An Active Buffer Management Technique for Providing Interactive Functions in Broadcast Video-on-Demand Systems

Zongming Fei, *Member, IEEE,* Mostafa H. Ammar, *Fellow, IEEE,* Ibrahim Kamel, *Member, IEEE,* and Sarit Mukherjee, *Member, IEEE*

***Abstract*— Multicast delivery is an efficient approach to the provision of a video-on-demand service. Interacting with the video stream is a desirable feature for users. However, it is a challenging task to provide the functionality in the multicast environment because a lot of users share multicast delivery channels. In this paper, we propose an active buffer management technique to provide interactive functions in broadcast video-on-demand systems. In our scheme, the client can selectively prefetch segments from broadcast channels based on the observation of the play point in its local buffer. The content of the buffer is adjusted in such a way that the relative position of the play point is kept in the middle part of the buffer. Our simulations show that the active buffer management scheme can implement interactive actions through buffering with a high probability in a wide range of user interaction levels.**

***Index Terms*— Multimedia streaming, video-on-demand, interactive functions.**

I. Introduction

A Video-on-Demand (VoD) service provides subscribers with a set of videos and sends a specific video to customers upon request. In unicast VoD systems, customers are serviced individually by dedicating a transmission channel and a set of server resources to each customer. This leads to an expensive-to-operate, non-scalable system. Using multicast to send popular videos has been demonstrated by many researchers to be an efficient way for their delivery [1], [2].

Two approaches to providing multicast video-on-demand services have been studied. One is *on-demand batching*, in which the server allocates a channel to a group of pending clients requesting the same video and sends it over that channel. These systems are *on-demand* in the sense that the server allocates a channel to send a video only if there are requests sent over upstream channels by the clients requesting the video.

The other approach is *Continuous broadcast*, in which the server allocates one or several channels to one video and each channel sends the whole video or a part of it in cycles. In these systems, the server keeps broadcasting videos even if there are no requests from clients. There are no upstream channels and the method is most suitable for broadcasting popular videos. Usually users may have to wait for a short period of time before they can start the playout of the video. A simple form of this approach is the pay-per-view model (or *staggered broadcast*) in which several channels broadcast a video periodically with staggered start times. In such a system, the maximum startup latency is equal to the length of the video divided by the number of channels allocated to it. More recently, *partitioned video broadcast* divides a video into segments and sends each segment over one channel in cycles [3]–[6]. In these schemes, clients have to be able to download at a higher rate than the playback rate, either from a channel sending at a higher rate, or from multiple channels sending at the same rate as the playback rate. The benefit is that they greatly reduce the startup latency while using the same number of multicast channels as the staggered broadcast.

The multicast approach to video-on-demand satisfies requests from several users at one time and thus, to some extent, may sacrifice special requirements of each individual user. The startup latency is a tradeoff of some degradation of service quality for scalability. The desirable interactive functions (also called *VCR functions* in this paper) for video-on-demand service are also difficult to provide in multicast systems. Some recent VoD work begins to deal with this problem [2], [7]–[10]. Most of the work has addressed VCR functions in the environment of an *on-demand batching* video delivery system and depends on both client buffering and "interactive" channels to provide VCR functions. The client side buffering technique can be extended to apply to the *continuous broadcast* model of video delivery, but interactive channels are not available in this model. Even if we have interactive channels, it is not preferable because using interactive channels to provide VCR functions compromises the scalability of the multicast paradigm. As an alternative to interactive channels in a continuous broadcast system, interactive functions that cannot be handled through client buffering can be accomplished by switching broadcast streams and incurring a possible discontinuity.

In this paper, we propose an *active buffer management* technique [11], [12] for providing VCR functions in continuous video broadcast systems. The active buffer management scheme uses client side buffering in a novel fashion that

Manuscript received December 11, 2001; revised January 29, 2003. This work was supported in part by the National Science Foundation under Grant CCR-0204304.

Zongming Fei is with the Department of Computer Science, University of Kentucky, Lexington, KY 40506 USA (e-mail:fei@cs.uky.edu).

Mostafa H. Ammar is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail:ammar@cc.gatech.edu).

Ibrahim Kamel is with the College of Information Systems, Zayed University, Dubai, UAE (e-mail:ibrahim.kamel@zu.ac.ae).

Sarit Mukherjee is with Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail:sarit@lucent.com).

relies on the simultaneous availability of "past", "present" and "future" parts of a video. The client can selectively prefetch segments from broadcast channels based on the current status of their buffers.

The challenge in the partitioned broadcast scheme is how to deal with the unequal sizes of the segments. After a VCR action moves the play point to a new position in the video stream and we resume normal playback, we may face a situation of discontinuity of playback because we cannot get the frames before their scheduled playback time due to the unequal segment sizes. To deal with this problem, we introduce the concept of *feasible points* so that we can adjust a client's destination to those points that can preserve the continuity of playback.

The rest of this paper is structured as follows. In Section II we describe the background and related work. Section III proposes an active buffer management technique and Section IV describes the scheme for providing VCR functions. Section V presents the performance evaluation and Section VI concludes the paper.

II. Background and Related Work

A. Partitioned Video Broadcast

Partitioned video broadcast divides a video into segments and uses one channel to send each segment in cycles [3]–[5]. The more channels we can get, the shorter the startup latency for a video. Assume the bandwidth allocated to a video is B Mbits/sec (usually a multiple of video playout rate). The bandwidth is divided into equal-bandwidth channels. The bandwidth of each channel is equal to the playback rate, b Mbits/sec. The number of channels is $K = \lfloor \frac{B}{b} \rfloor$.

A video of length L is divided into K segments and channel i periodically broadcasts segment i, where $1 \leq i \leq K$. The length of each segment is determined by a weight function $f(i)(1 \leq i \leq K)$. The size of segment i is $\frac{f(i)*L}{\sum_{j=1}^{K} f(j)}$. It is this weight function $f(i)$ that distinguishes one broadcasting scheme from another. The goal of these schemes is to minimize the startup latency. One way to achieve this is to make $f(i)$ increase as fast as possible. However, if function $f(i)$ increases too fast, i.e., the difference between sizes of segment i and $i+1$ is too large, the client may possibly not be able to get the start of segment $i+1$ when it finishes playing out segment i. Therefore, function $f(i)$ is subject to the continuity condition that guarantees that with a given number of loaders, clients can download each segment of the video in a timely fashion in order to playback without jitter. The function $f(i)$ generates a series specifying the relative weights of each segment.

The skyscraper broadcasting scheme [4] assumes that each client has two loaders (can download from two channels at the same time as the client plays back from the buffer) and the series generated by the function $f(i)$ is $1, 2, 2, 5, 5,\ 12, 12, 25, 25, \cdots$. There is an upper bound parameter W, which sets $f(i)$ to W if $f(i) > W$. The greedy disk-conserving broadcasting scheme [5] generalizes the skyscraper scheme to the case in which each client can have more than 2 loaders. A series generated by function $f(i)$ with $4b$ disk bandwidth requirement for each client is $1, 2, 4, 8, 14, 24, 40, 70, \cdots$. We analyze the problems encountered while we try to provide interactive functions in these schemes. Then we design a new broadcasting series that is more suitable for providing interactive functions.

B. VCR Actions

We consider a video as a continuous sequence of frames. The *play point* of a client is the frame currently being played out in the video. The *destination point* of a VCR action is the frame in the video where the client finishes the VCR action. A VCR action can be described as a pair $(\Delta t, \Delta \ell)$, where Δt is the time it takes for the VCR action to complete and $\Delta \ell$ is the relative position of the destination point with respect to the current play point (assume it is p). Obviously $\Delta t \geq 0$ and the destination is $p + \Delta \ell$. The destination is in the forward direction if $\Delta \ell > 0$ and in the backward direction if $\Delta \ell < 0$. We assume that the part of the video (of length $\Delta \ell$) is evenly distributed in the time interval Δt. Assuming the VCR action is issued at the play point p and the normal playback rate is b, we consider following VCR actions.

1. Jump Forward/Backward (JF/JB): $\Delta t = 0, \Delta \ell \neq 0$.
2. Fast Forward/Backward (FF/FB): $\Delta t > 0, \frac{|\Delta \ell|}{\Delta t} > b$.
3. Slow Forward/Backward (SF/SB): $\Delta t > 0, 0 < \frac{|\Delta \ell|}{\Delta t} < b$.
4. Pause: $\Delta t > 0, \Delta \ell = 0$.
5. Play/Play Backward: $\Delta t > 0, \frac{|\Delta \ell|}{\Delta t} = b$.

III. The Idea of Active Buffer Management

A. Conventional Client Buffering Scheme to Provide VCR Functions for Video Broadcast

Using client buffering is an intuitive solution to the problem of providing VCR functions [2], [9], [10]. However, conventional client buffering schemes lack the flexibility for providing interactive functions in broadcast VoD systems. We illustrate the conventional client buffering scheme in Figure 1 and focus on how the relative position of the play point in the buffer is changed by each VCR action.

The client buffer can be looked as a sliding window over a usable portion of a video. It is composed of video between the most recently downloaded frame and the oldest frame still held in the buffer. When a client begins to download a video, the buffer is empty. As the frames are downloaded from the broadcast channel, they are played out and the buffer is filled up at the same time. If there is no VCR action, the play point is the same as the most recent frame. When the buffer becomes full, the oldest frames are discarded to allow more recent frames to come in. In Figure 1, we assume the frame immediately before the oldest frame is N and the buffer is full. So the most recent frame is $N + 100$, which is also the frame currently being downloaded from the broadcast channel. If some VCR actions have been performed, then the play point can be frames other than the most recent frame. In the example, we assume the play point is $N + 60$. We show how each VCR action will change the relative position of the play point in the buffer. We can see that *play* does not change the relative position of the play point, *fast forward* and *jump forward* move the play point forward (called *forward VCR*

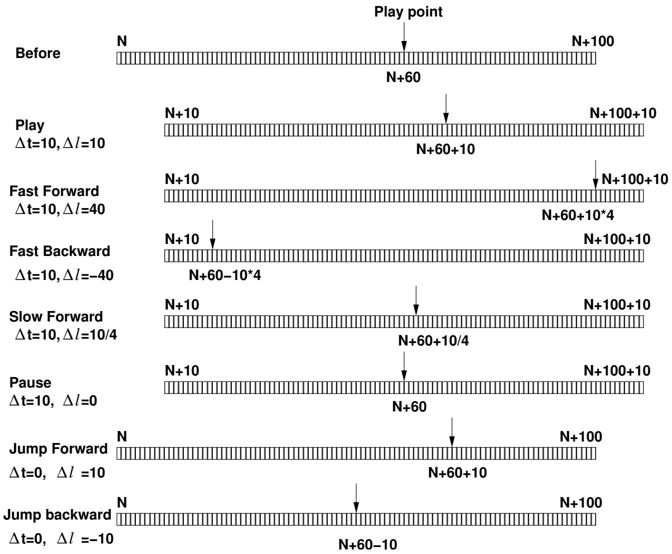

Fig. 1. The effects of VCR functions on play position

actions), and all other actions move the play point backward (called *backward VCR actions*).

Problems with the scheme: The problem is that the relative position of the play point is determined by the history of VCR actions. The effects of VCR actions in the same direction are cumulative. When consecutive VCR actions in the same direction are performed, the play point will ultimately move to a boundary of the buffer. At this time, VCR functions in this direction can no longer be implemented. It is at the mercy of VCR actions in the opposite direction to reverse the effects. The lack of self-adjustment is an inherent problem of the above buffer scheme.

B. Active Buffer Management

The probability that a VCR function can be implemented with a local buffer depends on many factors, including the buffer size, the play point within the buffer, buffer content and management. The key is how to make this probability high given a fixed buffer size. Our intuition is to keep the play point in the middle of the buffer so that there is a lower probability that VCR actions will move the play point out of the buffer. We use a *buffer manager* to adjust the content of the buffer after VCR actions by selectively downloading segments from the broadcasting channels. This is possible in partitioned video broadcast because all the segments are repeatedly broadcast in the channels ready for downloading and therefore, a video's "past", "present" and "future" are being broadcast simultaneously at any point in time.

We will explain the basic idea by a simplified example. Assume that the client buffer can hold three segments. At some point, the buffer holds segments $k, k+1, k+2$ and the play point is in the middle segment $k+1$. Now observe the situation after one segment time. 1) Case 1: If there is no VCR action, the play point will be in segment $k+2$. During this period, the loader finishes downloading segment $k+3$ and segment k is discarded. The buffer now holds segments $k+1, k+2, k+3$ and the play point is still in the middle segment (now $k+2$). 2) Case 2: If a *fast forward* action is issued during this period and moves the play point to a position within segment $k+3$. We observe the play point is no longer in the middle segment. The idea of active buffer management is to let the buffer manager select the segments to be downloaded based on the position of the play point. After observing that the play point is in the last segment $k+3$ of three segments $k+1, k+2, k+3$ in the buffer, the buffer manager will download segments $k+4, k+5$ at the same time. After one segment time, the buffer will hold segments $k+3, k+4, k+5$ and the play point will move to segment $k+4$. Therefore, the play point is moved back to the middle segment.

We see from this example that the buffer manager can compensate the effects of VCR actions. Therefore, our scheme can tolerate consecutive VCR actions in the same direction and solve the problem of conventional client buffering scheme [1].

C. Destination Adjustment for VCR Actions

An important and desired property of the destination of any VCR action is that *once the client resumes normal play from this destination, it should be able to play the rest of the video to the end without any interruption, if there are no further VCR actions.* We call all those destinations satisfying the above property *feasible points*.

Since we cannot predict the user behavior, there always exists a case in which the destination point of a user action is outside of the client buffer. Although we let the user specify an arbitrary destination for VCR actions, the algorithm adjusts it to the nearest feasible point automatically, if necessary [2]. This is what we called *discontinuous interactive function* implementation, for the reason of the discontinuity between the specified destination and the adjusted destination. By doing this, we can guarantee the continuity after we resume normal playback. An alternative way of implementation is to delay the VCR action until the buffer is filled with the content we want. However, the user has to wait before a VCR action can be implemented. We choose to adjust the destination under the assumption that the adjustment is usually not large and can be tolerated, as will be shown in the performance evaluation.

IV. Providing VCR Functions in Partitioned Video Broadcast

A. A VCR-oriented Broadcasting Series

The design of a new broadcasting series is motivated by the problems we encountered during the process we explored the possibility of providing VCR functions in the skyscraper scheme. The details can be found in [12]. We give a function

[1] If the probabilities of forward and backward VCR actions are skewed, we may prefer a different position. For example, if we have 90% of VCR actions in the forward direction, we will keep the relative position of the play point in the left part of the buffer.

[2] Three ways are possible to adjust the destination: 1) to the nearest feasible point *before* the destination; 2) to the nearest feasible point *after* the destination; 3) to the nearest feasible point, no matter in what directions. In our later discussion and simulations, we choose to move to the nearest feasible point *before* the destination, after considering a suggestion that automatic skipping may be less desirable than automatic replay.

that requires clients to be able to download from 3 channels at the same time.

$$f(i) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = 2 \\ 4 & \text{if } i = 3 \\ f(i-1) & \text{if } f(i-1) \neq f(i-2) \text{ and} \\ & \quad f(i-2) \neq f(i-3) \\ f(i-2) * 2 & \text{otherwise} \end{cases}$$

The closed form of the above function is $f(i) = 2^{i - \lceil \frac{i}{3} \rceil}$. It generates the following segment sizes: $1, 2, 4, 4, 8, 16, 16, 32, 64, 64, \cdots$. We use a parameter u to limit the maximum value of the series. We set $f(i) = f(i-1)$ if $i > u$. So all $f(i)$'s with $i \geq u$ are equal to $f(u)$. This restriction also limits the buffer requirements for clients.

To reduce the startup latency, we want the series to increase as fast as possible so that the size of the first segment will be small. Smooth playout by clients with a certain number of loaders is one constraint that prevents the series from increasing too fast. Providing VCR functions puts additional constraints on how fast the series can go. The intuitions behind designing the series in this way are to make sure that the broadcasting points in the channels can be used as the adjusted destination points of VCR actions and to guarantee that clients can download necessary frames in time for smooth playback after VCR actions.

The loading strategy for the clients is simple if there are no VCR actions. At the beginning three loaders are allocated to segments 1,2 and 3. Each will load the assigned segment as soon as it begins. The downloaded segments are put into the buffer. The display gets the frames from the buffer or from the channel at the same time when they are sent to the buffer. The playback of the video begins as soon as the loader begins downloading the first segment.

When a loader finishes loading segment j, it will be assigned to segment $j+3$. The loader downloads the segment when it is broadcast in the channel at the latest cycle before its playback time.

Our scheme satisfies the condition $\sum_{i=1}^{j-1} \geq f(j) - 1$ for any $j > 1$. We can also verify that our scheme satisfies the condition $f(j) \leq \sum_{i=j-m+2}^{j-1} f(i) + gcd(f(j), f(j-m+1))$ for all $j > m$, where $m = 4$. From the theorem 3.2 in [5], we know that to guarantee the continuity of playback, the client only needs the bandwidth of 4 times the display rate, including the I/O bandwidth the client needs to retrieve playback data from the disk to the display. We assume that the client can download from 3 channels simultaneously and, therefore, the bandwidth satisfies the requirement. So the continuity of playback is guaranteed if there are no VCR actions.

B. VCR Function Implementations with the Active Buffer Management Scheme

When clients download or playback video segments, they go through two phases. Segment i is in the pyramid phase (unequal segments) if $1 \leq i \leq u-1$, and in the equal segment phase if $u \leq i \leq K$.

We assume that the buffer size at the client (such as set-top box) is a constraint. The client is required to have three buffers with each buffer being of same size as the maximum segment and three loaders so that it can download from three channels at the same time. If the client has an unlimited buffer, backward VCR actions can all be handled. However, we still need to take care of the forward VCR actions, by using prefetching and dealing with the continuity issue after VCR actions.

We distinguish two components that work together to provide VCR functions. One is the *player*, which accepts user interaction commands and plays back the video as requested. The other is the *loader/buffer manager* (or simply manager), which decides which channels the client should listen to and download segments.

a) **Player:** The player keeps playing back the content from the buffers until it accepts a VCR command. It then makes the decision based on the availability of the content in the buffer and the broadcasting position of relevant channels.

When the player accepts a *Fast Forward/Fast Backward/Slow Backward/Play Backward* command, it checks whether the content to be played is in the buffer and whether the destination point is a feasible point at the finishing time. When a command passes these two tests, it can be implemented smoothly. Otherwise, it jumps to the nearest feasible point.

When the player accepts a *Jump Forward/Backward* command, it checks whether the destination is a feasible point. If it is, it moves to that point and resumes normal playback. Otherwise, the player selects a nearest feasible point. If the player accepts a *Pause or Slow Forward* command, it will implement them for the duration specified. This is because *Pause* or *Slow Forward* can be implemented smoothly from any feasible point.

After we complete a VCR action that has a destination in a segment different from the start point, the player checks the allocation of loaders/buffers and reallocates them, if necessary. Assume the segment containing the current play point is k. During the pyramid phase, if three segments $k, k+1, k+2$ are in the buffer or being downloaded, no actions will be taken. Otherwise, the player will reallocate loaders to these segments. During the equal segment phase, if segments $k-1, k, k+1$ are in the buffer or being downloaded, no actions will be taken. Otherwise, the player will reallocate loaders to these segments. Downloading begins immediately after the loaders are allocated to them.

b) **Loader/buffer manager:** It allocates loaders/buffers at the channel boundary, which is the time when the channel finishes one cycle of broadcasting the segment and begins the next cycle of broadcasting. Different channels may have different channel boundaries during the pyramid phase. When we finish downloading frames from a channel and reach its channel boundary, the loader allocated to it will be reclaimed by the manager, which will re-allocate the loader (and/or buffer) to download other segments. Assume the segment where the current play point is located is segment k [3]. Note the play point can be different from the downloading

[3]If the play point is located exactly at the boundary between segments k and $k+1$, we assume the current segment is k, rather than $k+1$.

point from the channel because of VCR actions and unequal segmentation. The manager allocates the loaders/buffers to segments relative to k.

During the pyramid phase, if any of the segments $k+1, k+2, k+3$ are not in the buffer and not being downloaded, we allocate loaders to them and downloading begins at their respective channel boundaries. When a client starts, we assume the current segment is 0. The manager allocates loaders to segments $k+1, k+2, k+3$, i.e., 1,2,3. Downloading of the first segment begins after we reach the boundary of channel 1. The time interval is the startup latency. Downloading of the second and the third segments may begin later, depending on channel positions. During the equal segment phase, if the play point is in the earlier half of the current segment k, the three loaders are assigned to segments $k-1, k, k+1$. If the play point is in the later half of the current segment k, the three loaders are assigned to segments $k, k+1, k+2$. If the content of a segment is in the buffer, no loading will occur. The downloaded segments will overwrite the earlier segments already in the buffer. However, if a VCR action is ongoing, all the buffer content for the action cannot be overwritten. If a loader is downloading a segment when we want to allocate a loader to the segment, no action is taken and we assume one loader is already allocated to the segment.

Our scheme has the flexibility for clients with more than three loaders and buffers. In this case, extra loaders/buffers are allocated to unallocated segments based on the priorities of these segments. We define the priority of a segment as its distance to the current segment. Smaller number means higher priority. If two segments have same priority number, the segment in the forward direction is given a higher priority.

The active buffer management scheme can also be extended to provide interactive functions in staggered broadcast VoD systems. The key problem is to select an appropriate channel to download the required segments. The details can be found in [11].

C. Feasible Points

We introduce the concept of feasible points to deal with the problems occurred in providing VCR functions in partitioned video broadcast in Section III-C. If we do not check the property of the destination point of a VCR action, we may face discontinuity of playback after we resume normal play following the VCR action, even if no further VCR actions are issued. We observe that even if all the video content required by a VCR action is in the buffer, performing it can still lead to the discontinuity in the future after we resume normal play [12].

Assume the start position and end position of segment i in a video (e.g., time from video starting point) are s_i and e_i, respectively. At any time, each channel is broadcasting a specific frame in the segment. It is called the channel point denoted as c_i. We have $s_i \leq c_i \leq e_i$ and $e_i = s_{i+1}$. We use $[y_1, y_2]$ to represent the part of video between two points y_1 and y_2. Assume the destination point $d(= p + \Delta\ell)$ is in segment j. Note that d can be different from c_j.

If the destination point d is located at or before the channel point c_j (i.e., $d \leq c_j$), the segment containing d is designated as the *target segment*. We use k to denote the target segment number and have $k = j$. We now give the rules for these destination points. Later we will consider the rules for destination point d located after the channel point c_j. Note these rules are given in the form of deciding whether a frame is a feasible point or not, but they can also be used to find the nearest feasible point with regard to a target frame. We distinguish three cases based on the relative sizes of segments involved, and subcases based on their alignment in the broadcast channels.

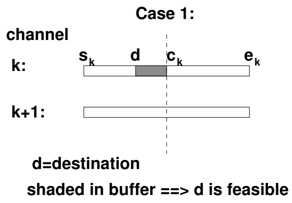

Fig. 2. Rules for feasible points (case 1)

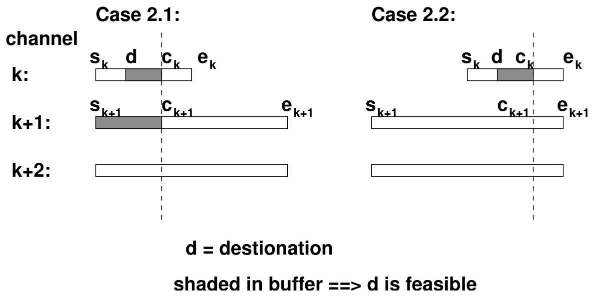

Fig. 3. Rules for feasible points (case 2)

- Case 1 (Figure 2):
  - If $[d, c_k]$ is in the buffer, then d is feasible.
- Case 2 (Figure 3):
  - case 2.1: If $[d, c_k]$ and $[s_{k+1}, c_{k+1}]$ are in the buffer, then d is feasible.
  - case 2.2: If $[d, c_k]$ are in the buffer, then d is feasible.
- Case 3 (Figure 4):
  - case 3.1: If $[d, c_k]$, $[s_{k+1}, c_{k+1}]$ and $[s_{k+2}, c_{k+2}]$ are in the buffer, then d is feasible.
  - case 3.2: If $[d, c_k]$, $[s_{k+1}, c_{k+1}]$ are in the buffer, then d is feasible.
  - cases 3.3 and 3.4: If $[d, c_k]$ are in the buffer, then d is feasible.

Now we consider the case in which the destination point d in segment j is located after the channel point c_j (i.e., $d > c_j$). If we designate segment $j+1$ as the target segment, we can use the above rules. If we still use k to denote the target segment number, then we have $k = j+1$. In this situation, $[d, c_k]$ means $[d, e_{k-1}]$ and $[s_k, c_k]$.

The correctness of these rules can be explained from two aspects. First, based on the broadcasting series we designed, there are only three different cases based on the relative sizes of three consecutive segments, as described in the above rules. Second, for each subcase, the requirement is a necessary and sufficient condition for the destination d to be feasible. Take

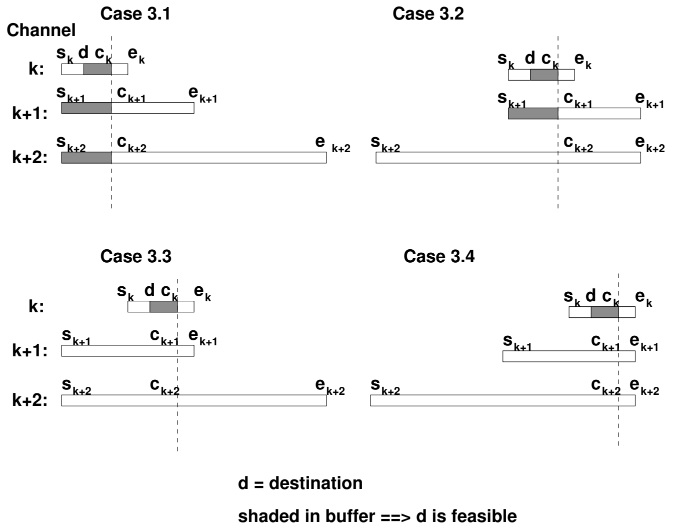

Fig. 4. Rules for feasible points (case 3)

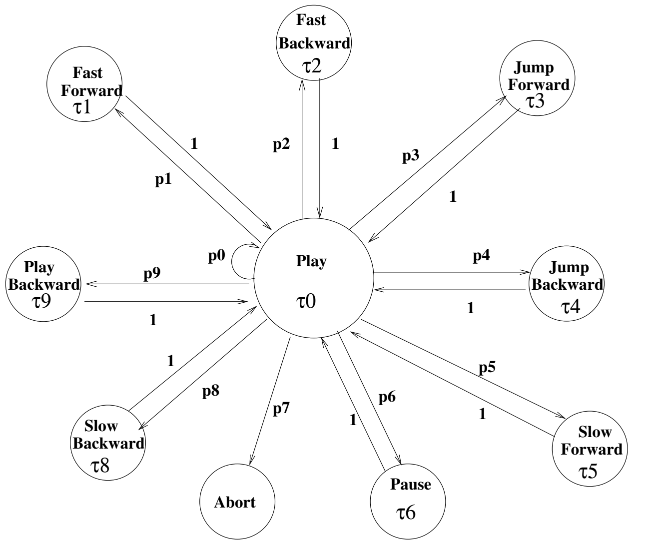

Fig. 5. User interaction model

case 3.2 as an example. If three segments are aligned as shown in case 3.2 in Figure 4 and $[d, c_k]$, $[s_{k+1}, c_{k+1}]$ are in the buffer, we can derive that d is feasible. We can let two loaders to download from channels k and $k+1$ at this time and they will fill in the buffer with $[c_k, e_k]$ and $[c_{k+1}, e_{k+1}]$. We can play $[d, e_k]$, and then $[s_{k+1}, e_{k+1}]$ without jitter. After time $[c_{k+1}, e_{k+1}]$, one loader will finish getting $[c_{k+1}, e_{k+1}]$ and it will be allocated to download segment $k+2$ from the start. So we can get the beginning of segment $k+2$ before its playout time. The segments greater than $k+1$ can also be downloaded before playout time. This shows that $[d, c_k]$, $[s_{k+1}, c_{k+1}]$ in the buffer is a sufficient condition for d to be feasible.

On the other hand, if either $[d, c_k]$ or $[s_{k+1}, c_{k+1}]$ is not in the buffer, we may have a problem if we start playing from d. First assume that $[d, c_k]$ is not in the buffer. Actually we need to play them immediately. However, they cannot be downloaded from broadcast channel k because the channel is broadcasting $[c_k, e_k]$ right now. So we will have to wait, thus discontinuity. Second assume that $[s_{k+1}, c_{k+1}]$ is not in the buffer. We need to play them after $[d, e_k]$ time, during which channel $k+1$ will be broadcasting part of $[c_{k+1}, e_{k+1}]$. Therefore, we cannot get $[s_{k+1}, c_{k+1}]$ when we need to play them. This shows that $[d, c_k]$, $[s_{k+1}, c_{k+1}]$ in the buffer is a necessary condition for d to be feasible.

V. Simulation Results

A. Experimental Settings

In our experiments, we assume the video length is 120 minutes and we have 30 channels for broadcasting the video. The video is divided into 30 segments. The first 8 segments are of unequal size with the weights defined by the series in Section IV-A. Segments from 9 to 30 are of equal size. The size of the first segment is 0.0805 minutes or 4.83 seconds. So the maximum startup latency is 4.83 seconds and the average is half of it, i.e., 2.42 seconds. The size of the largest segment is 5.15 minutes. The size of buffer required at the client side is a multiple (at least 3) of the size of the largest segment.

The user interaction model is shown in Figure 5, which gives the probability of issuing a specific VCR action and the mean duration. The duration of a VCR action is exponentially distributed with mean τ_i. The model is further simplified to reduce the number of experimental parameters. We set $p_7 = p_8 = p_9 = 0$. Let $p_v = \sum_{i=1}^{6} p_i$ and we have $p_0 = 1 - p_v$. Also we let $\tau_i = \tau_v$ for $1 \le i \le 6$. After the simplification, we have only three parameters p_v, τ_0 and τ_v. We call $\frac{\tau_v}{\tau_0}$ the *duration ratio*. We experiment with two ways to change the user interaction level.

1) *Change probability.* In this case, we let $\frac{\tau_v}{\tau_0} = 0.5$ and vary p_v from 0.1 to 0.9.

2) *Change duration ratio* $\frac{\tau_v}{\tau_0}$. In this case, we set $p_v = 0.6$ and change the duration ratio $\frac{\tau_v}{\tau_0}$ from 0.2 to 1.0.

B. Numerical Results

We use the percentage of VCR actions implemented by the content in the client buffer (abbreviated as *percentage of buffer VCR actions*) as a performance measure of our scheme. The higher this percentage, the better the interactive performance of the system. We also measure the difference between the specified destination and the adjusted destination for those VCR actions with destination changed for the feasibility reason. We use the *percentage of destination shift* (defined as the percentage of the difference over the length of VCR actions) as another performance measure. The smaller this percentage, the better the performance.

In our first experiment (Figure 6), we examine the effect of frequency of issuing VCR actions on the percentage of buffer VCR actions. We change the probability of issuing VCR actions (p_v) from 0.1 to 0.9. We set τ_0 to be about half of maximum segment size (2.58 minutes). As expected, the percentage of buffer VCR actions decreases as the probability of issuing VCR actions increases. However, the percentage of buffer VCR actions stays above 87% even if the probability of issuing VCR actions is as high as 0.9. Though this high interactive degree rarely happens, the simulation results in this

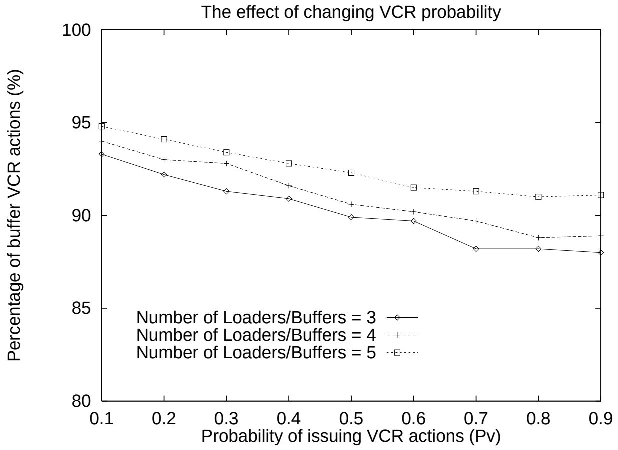

Fig. 6. Percentage of buffer VCR actions when changing probability

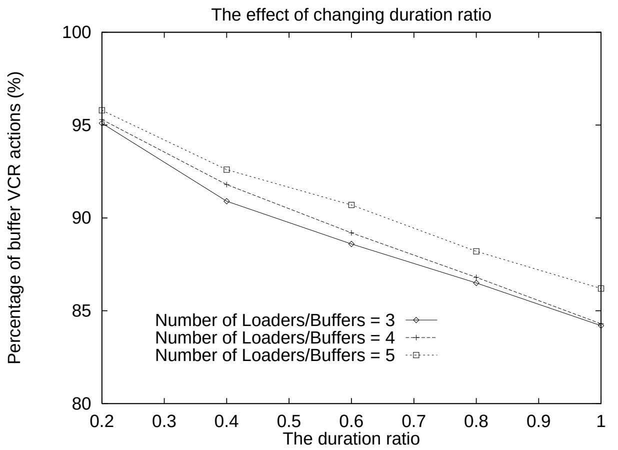

Fig. 7. Percentage of buffer VCR actions when changing duration ratio

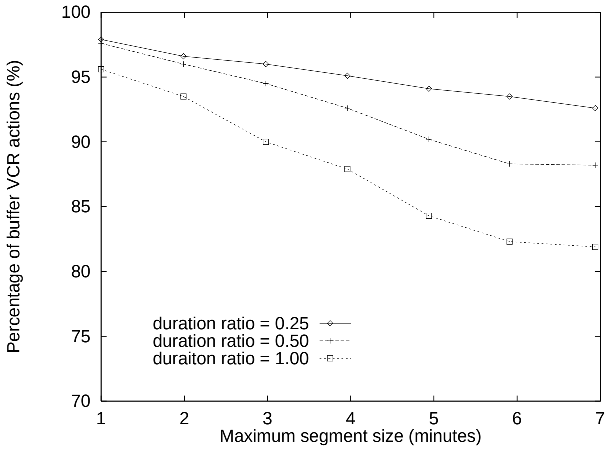

Fig. 8. The effect of maximum segment size on the percentage of buffer VCR actions

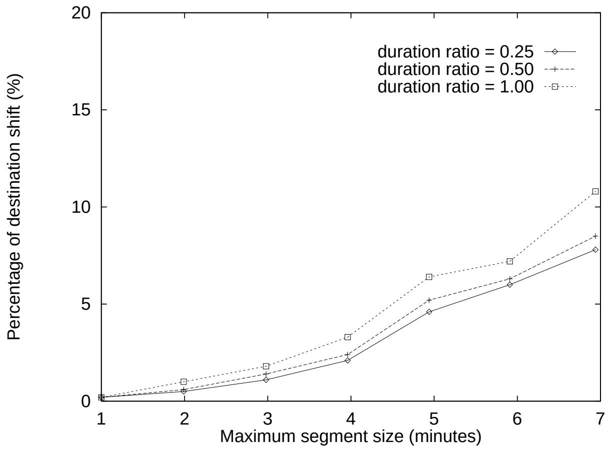

Fig. 9. The effect of maximum segment size on the percentage of destination shift

setting show that our scheme can deal with a very high level of user interactions.

We change the duration ratio in Figure 7, from 0.2 to 1.0. The percentage of buffer VCR actions decreases when the ratio increases, because the VCR actions last relatively longer with a higher duration ratio. The degradation of performance is larger than the case in which we change the probability. However, even when the ratio is 1.0, we still only need 3 loaders/buffers to keep the percentage of buffer VCR actions above 84%. In most practical situations, the ratio is usually low. We can achieve 95% buffer VCR actions when the ratio is 0.2.

Next we explore the effects of the maximum segment size. We set $p_v = 0.6$ and $\tau_0 = 2.58$ minutes. We let the duration ratio be 0.25, 0.50 and 1.0. We vary the number of the channels so that the size of maximum segment changes from 1.0 minute to 6.9 minutes. The correspondence is listed as follows.

# of channels	127	67	47	37	31	27	24
Segment size	1.0	2.0	3.0	4.0	4.9	5.9	6.9

The client has three loaders. When the size of the maximum segment is 6.9 minutes, each client needs three 6.9 minute buffers. So the total buffer size in this case is 20.7 minutes. We let the clients in the cases with a smaller maximum segment size have a buffer of the same size, so they may have more than 3 buffers of the maximum segment size.

Figure 8 shows the percentage of buffer VCR actions when the maximum segment size changes from 1 minute to 6.9 minutes. As the maximum segment size increases, the percentage of buffer VCR actions decreases. However, even when the segment size is 6.9 minutes, the percentage of buffer VCR actions is still above 82%. We can see from the plot that we can achieve better performance by increasing the number of channels and thus decreasing the maximum segment size. When the buffer size is 2 minutes, the percentage of buffer VCR actions is 93% for the duration ratio of 1.0.

Next we measure the percentage of destination shift for those adjusted VCR actions. Figure 9 shows the percentage of destination shift. In all cases, the percentages of shift are always less than 11%. With the total buffer size being the same, the client can achieve smaller shift if the segment size is smaller. This implies that the increase of the number of channels not only reduces the startup latency, but leads to better performance for VCR functions as well.

VI. Concluding Remarks

Multicast delivery is an attractive approach to the provision of video-on-demand services because it scales well to a very large number of clients. The problem is how to provide interactive functions to individual clients within the multicast framework without compromising the scalability of the multicast paradigm. After analyzing the problems of providing interactive functions in the existing broadcasting video-on-demand systems, we designed a VCR-friendly broadcast series and proposed an active buffer management technique to implement the functionality of providing interactive services in broadcast VoD systems. We showed that our scheme can implement VCR actions through buffering with a high probability in a wide range of user interaction levels.

References

[1] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of ACM Multimedia 94*, 1994, pp. 15–23.

[2] K. C. Almeroth and M. H. Ammar, "A scalable interactive video-on-demand service using multicast communication," in *Proceedings of International Conference on Computer Communication and Networks*, 1994, pp. 292–301, san Francisco, CA.

[3] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 3, no. 4, pp. 197–208, May 1996.

[4] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM Sigcomm'97*, 1997.

[5] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proceedings of NOSSDAV'98*, 1998.

[6] L.-S. Juhn and L.-M. Tseng, "Harmonic broadcasting for video-on-demand dervice," *IEEE Transactions on Broadcasting*, vol. 43, no. 3, pp. 268–271, September 1997.

[7] K. C. Almeroth and M. H. Ammar, "On the performance of a multicast delivery video-on-demand service with discontinuous VCR actions," in *Proceedings of ICC'95*, 1995, pp. 292–301, seattle, WA.

[8] ——, "On the use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 6, pp. 1110–1122, August 1996.

[9] W. Liao and V. O. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, no. 4, pp. 51–62, October-December 1997.

[10] E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS'98), Austin, Texas*, 1998, pp. 66–75.

[11] Z. Fei, I. Kamel, S. Mukherjee, and M. H. Ammar, "Providing interactive functions for staggered multicast near video-on-demand systems," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, June 1999, pp. 949–953, florence, Italy.

[12] Z. Fei, M. H. Ammar, I. Kamel, and S. Mukherjee, "Providing interactive functions through active client-buffer management in partitioned video multicast VoD systems," in *Proceedings of First International Workshop on Networked Group Communication (NGC'99)*, Nov. 1999, pp. 152–169, Pisa, Italy.

Zongming Fei Zongming Fei is Assistant Professor in the Department of Computer Science at University of Kentucky. He received his Ph.D. in computer science from Georgia Institute of Technology in 2000. His research interests include network protocols and architectures, overlay networks, mobile computing, and multimedia systems.

Mostafa H. Ammar Mostafa H. Ammar is a Regents' Professor with the College of Computing at Georgia Tech where he has been since 1985. His research interests are in the area of computer network architectures, protocols and services. He received the S.B. and S.M. degrees from the Massachusetts Institute of Technology in 1978 and 1980, respectively and the Ph.D. in Electrical Engineering from the University of Waterloo, Ontario, Canada in 1985. For the years 1980-1982, he worked at Bell-Northern Research (BNR), Ottawa, Canada.

He was the co-recipient of the Best Paper Awards at the 7th WWW conference for the paper on the "Interactive Multimedia Jukebox" and the 2002 Parallel and Distributed Simulation (PADS) conference for the paper on "Updateable Network Simulation." He served as the Editor-in-Chief of the IEEE/ACM Transactions on Networking from 1999 to 2003. He is a Fellow of the IEEE and a Fellow of the ACM.

Ibrahim Kamel Dr. Ibrahim Kamel got Ph.D. degree in computer science from University of Maryland, College Park 1994, M.Sc. in Environmental studies in 1988 and B.Sc. in computer science from university of Alexandria, Egypt in 1984. His research interest includes multimedia systems, spatial and temporal indexing, smart appliances, and Internet security. Dr. Kamel was a Lead scientist and a Group leader at Panasonic Information and Networking Technologies laboratories (PINTL), Princeton, NJ. He published more than forty papers in international conferences and journals and has nineteen patents filed in USA, Europe and Japan. He co-chaired several international workshops like the International Workshop on Multimedia Information Retrieval 2000 and the International workshop on Mining Spatial and Temporal Data 2001. Dr. Kamel participated in many program committees. He served in the Industrial Advisory committee for the Department of Electrical and Computer Engineering, University of Miami and the Center for Advanced Information Processing (CAIP), Rutgers University. He is now an Associate Professor at Zayed University, Dubai. Dr. Kamel is starting the first CSIRT (Computer Security Incident Response Team) in the Middle East in Dubai.

Sarit Mukherjee Sarit Mukherjee is a lead researcher in the Bell Laboratories' Center for Networking Research, leading the research and development of the next generation wireless data service technologies. Before this he held a Technical Manager position with Lucent Technologies managing the research and development of internet content distribution appliances. Prior to this he managed the design and development of streaming appliances in a New York-based start-up company, led the video networking group in Panasonic Information and Networking Technology Lab at Princeton, and taught as an assistant professor of Computer Science and Engineering at the University of Nebraska-Lincoln. He received his Ph.D. in Computer Science from the University of Maryland, College Park in 1993. He published in several renowned technical journals and conferences, served in the technical committees of number of international conferences, and holds dozen of US patents. His research interests include high speed network architectures and protocols, wireless networks and multimedia applications.