The Split and Merge (SAM) Protocol for Interactive Video-on-Demand Systems*

Wanjiun Liao and Victor O. K. Li†

Communication Sciences Institute
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-2565, USA

Abstract

A true Video-on-Demand (VOD) system provides the ultimate flexibility in video services by allowing users to select any video programs, at any time, and to perform any VCR-like user interactions. To allow true VOD, one approach is to have a dedicated video stream for each customer. This is expensive, especially when multiple identical video streams are sent to multiple customers accessing the same video. To be commercially viable, VOD service must be priced competitively with existing video rental services. Batching may be used to reduce this cost. It allows multiple users accessing the same video to share the same video stream. The batching approach, however, complicates the provision of user interactions. Existing batching schemes only allow near VOD services. This paper describes a new protocol, called Split and Merge (SAM), which offers true VOD services while allowing multiple users to share the same video stream. This sharing is transparent to the users and it appears as if each has a dedicated video stream. Our approach is to split an interactive user from the batch and to serve him with a dedicated video stream. We develop an innovative way to merge these individuals back to the batching streams when they resume normal play mode. The SAM protocol therefore significantly improves the system resource utilization and the number of simultaneous users, and more importantly, allows true VOD services.

1 Introduction

Video-on-Demand (VOD) combines the TV with information retrieval technology to provide electronic video rental services over the broadband network [7]. A VOD system contains many components, including the video server, the transport network, and the set-top box. Fig. 1 shows the typical architecture of a VOD system. For a more detailed description of the individual components, the reader is referred to [7].

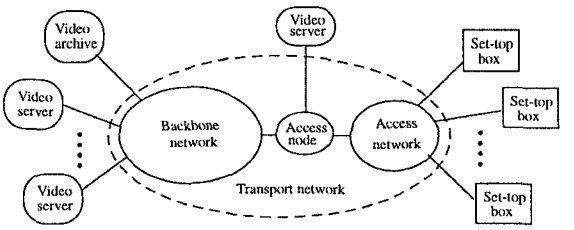

Figure 1: VOD system architecture.

In true VOD, customers are allowed to select the programs they wish, view at the time they wish, and interact with the programs via VCR-like functions, such as fast-forward, rewind, etc. [3, 4, 8, 10, 11]. A VOD system which does not satisfy all of the above requirements is called quasi-VOD or near VOD. To be competitive with existing video rental services, true VOD is desired. In fact, most existing VOD field trials [9] do provide true VOD. To allow true VOD, one solution is to have a dedicated video stream [1] for each customer. This is expensive since each stream requires high-speed [2] data transport, especially when multiple identical video streams are sent to multiple customers accessing the same video. To be commercially viable, VOD service must be priced competitively with existing video rental services. To reduce the per-user video

*This research is supported in part by the Pacific Bell External Technology Program.

†Corresponding author: vli@usc.edu, phone: (+1) 213-740-4665, and fax: (+1) 213-740-8729.

[1] A video stream will typically consist of video segments, retrieved by a read-write head from the video server, transported over a high speed network, and delivered to the set-top boxes at the customer premises.

[2] For example, existing VOD field trials typically deliver MPEG-1 or MPEG-2 compressed video, requiring 1.5 Mbps and 3 Mbps, respectively.

0-8186-7780-5/97 $10.00 © 1997 IEEE

delivery cost, batching may be used. In a batching operation, the same video stream will be multicasted to, and shared by, multiple users accessing the same video. The goal is to make this sharing transparent to the users, while allowing true user interactivity. Existing solutions fail to achieve this goal. In staggered VOD [2], for example, multiple copies (streams) of the same video program will be broadcasted, staggered in times. A user will be served by one of the streams, and user interactions are simulated by jumping to a different stream. However, not all user interactions can be simulated in this fashion, and even for those that can be simulated, the effect is limited by the staggering interval. Yu et al.[12] developed a look ahead scheduling with set-aside buffer which attempts to take advantage of batching, but only supports the interactive operation of pause and resume. Almeroth and Ammar [1] used the set-top box buffer to provide limited interactive functions, utilizing staggered streams. We would like to allow the full spectrum of user interactions. In [6], user interactions are handled by creating a new stream for each interactive user, who will hold on to this stream until disconnection. This will work only if very few users are expected to issue interactive operations. Otherwise, the system may start in a batch mode, but will degrade to a non-sharing mode as more and more users split off into their own streams. In this paper, we describe a new protocol, called Split and Merge (SAM) which allows the sharing of a video stream to be transparent to the users, while allowing true user interactivity. While user interactions are initially handled by splitting off the interactive user to a new stream in our protocol, we develop an innovative way in SAM to merge these individuals back to the batching streams. The SAM protocol therefore significantly improves the system resource utilization and the number of simultaneous users, and more importantly, allows true VOD services.

SAM may be implemented in various network infrastructures, including telephone, cable TV, direct broadcast satellite, wireless cable, local area networks, and on the Internet. For this paper, it suffices to say that we assume a generic network protocol running on a generic network infrastructure. The network protocol must support multicasting operations. In addition, some buffering, typically located at the access node, is available to be shared by all users for synchronization purposes. The rest of this paper is organized as follows. Section 2 presents the basic SAM protocol in detail. Variations of the basic scheme are presented in Section 3. Numerical results are shown in Section 4. Finally, we conclude in Section 5.

2 The Split and Merge (SAM) protocol

The goal of SAM is to reduce the per-user video delivery cost, or, alternatively, increase the number of users who can be served with given system resources, while providing true VOD services.

Since user interactions typically last a short time compared to normal play, we divide the video streams in the system into:

1. Service stream (**S** stream): such streams are used to serve the users during normal playback. It is typically a multicast stream and will serve multiple users simultaneously.

2. Interaction stream (**I** stream): such streams are used to satisfy some of the users' requests for VCR-like user interactions. An **I** stream is used by one user.

The fundamental principles of our proposed SAM protocol are as follows:

1. SAM fulfills true VOD services, while taking full advantage of batching. Originally, a number of users are batched and served by an **S** stream. Each of the batched users may initiate user interactions. As soon as a user interaction is admitted, this user will be split out of the original **S** stream, and temporarily allocated to an **I** stream to perform interaction. Once the user interaction is done, the user will be merged back to an on-going **S** stream[3]. Such split-and-merge operations are repeated, whenever a user interaction is initiated, until the original **S** stream terminates.

2. A user request for VOD service may be blocked if all **S** streams are occupied. Once the request has been admitted, however, any further user interactions will not be blocked, even if all system resources are busy, but will be allowed to wait until the resource is available. During the wait, a user continues normal playback, and switches to the user interaction mode as soon as resources are available.

3. SAM is an adaptive protocol. As demand for a particular video increases, more **S** streams will be generated in the system for that video. As demand wanes, less **S** streams will be generated. Although SAM works in any scenario, it works most efficiently for popular video.

[3]For the pause operation, no **I** stream is required and the user will be merged to an **S** stream as soon as the user resumes.

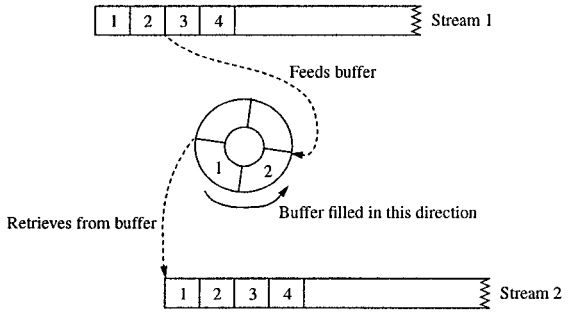

Figure 2: Illustration of how the synch buffer synchronizes two streams.

The *synch buffer*, located at the access nodes, and shared by all the users, is an important component of SAM. Each time a user resumes normal play after a user interaction, he requires a video stream (of the same video) which may be offset in time from his original **S** stream. For example, if he jumps forward by seven minutes, he needs a video stream which starts seven minutes before his original **S** stream. Since no such real stream may exist, SAM attempts to create a virtual one by utilizing one of the on-going **S** streams and the synch buffer. First we identify the closest on-going **S** stream, i.e., that with the smallest offset in time from the virtual stream. This real **S** stream feeds the synch buffer, and the virtual stream retrieves from the synch buffer, after the required time offset. Each user is dynamically allocated a synch buffer of size up to $SB \times R_p$, where SB is the maximum duration of video which can be stored for a user, and R_p is the playback rate of the stream. It is a circular buffer. Each circular buffer has two operation ends: one for putting in video contents and the other for taking them out. Fig. 2 illustrates how the synch buffer synchronizes two streams. Stream 1 (the real stream) is ahead of stream 2 (the virtual stream created). To help explain the operation, we have broken down the streams into segments[4], and labeled them 1, 2, ... Thus, stream 1 is 2 segments ahead of stream 2. Stream 1 will feed the synch buffer at one end, and stream 2 will retrieve buffer contents from the other end. Here, we show the contents of the buffer as stream 2 starts to retrieve the first segment from the buffer.

There are a number of videos available in the system. Suppose a request for a video, say video i, arrives. If there is already a batch being formed for video

[4]Note that these segments are artificial and are used to help us explain the concept. They are just equal sized pieces of the video

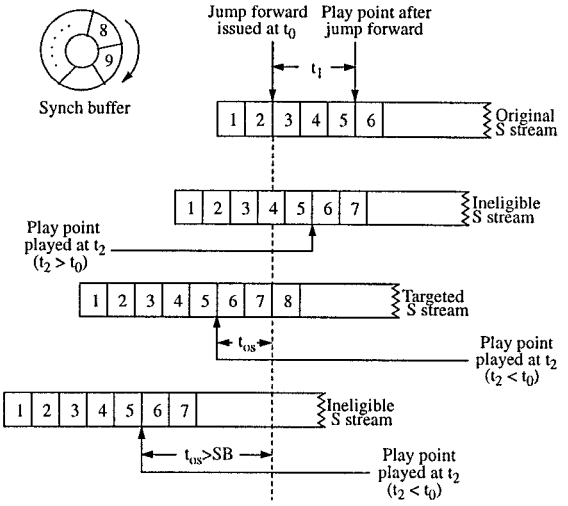

Figure 3: Illustration of the operation of jump forward.

i, this request will join the batch and wait for the end of the batching interval, at which time an **S** stream will be initiated to serve the batch; otherwise, the new request has to form a new batch. It will request an **S** stream from the pool of available **S** streams, and start a timer of duration W_i, the batching interval. If no **S** stream is available, the request will be blocked. The user may then try again later, or may just decide not to watch the video. If an **S** stream is available, it will be reserved, and after the batching interval W_i, an **S** stream will be initiated to serve this user, plus all other requests for video i which have arrived during the batching interval. Thus, after a user request is accepted, the maximum waiting time is W_i. The actual value of W_i is a system design parameter, corresponding to the maximum tolerable waiting time before a user reneges from the system. Note that each batch is initiated on-demand, rather than by periodical broadcasting, as in staggered VOD.

SAM allows users to interact with video programs via VCR-like functions including play, stop, pause, resume, fast-forward (FF), rewind (REW), jump-forward, and jump-backward. The jump operations allows the user to jump directly to a particular video location. Although not supported in current VCR machines, such random access operations, together with FF and REW, are expected to provide the most desirable search mechanism for digital video services. We will now describe how SAM supports these interactive operations.

2.1 Jump forward and jump backward

In the following, we focus on jump forward. Jump backward is handled in a similar fashion. First we determine if there is an *eligible* **S** *stream*. The concept of an eligible **S** stream is best explained by a figure. In Fig. 3, the user accessing video i is being served by the stream labeled original **S** stream. At time t_o, he issues a jump forward operation. Suppose the user jumps to a point in the video which is t_1 s in the future[5]. In Fig. 3, this corresponds to the beginning of segment 6. This is known as the *play point*, which is the location of the video at which the user will resume normal play after the user interaction. We look at all on-going **S** streams for video i and see if there is an eligible one for this user to merge into. An **S** stream is eligible if its corresponding play point (the beginning of segment 6 in this example) is before t_o, but not more than SB before, where SB is the maximum duration of video which may be stored in the synch buffer. SB is assumed to be four segments in this example. Thus in Fig. 3, the second **S** stream is ineligible because it has not reached the play point yet, while the last **S** stream is ineligible because it is more than SB segments offset in time from the virtual stream created by the jump, and the synch buffer is not large enough to synchronize to it. The stream labeled targeted **S** stream is eligible. If there are multiple eligible **S** streams, the user will merge with the one whose offset t_{os} with the virtual stream is the smallest. (This minimizes synch buffer usage.) Denote the time instant the play point in the targeted **S** stream is played by t_2. The offset t_{os} is defined as $t_{os} = t_o - t_2$.

Once a targeted **S** stream has been identified, we will find an **I** stream, and split the user from the existing batch. If no **I** stream is available, the request will join a FCFS queue. In the meantime, normal play continues from the original **S** stream. During the wait, the targeted eligible **S** stream may become ineligible, in which case we need to search for another eligible one. The **I** stream will be used for normal play for a duration equal to the time offset t_{os}. In the meantime, a connection will be made to the targeted **S** stream, which will feed the synch buffer. After the synch buffer has been fed for a period equal to t_{os}, corresponding to segments 8 and 9 in our example, the user will start to retrieve the video from the synch buffer, and release the **I** stream. In other words, the user has successfully merged with the **S** stream. If there is no eligible **S** stream to merge into, a new **S** stream will be initiated to serve this user. If there is no available **S** stream, the request will join a FCFS queue to wait for the first available **S** stream. In the meantime, normal play continues from the original **S** stream.

[5] If t_1 is too large, and carries the user beyond the end of the video, we will jump the user to the end of the video.

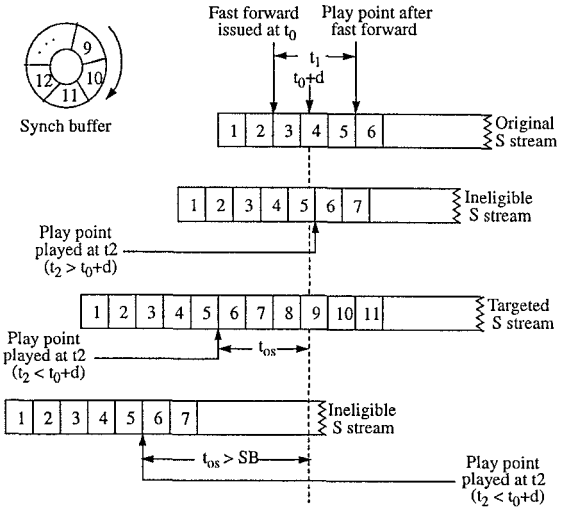

Figure 4: Illustration of the operation of fast forward.

The above assumes that the user was fed by an **S** stream directly. Suppose the user is being served by the synch buffer fed by an **S** stream, i.e., the user has issued an interactive operation earlier, how does the operation of jump forward change? Fortunately, it does not. Again, we need to identify the play point after the jump forward operation (beginning of segment 6 in this example), and the time the jump forward is issued (t_o in this example). We then try to merge with an existing **S** stream. The criteria for selecting this targeted **S** stream is the same as before. The only differences are that when we are waiting for an **I** or **S** stream, we will continue normal play from the buffer, fed by the original **S** stream, and when we have successfully merged with the targeted **S** stream, the connection to the original **S** stream is torn down, and everything in the buffer corresponding to the old **S** stream is discarded. In addition, if the jump forward is to a point in the video already in the buffer, we can avoid the split and merge operation altogether.

The operation of jump backward is similar, except the eligible **S** streams will have negative offsets.

2.2 Fast forward and rewind

Again each user is allocated a synch buffer of maximum size SB, assumed to be four in this example. Suppose the user accessing video i is being served by the stream labeled original **S** stream in the figure. At time t_o, he issued a fast forward operation.

An **I** stream is requested to serve the user. If one is available, video will be delivered in fast forward mode; otherwise, the request for an **I** stream joins a FCFS queue. In the meantime, normal play continues. After a duration d, the user terminates the fast forward operation, and resumes normal play. SAM will attempt to merge him back to one of the on-going **S** streams. Suppose the fast forward operation takes the user to a point t_1 beyond the initialization of the fast forward, i.e , the beginning of segment 6 in our example. We look at all on-going **S** streams for video i and see if there is an eligible one for this user to merge into. An **S** stream is eligible if its corresponding play point (the beginning of segment 6 in this example) is before $t_o + d$, but not more than SB before. Thus in Fig. 4, the second and the last **S** streams are ineligible. If there is no eligible **S** stream to merge into, a new **S** stream will be initiated to serve the user. If there is no available **S** stream in the system, the request will join a FCFS queue to wait for the first available **S** stream. In the meantime, normal play continues on the **I** stream. If there is at least one eligible **S** stream, the user will merge with the eligible **S** stream whose offset $t_{os} = t_o + d - t_2$ is the smallest. The **I** stream held by the user will continue to serve the user starting at the play point in normal play mode. In the meantime, the targeted **S** stream will feed the synch buffer. After the synch buffer has been fed for a period equal to t_{os}, corresponding to the segments 9 (last half), 10, 11, and 12 (first half) in our example, the user will start to retrieve the video from the synch buffer, and release the **I** stream. In other words, the user has successfully merged with the **S** stream.

As in jump forward, if the user is originally served by the synch buffer fed by an **S** stream, the operation of SAM remains pretty much the same.

The operation of rewind is similar, except the eligible **S** streams will have negative offsets.

2.3 Pause and resume

Again, each user is allocated a synch buffer of maximum size SB, assumed to be four in our discussion. Suppose the user accessing video i is being served the original **S** stream in Fig. 5. As soon as the pause is issued, the synch buffer is fed by the original **S** stream. After the pause operation, the only eligible **S** streams that the user may merge into must have started later than the original stream. Suppose the earliest of these eligible streams, labeled targeted **S** stream in the figure, starts a duration t later. This is the **S** stream that the user will try to merge into. We distinguish between two major cases. Case 1 occurs when $t \leq SB$, and case 2 occurs when $t > SB$. Case 1 (shown in Fig. 5(a)) can be further divided into subcases 1a and 1b. In case 1a, the pause period $d < t$. Since $d < t \leq SB$, the synch buffer is not yet full when the user resumes (in fact, it will contain segments 3 and 4 in this example), and he can just retrieve the video from the synch buffer. In case 1b, $d \geq t$, that is, the corresponding play point (the start of segment 3 in this example) of the closest eligible stream (the targeted **S** stream) will be played before the pause operation terminates. In the figure, since t corresponds to 3 segments, the corresponding play point will arrive after 3 segments of the original **S** stream, namely, segments 3, 4, and 5, have been stored. The user is merged (switched) to the new stream, i.e., the synch buffer will be fed by the targeted **S** stream starting at the play point. Case 2 (shown in Fig. 5(b)) can be further divided into subcases 2a and 2b. If the buffer is filled before the pause terminates, we have Case 2. A new **S** stream may have to be initiated, depending on whether the pause terminates before or after an ongoing **S** stream has reached the play point. As soon as the buffer is filled, a reservation request is made for a new **S** stream, the contents are purged from the synch buffer, and the user is split from the original **S** stream. In case 2a, the pause terminates before an ongoing **S** stream has reached the play point, a new **S** stream (the one we have reserved earlier) will be initiated to serve the user; otherwise, we have case 2b. It is possible that no **S** streams are available in the system, and the user will then join a FCFS queue to wait for the

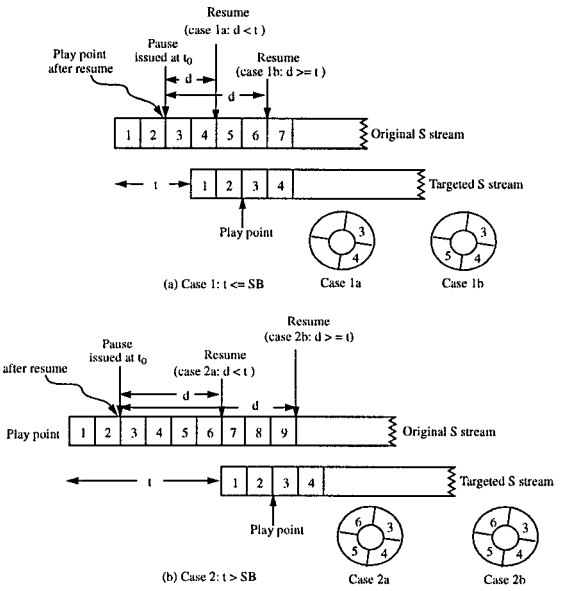

Figure 5: Illustration of the operation of pause.

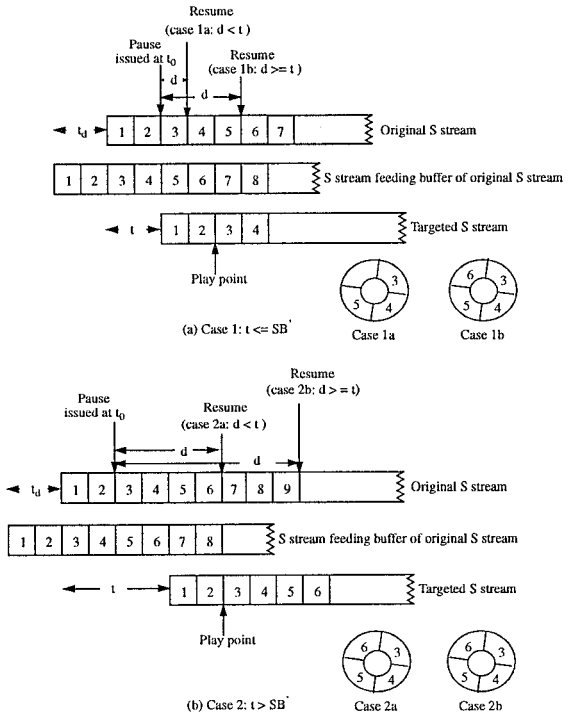

Figure 6: Illustration of the operation of pause when the user is operating from the synch buffer.

next available **S** stream, or to wait until an ongoing **S** stream has reached the play point, at which time he can merge with this ongoing stream. In case 2b, the corresponding play point (the start of segment 3 in this example) of the closest eligible stream will be played before the pause operation terminates. In the figure, since t corresponds to 6 segments, the corresponding play point will arrive after 6 segments of the original stream have arrived. Since the buffer is of size 4, it can only store four of them, namely, segments 3, 4, 5, and 6. The reserved **S** stream is released, the user is merged (switched) to the new stream as soon as the synch buffer is full, and the synch buffer is now fed by the new stream, starting at the play point.

In cases 1b and 2b, since the user is still in the pause mode after he is switched to the targeted ongoing **S** stream, the above procedure has to be repeated, perhaps multiple times, until he terminates the pause operation.

The above assumes that the user is fed by an **S** stream directly. We now study the situation in which the user is being served by the synch buffer fed by an **S** stream, i.e., the user has issued an interactive operation earlier. Fig. 6 illustrates how SAM handles pause and resume in this situation. Again, each user is assumed to be allocated a synch buffer of size four in this example. Suppose the user accessing video i is being served by the original **S** stream in Fig. 5. Note that this is a virtual **S** stream, and is fed by an ongoing **S** stream which has started two segments earlier. In other words, two segments of the synch buffer have already been taken up in order to synch to this feeding **S** stream, leaving only two segments available. This is the only difference between how SAM handles the pause operation when the user is fed directly, or via a synch buffer. Note that the synch buffer size is now $SB' = SB - t_d$, where t_d is the lead time of the feeding **S** stream compared to the original **S** stream. In addition, t, the time until the nearest eligible ongoing **S** stream, is calculated from the original **S** stream, not the feeding **S** stream. Since t_d is 2 in our example, $SB' = 2$. t is 2. The operation is the same as when the user is fed by a real **S** stream. The only difference is the size of the synch buffer shall be SB' instead of SB. In addition, once a user switched to a new stream (case 1b and case 2b), he can discard the existing contents in the synch buffer, effectively augmenting his synch buffer to the maximum value of SB again.

3 Variations of the basic scheme

The following describes variations of the basic SAM protocol:

1. No initial batching delay – As soon as a video request is received, determine if there are eligible **S** streams. If there is, serve this request with an **I** stream while feeding synch buffer with the ongoing **S** stream for an interval equal to the offset. After the offset, merge into the on-going stream for this video. Otherwise, initiate a new **S** stream to serve this request.

2. Adjustable batching intervals – Batching interval may be different for different videos, due to differences in popularity. More popular ones should have shorter batching interval. In addition, since popularity may change over time, we can periodically change the batching interval, based on observed video request rate in the previous period.

3. A variation of basic batching idea – Instead of initiating a timer when the first request in a batch arrives, we divide time into fixed length intervals, say every five minutes. If at least one request for video i arrives during an interval, initiate an **S** stream. This has the advantage that, for popular videos, average wait is equal to half of the

batching interval, instead of the batching interval. These will be not much difference for popular ones.

4. A synch buffer may have one input stream, but multiple output streams, serving different users with different time offsets. This saves buffering.

5. We can use a pricing mechanism to control the amount of user interactivity. The more the user is willing to pay, the closer he gets to true VOD service. At the high end, no initial batching delay, and full user interactivity is allowed; then, initial batching delay, full interactivity. Finally, for the least cost, initial batching delay, limited interactivity.

4 Numerical Results

In this section, we present some simulation results. The results are generated by a C program running on a HP C160. 24 hours of real time are simulated, requiring on the average 30 minutes of CPU time for each arrival rate. The following system parameters are used. There are 30 available videos, each lasting 120 minutes. The probability a particular movie is accessed, p_i, $i = 1, 2, \ldots, 30$, is assumed to follow the Zipf distribution [5]. Therefore, video 1 is the most popular, followed by video 2, etc. We also assume the following user activity model. A user starts in the normal play mode, and will stay there for an exponential amount of time with mean 30 minutes, then he goes to the interaction mode with probability 0.75, and he quits with probability 0.25. In the interaction mode, he is equally likely to issue a jump forward, jump backward, fast forward, rewind, or pause operation. Each pause is exponentially distributed with a mean of 5 minutes. Each fast forward/rewind is exponentially distributed with a mean of 0.5 minutes, i.e., the user will be holding down the fast forward or rewind button for an average of 0.5 minutes. Fast forward and rewind will take the user to a point in the video which is offset from the original point by an interval uniformly distributed between 1 and 90 seconds. Each jump operation lasts one second, and will take the user to a point in the video which is offset from the original point by an interval uniformly distributed between 1 and 1000 seconds. After a user interaction, the user resumes normal play mode. This is repeated until the user quits.

The total number of video streams is 515. The batching interval used is 10 minutes, and the maximum synch buffer allocated per user varies from one to ten minutes. Fig. 7 shows how the blocking probability changes as the video request rate increases for the

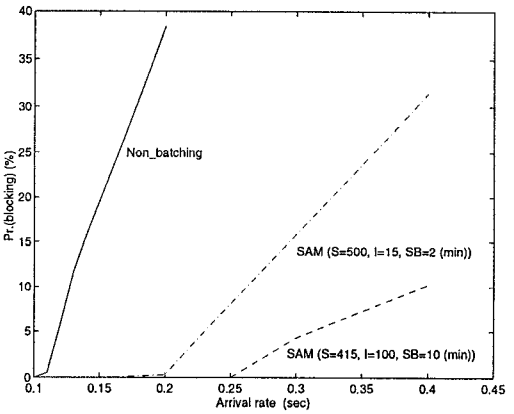

Figure 7: Blocking probability for the cases of batching and no batching.

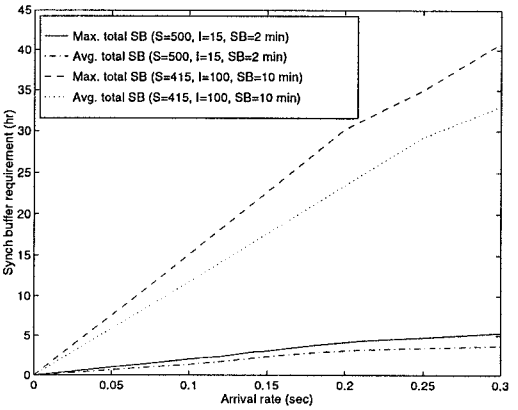

Figure 8: The average amount of synch buffer required as a function of arrival rates.

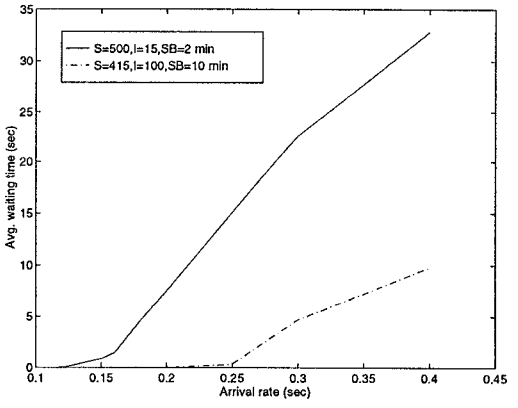

Figure 9: Average interaction delay as a function of arrival rates.

cases of batching (SAM) and non-batching (point-to-point connection for each user). As expected, there is a large reduction in the blocking probability with SAM[6]. The price we pay is the initial batching delay, which is bounded by 10 minutes, and averages 5 minutes. In addition, we require a synch buffer and the user may experience some interaction delay. Fig. 8 shows the average total synch buffer required in the system as a function of arrival rate. Note that this translates into very small per-user synch buffer requirements. For example, the buffer required is 13.2 seconds per user when the arrival rate is 0.3 requests per second, for S= 500, I = 15, and SB = 2 minutes. Fig. 9 shows the average interaction delay, which is acceptable except for a highly loaded situation. (To ensure that the blocking probability is small, we will not operate the system at such high load anyway.) This leads us to conclude that our proposed SAM protocol is an excellent candidate to be used in VOD systems.

5 Conclusions

VOD is expected to be the most important commercial application of distributed multimedia systems. It provides an electronic video rental service, which gives the users the ultimate flexibility in selecting any video programs, at any time, and in performing any VCR-like user interactions. However, to be commercially successful, VOD must be priced competitively compared with existing video rental services. It is estimated that approximately half of the video rental revenues go to the program providers. That means the other half will go towards the cost of delivering the video, and for the profits of the service provider. In existing video rental stores, the major cost of the delivery is borne by the users, and the only costs incurred by the service provider is shelf space. For VOD, the costs of video delivery include the costs of the high capacity video server, and the high speed network, both of which are substantial. Our proposed protocol allows multiple users to share the same video stream, thereby dramatically increasing the capacity of the system, and greatly reducing the costs per user. At the same time, the price to be paid, i.e., batching delay, interaction delay, etc. are tolerable. This leads us to conclude that our proposed SAM protocol is an excellent candidate to be deployed in interactive VOD systems.

[6] Actually, the improvement for popular videos is much more dramatic than that indicated in the figure, which shows the reduction in blocking probabilities averaged over all videos, including both popular and unpopular ones.

References

[1] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal On Selected Areas in Communications*, 14(6):1110–1122, August 1996.

[2] Robert O. Banker et al. Method of providing video on demand with VCR like functions. In *U.S. Patent 5,357,276*, 1994.

[3] Y. H. Chang et al. An open-systems approach to video on demand. *IEEE Communications Magazine*, 32(5):68–80, May 1994.

[4] D. Deloddere, W. Verbiest, and H. Verhille. Interactive video on demand. *IEEE Communications Magazine*, 32(5):82–88, May 1994.

[5] D. Knuth. *The art of computer programming, Vol 3: Sorting and searching.* Addison-Wesley, 1973.

[6] V. O. K. Li et al. Performance model of interactive video-on-demand systems. *IEEE Journal On Selected Areas in Communications*, 14(6):1099–1109, August 1996.

[7] V. O. K. Li and W. J. Liao. Distributed multimedia systems. *Report No. CSI-96-05-01, Communication Sciences Institute, University of Southern California*, May 1996. (Submitted for publication).

[8] T. D. C. Little and D. Venkatesh. Prospects for interactive video-on-demand. *IEEE Multimedia*, pages 14–24, Spring 1994.

[9] T. S. Perry. The trials and travails of interactive TV. *IEEE Spectrum*, pages 22–28, April 1996.

[10] W. D. Sincoskie. System architecture for a large scale video on demand service. *Computer Networks and ISDN Systems*, 22:155–162, 1991.

[11] J. Sutherland and L. Litteral. Residential video services. *IEEE Communications Magazine*, pages 36–41, 1992.

[12] P. S. Yu, J. L. Wolf, and H. Shachnai. Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand application. *Multimedia Systems*, 3(4):137–150, 1995.