figure 3. Final channel transition

after the channel transition, R_{13} needs the segment S_2 for viewing and it is already in its buffer because it had been stored just before the channel transition. After viewing S_2, R_{13} needs S_3 segment which is the first half part of the second segment S^1_2 (=S_3 + S_4) transmitted by the second channel after channel transition. Thus, the data of the segment S_3 can be made available to R_{13}. Using similar discussions, we can show that a request received in any time slot can receive timely video data. We now discuss the final channel transition.

3.3 Final Channel Transition

In order to carry out the final channel transition, the video size must be known and it is possible only when the live video transmission is over. Since the video size is known, we can construct the video segments. The live video can be over at any point of time. If the data broadcast by the live channel does not form a complete segment, we can add dummy data to make it a complete segment. The live video can be over in any time slot, which means that the last video channel can have any number of segments ranging from one to its maximum capacity. If its capacity is full, then nothing is required to do. To carry out the final channel transition, the last channel must have at least one segment and at least one empty time slot. To occupy empty time slots with the video data, we need to increase the number of segment to $(2^K - 2)$ so that all time slots of all the video channels K are occupied. Increasing the number of segments decreases the segment size as the video size is of fixed length. The first channel C_1 transmits $S_1^{\ell-1}$ and the second channel C_2 transmits the segments $S_2^{\ell-1}$ and $S_3^{\ell-1}$ just before the final transition. Here the superscript ℓ of a segment (time slot) signifies the segment (time slot) after the final (last) channel transition and $(\ell\text{-}1)$ for a segment (time slot) just before the final channel transition. After the final channel transition, the segment size decreases, i.e., some last portion of $S_1^{\ell-1}$ segment is added to the beginning of $S_2^{\ell-1}$ and some last portion $S_2^{\ell-1}$ is added to the beginning of $S_3^{\ell-1}$, and so forth. The number of new segments is $(2^K - 2)$ and they can occupy all the time slots on all channels. While carrying out this process, timely video data delivery to the current as well as future requests must be ensured.

We now discuss how the video data delivery can be timely maintained to all user requests by taking an example. Consider that the video server has allocated four video channels to the video. The last (i.e., fourth) video channel can accommodate the segments S_8, S_9,…, S_{14}. The live video can be over in any time slot $ST_i^{\ell-1}$ (*7<i<14*). Let *i=8*, i.e., the last video channel has to transmit the last segment as $S_9^{\ell-1}$ in the time slot $ST_8^{\ell-1}$. The segment $S_9^{\ell-1}$ will be available at the video server for transmission in the time slot $ST_9^{\ell-1}$ (refer Figure 3). We can carry out the channel transition immediately after the time slot $ST_9^{\ell-1}$. The request R_8 that begins to download the data from the time slot $ST_9^{\ell-1}$ onward can download, if required, the segments $S_1^{\ell-1}$, $S_3^{\ell-1}$, $S_5^{\ell-1}$, and $S_9^{\ell-1}$ in that time slot (refer Figure 3). The segment $S_1^{\ell-1}$ is viewed while downloading in the time slot $ST_9^{\ell-1}$ and in the next time slot (i.e., after channel transition) this request needs $S_2^{\ell-1}$. The segment $S_2^{\ell-1}$ is distributed among the segments S_2^{ℓ} and S_3^{ℓ} after the channel transition. The segment S_2^{ℓ} is available for downloading just after the channel transition, but the initial portion of segment S_2^{ℓ} comprises some last portion of $S_1^{\ell-1}$ and the remaining is some initial portion of the segment $S_2^{\ell-1}$. It means that just after channel transition the initial portion of S_2^{ℓ} that

is from the segment $S_1^{\ell-1}$ will be available, not the segment $S_2^{\ell-1}$. Thus, the request R_8 will not get the data of the segment $S_2^{\ell-1}$ in time. To circumvent this problem, we delay the final channel transition one time slot after the live video is over. In the instance case, we carry out final channel transition at the end of the time slot $ST_{10}^{\ell-1}$, not $ST_9^{\ell-1}$. By doing so, we have one empty time slot (shown as gray-colored on the last video channel in Figure 3) on the last video channel that can be used to broadcast $S_2^{\ell-1}$ or $S_3^{\ell-1}$ depending upon the last segment broadcast by the live channel is even or odd. The segments available for downloading to the request R_8 in the time slot $ST_{10}^{\ell-1}$ are $S_2^{\ell-1}$ and $S_6^{\ell-1}$. The request R_8 has segments $S_2^{\ell-1}$ and $S_3^{\ell-1}$ in its buffer and will need $S_4^{\ell-1}$ for viewing in the time slot $ST_{12}^{\ell-1}$, which is not in the buffer storage. After the channel transition, the segment $S_4^{\ell-1}$ is distributed among S_5^{ℓ}, S_6^{ℓ}, and S_7^{ℓ} and the time slot $ST_{12}^{\ell-1}$ is distributed in the time slots ST_2^{ℓ}, ST_3^{ℓ}, and ST_4^{ℓ}. The segment S_5^{ℓ} can be downloaded in the time slot ST_2^{ℓ} and the segments S_6^{ℓ} and S_7^{ℓ} in the time slots ST_3^{ℓ} and ST_4^{ℓ}, respectively. It may be noticed that the segment S_5^{ℓ} is available at the beginning of the time slot ST_2^{ℓ}, but some initial portion of ST_2^{ℓ} comprises the last portion of the time slot $ST_{11}^{\ell-1}$. Thus, the request R_8 can get the video data in time. Consider another request R_9 that begins downloading the video data from the time slot $ST_{10}^{\ell-1}$ onward and can download, if required, the segments $S_2^{\ell-1}$, $S_6^{\ell-1}$, and, $S_3^{\ell-1}$. The request R_9 requires the segment $S_4^{\ell-1}$ in the time slot $ST_{13}^{\ell-1}$. The segment $S_4^{\ell-1}$ is distributed among S_5^{ℓ}, S_6^{ℓ}, and S_7^{ℓ} and the time slot $ST_{13}^{\ell-1}$ becomes a part of the time slots ST_4^{ℓ} and ST_5^{ℓ}. The segments S_5^{ℓ}, S_6^{ℓ}, and S_7^{ℓ} can be downloaded in the time slots ST_2^{ℓ}, ST_3^{ℓ}, and ST_4^{ℓ}, respectively. Thus, the segment $S_4^{\ell-1}$ can be made available in time to request R_9. Using similar discussions, we can show that all requests can be provided the required data in time.

We now find out those segments S_i^{ℓ} that have the data of the segment $S_4^{\ell-1}$ after the final channel transition. The indices of the segments S_i^{ℓ} that have the data of segment $S_4^{\ell-1}$ are I_L, I_{L+1}, …, I_H, where I_L and I_H are given by $I_L = n_1$ such that $\min_{n_1}\left\lfloor \frac{n_1 * p}{N} \right\rfloor \geq 3$ and $I_H = n_2$ such that $\min_{n_2}\left\lfloor \frac{n_2 * p}{N} \right\rfloor \geq 4$ where p is the index of the last segment broadcast by the live channel and N is the number of segments that is given by (1).

For the request R_9 that receives the data from $ST_{10}^{\ell-1}$ time slot onward, assuming that the last segment broadcast by the live channel is $S_9^{\ell-1}$, the segment $S_4^{\ell-1}$ would be distributed among the segments S_5^{ℓ}, S_6^{ℓ}, and S_7^{ℓ} as $I_L = 5$ and $I_L = 7$ because for $n_1 = 5$, $\min_{n_1}\left\lfloor \frac{n_1 * 9}{14} \right\rfloor \geq 3$ and for $n_2 = 7$, $\min_{n_2}\left\lfloor \frac{n_2 * 9}{14} \right\rfloor \geq 4$ hold. This can easily be verified as follows.

$S_1^{\ell} = 9*S_1^{\ell-1}/14$; $S_2^{\ell} = 5*S_1^{\ell-1}/14 + 4*S_2^{\ell-1}/14$;

$S_3^{\ell} = 9*S_2^{\ell-1}/14$; $S_4^{\ell} = S_2^{\ell-1}/14 + 8*S_3^{\ell-1}/14$;

$S_5^{\ell} = 6*S_3^{\ell-1}/14 + 3*\mathbf{S_4^{\ell-1}}/14$; $S_6^{\ell} = 9*\mathbf{S_4^{\ell-1}}/14$; $S_7^{\ell} = 2*\mathbf{S_4^{\ell-1}}/14 + 7*S_5^{\ell-1}/14$.

The video channels allocated by the video server to the video transmit new segments S_i^{ℓ} (i=1,2,..,N) in the new time slots ST_i^{ℓ}(i=1,2,…,N,…) according to the broadcasting procedure. Thus, we have discussed channel transition. In next Section, we discuss the results.

4. Results

The Fast broadcasting scheme is one of the simplest broadcasting schemes. That's why we have considered it for live video broadcasting. In its architecture, we have made a simple modification so that the number of segments transmitted by its last video channel is equal to the total segments transmitted by its remaining channels. This modification makes the final channel transition at the optimal time point. In other words, the channel transition is performed after all time slots of all the video channels are full with the video segments. Consider again Figure 2. The request R_0 arrived in the 0^{th} time slot ST_0 begins downloading the segments S_2 onward from the live channel and S_1 from the video server. The buffer requirement for this request is one segment. The request R_1 arrived in the 1^{th} time slot ST_1 begins downloading the segments S_3 onward from the live channel into its buffer and S_1 and S_2 from the video server. Its buffer requirement is of two segments. We describe the buffer computation for an arbitrary request. Consider an arbitrary request, say, R_9, that arrives in the 9^{th} time slot ST_9. This request begins downloading the data from the time slot ST_{10} onward. The segments S_{11} onward are downloaded from the live channel and the segments S_1 to S_{10} from the video server. The segments available for downloading, actually downloaded, and that required for viewing, in different time slots are shown in Table 1.

In last column '+' and '-' signs signify that the segment is stored in and read from the buffer, e.g., +2-1+1=2 means 2 segments are stored from the video server into the user's buffer, one is read from the user's buffer, and one is stored from the live channel into the user's buffer. Thus, the net buffer requirement is of two segments. The segment S_1 is viewed while downloading.

Using similar discussions, we can compute the buffer requirement for any request. Table 2 shows the buffer requirement for different requests (referring Figure 2), assuming that four video channels have been allocated to the video by the video server.

In Table 2, for the requests R_7, R_8, R_9, R_{10}, and R_{11}, there are two different storage requirements. The first requirement is one when the live video is going on. The second requirement refers to one when the live video is over after the 14^{th} segment. The maximum user's waiting

Table 1. Segments stored from the live channel and the video server by request R_9

Time slot	Segments available for storing from Video Server	Segments stored from Video Server	Segments stored from live channel	Segment required for viewing	Total segments required
ST_{10}	$S_1+S_2+S_6+S_{10}$	S_2	S_{11}	S_1	+1+1=2
ST_{11}	S_3+S_7	S_3	S_{12}	S_2	+1-1+1=1
ST_{12}	S_4	S_4	S_{13}	S_3	+1-1+1=1
ST_{13}	S_5	S_5	S_{14}	S_4	+1-1+1=1
ST_{14}	S_6	S_6	S_{15}	S_5	+1-1+1=1
ST_{15}	$S_4^1/2= S_7$	S_7	S_{16}	S_6	+1-1+1=1
ST_{16}	$S_4^1/2= S_8$	S_8	S_{17}	S_7	+1-1+1=1
ST_{17}	$S_5^1/2= S_9$	S_9	S_{18}	S_8	+1-1+1=1
ST_{18}	$S_5^1/2= S_{10}$	S_{10}	S_{19}	S_9	+1-1+1=1
Buffer Storage required for request R_9 = 10S					

Table 2. Buffer Requirement for different Requests allocating four video channels

Request	Buffer Requirement	Request	Buffer Requirement
R_0	S	R_6	7S
R_1	2S	R_7	8S or 7S
R_2	3S	R_8	9S or 5S
R_3	4S	R_9	10S or 5S
R_4	5S	R_{10}	11S or 5S
R_5	6S	R_{11}	12S or 5S

time in this scheme is pre-decided for the initial users and remains the same till the channel transition time. After every channel transition except the final one the user's waiting time becomes double of the previous one. The final channel transition is done only when the live video transmission is over. After the final channel transition the user's waiting time is stabilized and the scenario becomes like a stored video. The size of a segment (time slot) after a channel transition except the final one becomes double. If S_i, ST_i and S^1_i, ST^1_i are the *i*th segments and time slots, respectively, before and after the first channel transition (assuming four video channels), then we have the following relations:

$S^1_i = S_{14+2i-1} + S_{14+2i}$

$ST^1_i = ST_{14+2i-1} + ST_{14+2i}$

In general, we have

$$S^k_i = S^{k-1}_{14+2i-1} + S^{k-1}_{14+2i}, \text{ for } k = 1, 2, \ldots, \ell\text{-}1, \quad (2)$$

$$ST^k_i = ST^{k-1}_{14+2i-1} + ST^{k-1}_{14+2i}, \text{ for } k = 1, 2, \ldots, \ell\text{-}1, \quad (3)$$

where S^0_i, ST^0_i denote the very first *i*th segment and time slot, respectively, i.e., $S^0_i = S_i$, $ST^0_i = ST_i$ and ℓ denotes the final channel transition.

We can determine the size of a segment (time slot) after any channel transition in terms of the original segments (time slots). For example, consider the third channel transition (assuming it is not the final channel transition). Then, (2a) & (2b) give

$$S^3_i = S^2_{14+2i-1} + S^2_{14+2i}. \quad (4)$$

$$ST^3_i = ST^2_{14+2i-1} + ST^2_{14+2i}. \quad (5)$$

We can take *i=1* because after any channel transition all the segments (time slots) are of same size. We will derive the relation for the segments only, and for the time slots exactly the same relation will hold. For *i=3*, we have from (3a) as

$S^3_1 = S^2_{15} + S^2_{16}$

We need to find out S^2_{15} and S^2_{16}, which are given by

$S^2_{15} = S^1_{14+29} + S^1_{14+30} = S^1_{43} + S^1_{44}$

$S^2_{16} = S^1_{14+31} + S^1_{14+32} = S^1_{45} + S^1_{46}$.

Again we need to find out S^1_{43}, S^1_{44}, S^1_{45}, and S^1_{46}, which are given by

$S^1_{43} = S^0_{14+85} + S^0_{14+86} = S^0_{99} + S^0_{100}$

$S^1_{44} = S^0_{14+87} + S^0_{14+88} = S^0_{101} + S^0_{102}$

$S^1_{45} = S^0_{14+89} + S^0_{14+90} = S^0_{103} + S^0_{104}$

$S^1_{46} = S^0_{14+91} + S^0_{14+92} = S^0_{105} + S^0_{106}$

The segments S^0_is are the original segments. Thus, we have

$S^3_1 = S_{99} + S_{100} + \ldots + \ldots + S_{106}$

For the time slots, we have

$ST^3_1 = ST_{99} + ST_{100} + \ldots + \ldots + ST_{106}$

The segment (time slot) size after the final channel transition (i.e., ℓth) is Θ times of the segment (time slot) size of that of the (ℓ-*1*)th channel transition, where $0.5 < \Theta \leq 1$. Here $\Theta = 1$ signifies that the live video transmission is over when all time slots of all the video channels are full. In that case, nothing is done and the user's waiting time is unchanged. For $\Theta \neq 1$, the last channel after the final but one channel transition must have at least one

segment and at least one empty time slot. Consider that the last video channel has N_S segments after the final but one channel transition. After the final channel transition, the segment size would be $(2^{K-1} + N_S)*S^{\ell-1}/N$. For K=4, N=14, and N_S=2, the segment size after the final channel transition is $(2^3 + 2)*S^{\ell-1}/ 14=10*S^{\ell-1}/14$.

The user's waiting time changes after a channel transition depending upon the segment size and this is fixed for the stored videos for a given bandwidth. In the proposed scheme, the initial user's waiting time is decided by the service provider and it is also equal to the segment size. This decision is necessary for arranging the video data that would be downloaded in terms of the segments and made on the basis of bandwidth allocated to the video. The video size increases after a new segment from the live channel has been downloaded, but this change affects broadcasting only after the channel transition. After the final channel transition, the user's waiting time generally decreases. The basic requirement to carry out the final channel transition is that there must be at least one segment and at least one empty time slot on the last video channel.

5. Conclusions

In this paper, we have proposed a technique so that the Fast Broadcasting scheme can be used for broadcasting the live videos. The Fast Broadcasting scheme does not support the live video services in its original form because it requires the number of segments of the video beforehand and for that the video length should be known, which is generally not known in advance in case of the live videos. In this proposed scheme, the live video data is stored in buffer at the video server in terms of fixed time durations, called time slots. The data downloaded in a time slot is considered as a segment. The downloaded segments are broadcast by the video server till there is free channel available with the video server. When no free channel is available and live video is there, channel transition is required. In the proposed scheme, the channel transition can be delayed maximally, i.e., up to the point when all time slots of all the video channels have been completely occupied. This study may be useful for designing a VOD system that can support the live video, such as cricket match, interactive education session, etc.

REFERENCES

[1] K. M. Ho and K. T. Lo, "Design of a decentralized video-on-demand system with cooperative clients in multicast environment," Advances in Multimedia Information Processing, Lecture Notes Computer Science, 4810, pp. 401–404, 2007.

[2] Y. B. L. Jack, "Channel folding – an algorithm to improve efficiency of multicast video-on-demand systems," IEEE Transactions on Multimedia, Vol. 7, No. 2, pp. 366–378, 2005.

[3] W. F. Poon, K. T. Lo, and J. Feng, "A hierarchical video-on-demand system with double-rate batching," JVCIR, Vol. 16, No. 1, pp. 1–18, 2005.

[4] D. J. Gemmell, H. M. Vin, D. Kandlur, P. V. Rangan, and L. A. Rowe, "Multimedia storage servers: a tutorial," Computer, Vol. 28, No. 5, pp. 40–49, 1995.

[5] N. J. Sarhan and C. R. Das, "Caching and scheduling in nad based multimedia servers," IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 10, pp. 921–933, 2004.

[6] A. L. Chervnak, D. A. Patterson and R. H. Katz, "Choosing the best storage system for video service," In Proceeding of third ACM international conference on Multimedia, San Francisco, USA, pp. 109–119, 1995.

[7] A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," IBM Watson Research Center, RC 19347, 1994.

[8] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation based pyramid broadcasting scheme for video on demand systems," In Proceeding International Conference on Multimedia Computing and Systems, pp. 118–126, 1996.

[9] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," In Proceeding of NOSSDAV'98, pp. 183–194, 1998.

[10] A. L. N. Reddy, J. Wyllie, and K. B. R. Wijayratne, "Disk scheduling in a multimedia I/O system," ACM Transactions on Computer Communication and Application (TOMCCAP), Vol. 1, No. 1, pp. 37–59, 2005.

[11] L. S. Juhn and L. M. Tseng, "Harmonic broadcasting scheme for video-on-demand service," IEEE Transactions on Consumer Electronics, Vol. 43, No. 3, pp. 268–271, 1997.

[12] H. Om and S. Chand, "Geometrico-harmonic broadcasting scheme with continuous redundancy," IEEE Transactions on Multimedia, Vol. 9, No. 1, pp. 410–419, 2007.

[13] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," ACM Multimedia System, Vol. 4, No. 4, pp. 197–208, 1996.

[14] L. S. Juhn and L. M. Tseng, "Fast data broadcasting and receiving scheme for popular video service," IEEE Transactions on Broadcasting, Vol. 44, No. 1, pp. 100–105, 1998.

