The Split and Merge Protocol for Interactive Video-on-Demand

Wanjiun Liao
National Taiwan University

Victor O.K. Li
The University of Hong Kong

A true video-on-demand (VoD) system lets users view any video program, at any time, and perform any VCR-like user interactions. To reduce the per-user video delivery cost, multiple users may be batched and share the same video stream. Existing sharing schemes do not allow true VoD. A new protocol, called Split and Merge (SAM), does allow true VoD. SAM also provides an innovative way to merge these individuals back into the batching streams when they resume normal play mode.

Video-on-demand (VoD) combines the service quality of cable TV with the interaction capabilities of a VCR (see Figure 1). Its birth followed technical advances in information retrieval, consumer electronics, networking, and digital signal processing, and the convergence of communications, computers, and entertainment. With VoD services, customers may select programs from massive, remote video archives, view them when they wish, and interact with the programs using VCR-like functions, such as fast-forward and rewind. A true VoD system satisfies three important requirements: viewing any video, at any time, using any VCR-like user interactions. A system that does not satisfy all these requirements is called a near-VoD system.

To compete with existing video rental services requires true VoD, which most existing VoD field trials[1] do indeed provide. One solution for true VoD assigns a dedicated video stream to each customer. (A video stream typically consists of video segments retrieved by a read-write head from the video server, transported over a high-speed network, and delivered to set-top boxes at the customer's premises.) This gets expensive, since each stream requires high-speed data transport. For example, existing VoD field trials typically deliver MPEG-1 or MPEG-2 compressed video, requiring 1.5 Mbps and 3 Mbps, respectively. To be commercially viable, VoD service pricing must compete with existing video rental pricing.

Batching[2] can reduce the per-user video delivery cost. (Batching here does not necessarily mean waiting a certain amount of time before serving user requests—it simply means resource sharing.) In a batching approach the same video stream is multicast to, and shared by, multiple users. This sharing, however, means that one user's interaction affects all other users on the same stream. For example, to satisfy one user's desire to rewind the video, all other users sharing the same stream would have to rewind. This won't work. We want to make the sharing transparent to users while allowing true user interactivity.

Existing batching solutions fail to achieve this goal. For example, staggered VoD[3] broadcasts multiple copies (streams) of the same video program at staggered times, with one stream serving multiple users. Jumping to a different stream is used to perform a user interaction. However, not all user

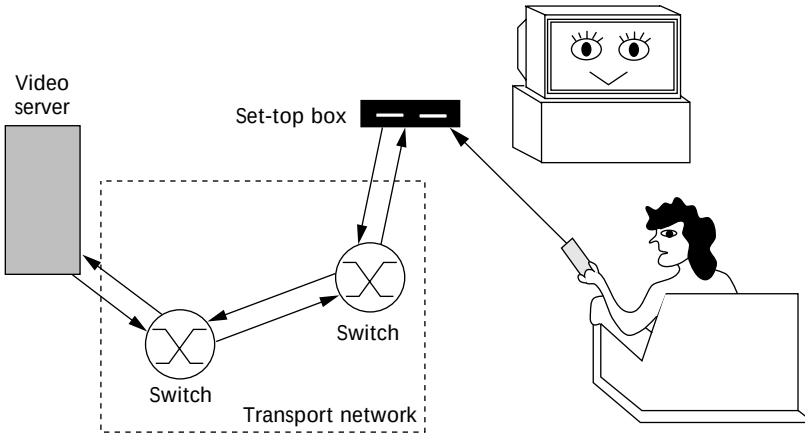


Figure 1. A VoD system combines the service quality of cable TV and the interaction capabilities of a VCR.

1070-986X/97/$10.00 © 1997 IEEE

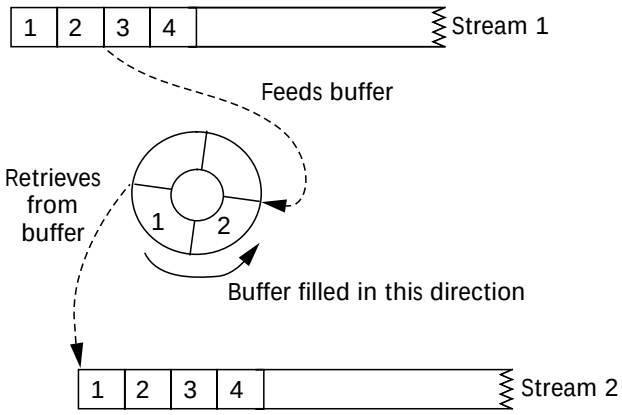


Figure 2. Illustration of how the synch buffer synchronizes two streams.

interactions can be performed by jumping streams. Consider, for example, fast-forward—none of the streams is in fast-forward mode. Moreover, even though this system can mimic some interactions, it might not deliver exactly the effect the user wants. For example, given a staggering interval of 5 minutes, you can jump forward for, say, 5, 10, or 15 minutes, but not for 7 minutes.

For staggered streams Almeroth and Ammar[4] used the set-top box buffer to provide limited interactive functions. Yu et al.[5] developed the look-ahead scheduling with set-aside buffer protocol, which attempts to take advantage of batching but only supports the interactive operation of pause and resume. Another approach[6] creates a new stream to handle user interactions for each interactive user, who will hold onto this stream until disconnection. This approach will work only if very few users issue interactive operations. Otherwise, the system may start in a batch mode, but will degrade to a nonsharing mode as more and more users split off into their own streams. In a different approach,[7] Golubchik, Lui, and Muntz proposed adaptive piggybacking. This involved changing the display rates of user requests to batch the nearby streams, thereby reducing the aggregate I/O bandwidth on the server.

In this article we describe a new protocol, called Split and Merge (SAM), which makes the sharing of a video stream transparent to users while allowing true user interactivity. Initially the protocol handles user interactions by splitting off the interactive user to a new stream. Then it provides an innovative way of merging these individuals back into the batching streams. The SAM protocol therefore significantly improves system resource use and the number of simultaneous users. More importantly, it enables true VoD services.

SAM works in various network infrastructures, including telephone, cable TV, direct broadcast satellite, wireless cable, local area, and the Internet. For this article, we assume a generic network protocol running on a generic network infrastructure. The network protocol must support multicasting operations. For example, we can run Asynchronous Transfer Mode (ATM) on a Hybrid Fiber Coax network. In addition, for synchronization purposes all users can share some buffering, typically located at the access node.

The Split and Merge (SAM) protocol

The SAM protocol aims to provide true VoD services while reducing the per-user video delivery cost or, alternatively, increasing the number of users served with given system resources. We can summarize SAM as follows:

> Split and Merge (SAM) refers to the split and merge operations incurred when each user performs user interactions. These operations enable any kind of user interactions. SAM starts by serving customers in a batch. When a user in a batch initiates a user interaction, the protocol splits off the interactive user from the original batch and temporarily assigns that user to a new video stream. With a dedicated video stream, the user can perform any interactions desired. As soon as the user interaction terminates, the system merges this user back to the nearest ongoing video stream.

Since user interactions typically last a short time compared to normal play, we divide the system's video streams into two types, service and interaction. *Service streams* (S streams) serve users during normal playback. Typically a multicast stream, an S stream serves multiple users simultaneously. *Interaction streams* (I streams) satisfy some user requests for VCR-like interactions, one I stream for one user.

The SAM protocol's fundamental principles follow:

1. SAM delivers true VoD services while taking full advantage of batching. To begin, a number of users are batched and served by an S stream. Each user may initiate user interactions, but then splits from the original S stream and is temporarily allocated to an I stream for interaction. When done, the user merges back to an ongoing S stream. Note that the pause operation does not require an I

stream. Such split-and-merge operations repeat whenever a user performs interaction, until the original S stream terminates.

2. The system may block a user request for VoD service if all S streams are occupied. Once the system has admitted the request, however, it will not block further user interactions even if all system resources are busy. The request waits until the resource becomes available. During the wait, a user continues normal playback, switching to the user interaction mode when resources become available.

3. SAM is an adaptive protocol. As demand for a particular video increases, the system will generate more S streams for that video. As demand wanes, so do S streams. Although SAM works in any scenario, it works most efficiently in a system where some videos are very popular and accessed by many users.

The *synchronization (synch) buffer*, shared by all users, is an important component of SAM. Each user resuming normal play after a user interaction requires a video stream (of the same video), possibly offset in time from the original S stream. For example, a jump forward by seven minutes necessitates a video stream that started seven minutes before the original S stream. Since no such real stream may exist, SAM attempts to create a virtual one by using an ongoing S stream and the synch buffer. First the protocol identifies the closest ongoing S stream—the one with the smallest offset in time from the virtual stream. This real S stream feeds the synch buffer, and the virtual stream retrieves content from the synch buffer after the required time offset.

The synch buffer is circular. Each circular buffer has two operation ends, one for putting in video contents and the other for taking them out. The synch buffer's main role involves creating a virtual video stream for resource sharing.

Figure 2 illustrates how the synch buffer synchronizes two streams. To help explain the operation, we divided the streams into artificial segments labeled 1, 2, and so on. Stream 1 is a real and ongoing video stream. We want to create another video stream—stream 2 in the same figure—with a start time two segments behind stream 1. Instead of allocating system resources to generate this new video stream, stream 1 with two segments' worth of synch buffer is used to create stream 2—a *virtual* stream two segments offset in time from stream 1. At any particular time, stream 1 will feed the synch buffer at one end, and stream 2 will retrieve buffer contents from the other end. Figure 2 shows the buffer contents as stream 2 starts to retrieve the first segment.

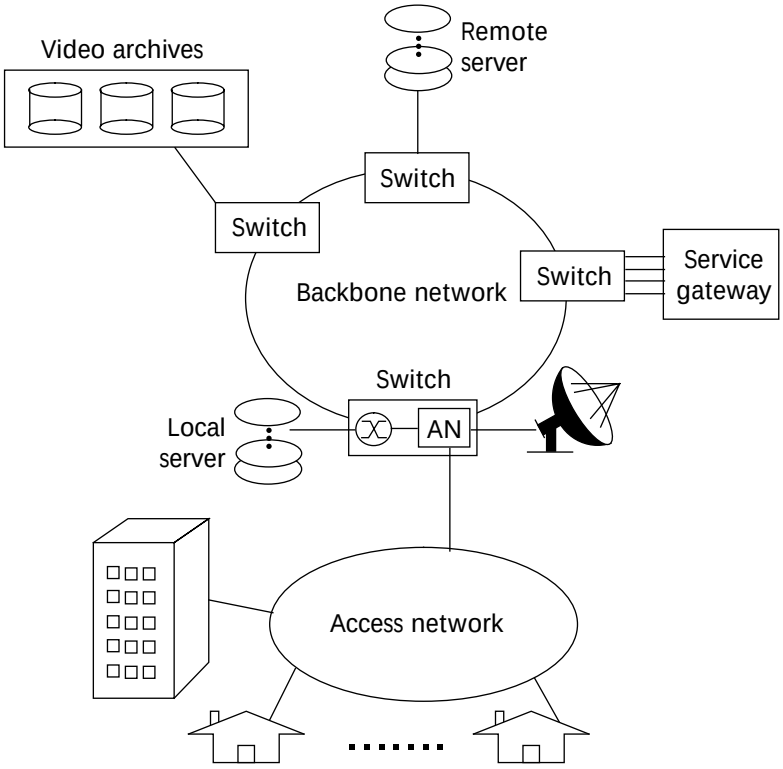


Figure 3. VoD system architecture. (AN stands for access node.)

The synch buffer's actual location depends on the network infrastructure. For large-scale VoD (residential, for example) with a typical network infrastructure as shown in Figure 3, it might sit at the access nodes. If so, both server and network bandwidth can be shared. For small-scale VoD (enterprise, for example) running on a local area network, the synch buffer might be located at the video server.

The system dynamically allocates to each user a synch buffer up to $SB \times R_p$, where SB equals the maximum duration of video storable per user and R_p represents the stream's playback rate. The size of SB is a system design parameter. Even though each user can claim up to a maximum synch buffer size, in general usage remains much less. For example, in Figure 2 the maximum synch buffer size per user is four segments, while the actual usage is two segments.

The flowchart in Figure 4 illustrates SAM's operation. Suppose a request for video *i* arrives. If a batch is already forming for video *i*, this request joins the batch and waits for the end of the batch-

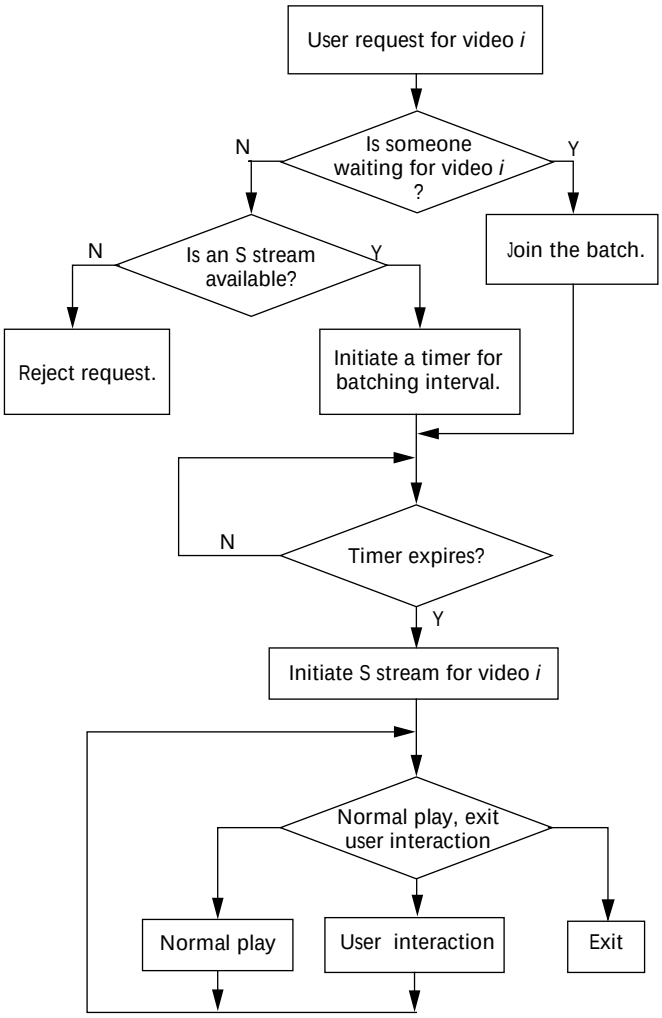


Figure 4. Flowchart of the SAM protocol's operation.

ing interval. At that time the system initiates an S stream 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 is blocked. The user may then try again later or just decide not to watch the video. If an S stream is available, it is reserved. After the batching interval W_i, an S stream is initiated to serve this user plus all other requests for video *i* that have arrived during the batching interval. Thus, after the system accepts a user request, 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.

Note that even though the SAM flowchart shows an initial batching delay, this is not necessary. One variation of this basic scheme (discussed later) avoids the initial batching delay.

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

Jump-forward and jump-backward

Figure 5 shows the flowchart for the jump-forward, jump-backward operation. The following discussion focuses on jump-forward, since SAM handles jump-backward similarly.

First the system determines if an eligible S stream exists. We can best explain the concept of an eligible S stream with a figure. In Figure 6, the stream labeled original S serves the user accessing video *i*. At time t_o the user issues a jump-forward operation. Suppose the user jumps to a point in the video t_1 seconds in the future. (Note that if t_1 would carry the user beyond the end of the video, the jump instead stops at the end of the video.) In Figure 6, t_1 seconds corresponds to the beginning of segment 6. This spot, known as the *play point*, is the video location at which the user resumes normal play after interaction.

The system looks at all ongoing S streams for video *i* to find an eligible one for the user to merge into. An S stream is eligible if its corresponding play point (the beginning of segment 6 in this example) plays before t_o, but not more than *SB* before, where *SB* equals the maximum size of the synch buffer dedicated to a user. This example assumes that *SB* is four segments. Thus in Figure 6, 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 offset in time more than *SB* segments from the virtual stream created by the jump—the synch buffer is not large enough to synchronize to it.

The stream labeled targeted S is eligible. Given multiple eligible S streams, the user will merge with the one having the smallest offset t_{os} from the virtual stream. (This minimizes synch buffer usage.) Denote by t_2 the time the play point in the targeted S stream plays. The offset t_{os} is defined as $t_{os} = t_o - t_2$. Once the system has identified a tar-

geted S stream, it will find an I stream and split the user from the existing batch. If no I stream is available, the request will join a first-come-first-served (FCFS) queue, and 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 find another eligible one. The I stream handles normal play for a duration equal to the time offset t_{os}. At the same time, a connection is 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.

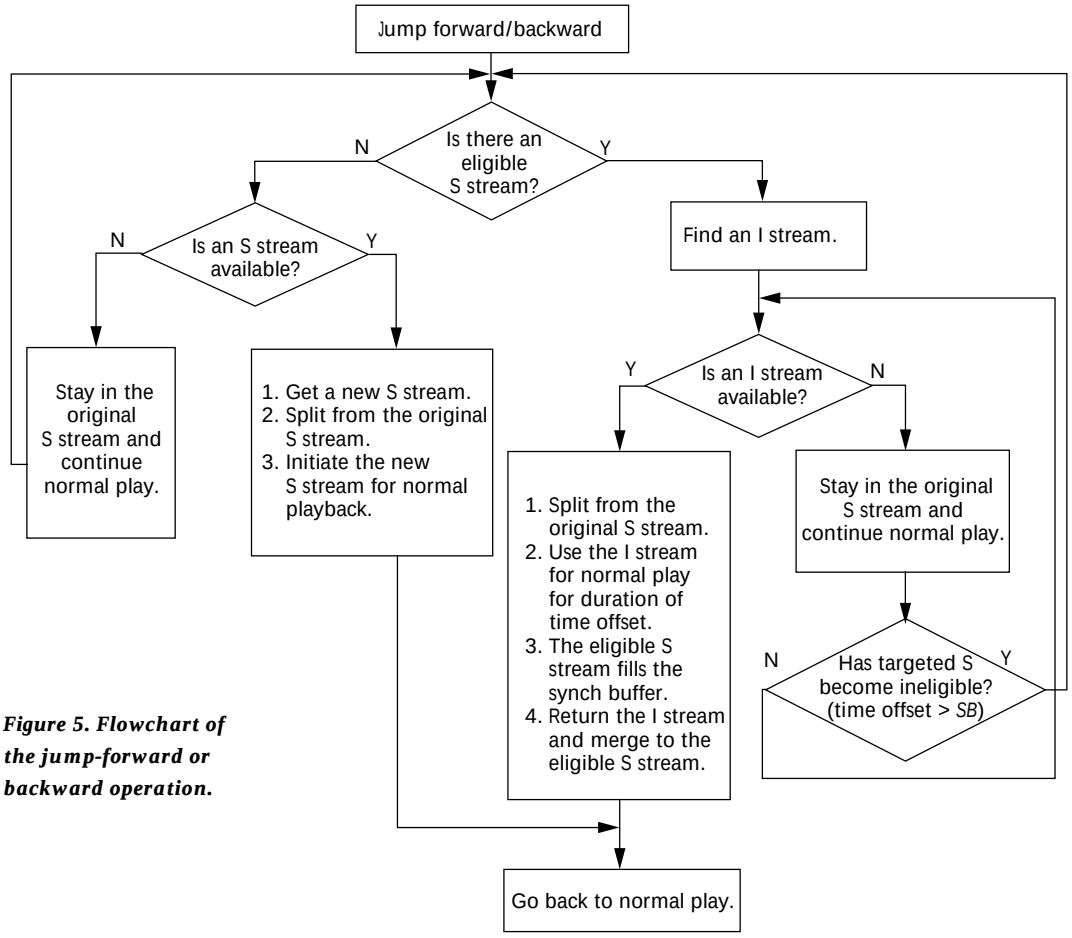


Figure 5. Flowchart of the jump-forward or backward operation.

If no eligible S stream is available for the user to merge into, the system initiates a new S stream to serve this user. If the system finds no available S streams, the request will join an FCFS queue to wait for the first available S stream. In the meantime, normal play continues from the original S stream.

This discussion assumes that an S stream feeds the user directly. Suppose instead the synch buffer, fed by an S stream, serves the user. That is, earlier the user issued an interactive operation. How does the jump-forward operation change? Fortunately, it doesn't. Again, we need to identify the play point after the jump-forward operation (the beginning of segment 6 in this example) and the time at which the user issued the jump-for-

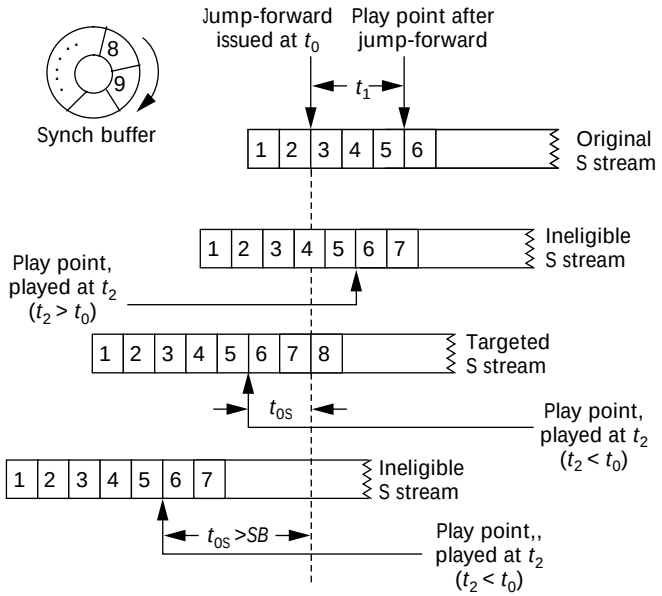


Figure 6. Illustration of the jump-forward operation.

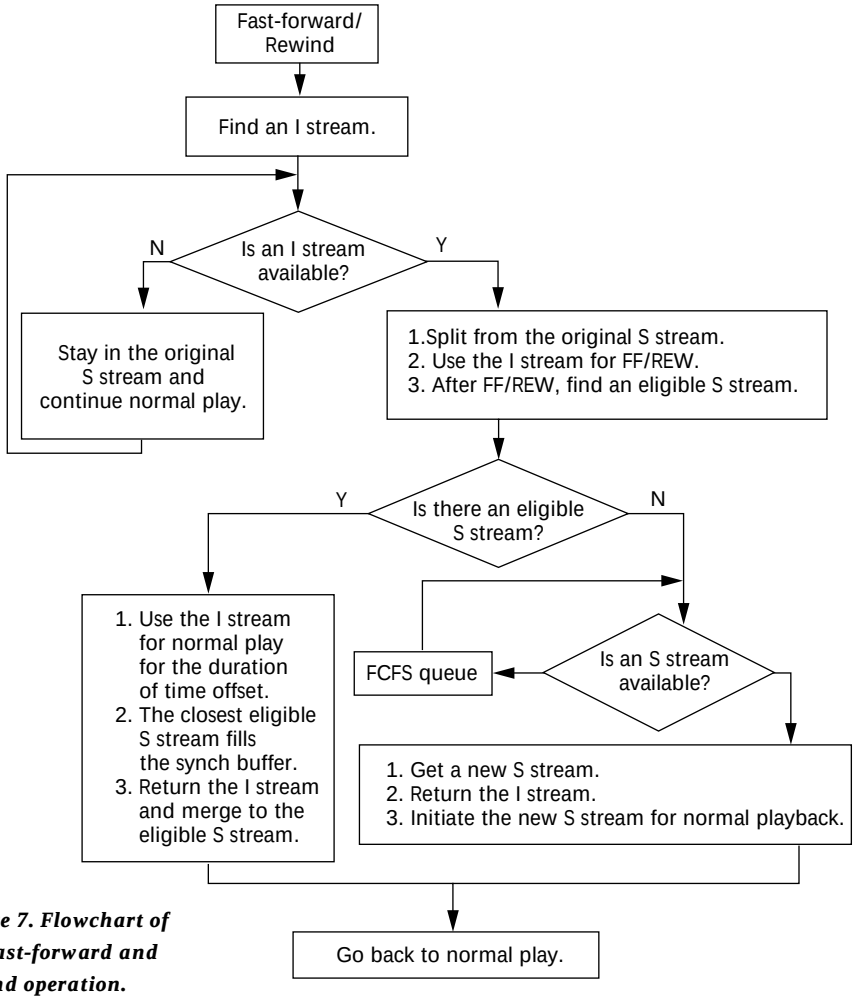


Figure 7. Flowchart of the fast-forward and rewind operation.

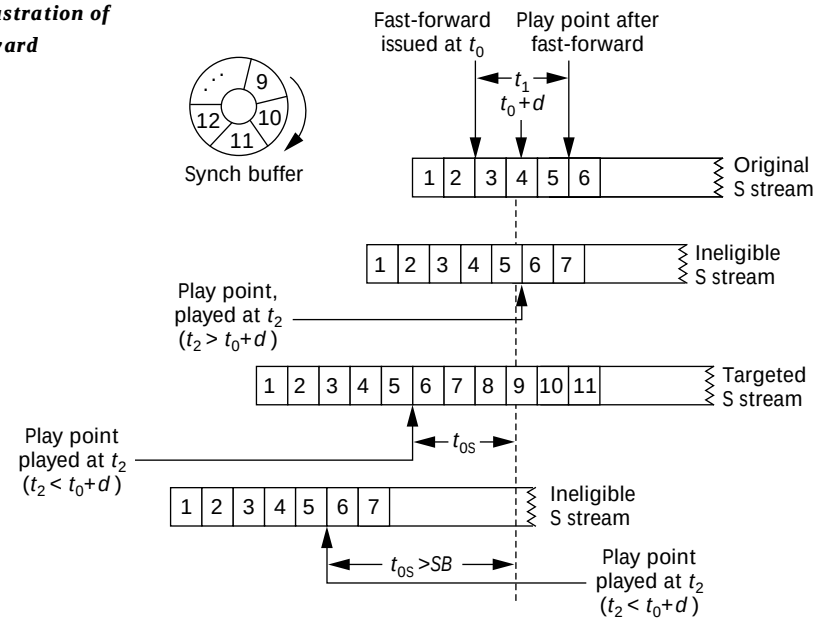


Figure 8. Illustration of the fast-forward operation.

ward (t_o in this example). We then try to merge with an existing S stream.

The same criteria as before applies when selecting this targeted S stream, with just a few differences: First, when waiting for an I or S stream, normal play continues from the buffer, fed by the original S stream. Second, following a successful merge with the targeted S stream, the connection to the original S stream ends 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.

Fast-forward and rewind

The flowchart for the fast-forward and rewind operation appears in Figure 7. Figure 8 illustrates how SAM handles fast-forward. Again each user is allocated a synch buffer of maximum size *SB*, four in this example. Suppose the stream labeled original S in the figure serves the user accessing video *i*. At time t_o, the user issues a fast-forward operation. The protocol requests an I stream to serve the user. If one is available, the system will deliver video in fast-forward mode; otherwise, the request for an I stream joins an FCFS queue. Meanwhile, normal play continues. After a duration *d*, the user terminates the fast-forward operation and resumes normal play. SAM then attempts to merge the user back to one of the ongoing S streams.

Suppose the fast-forward operation takes the user to a point t_1 beyond initialization of the fast-forward—the beginning of segment 6 in our example. SAM looks at all ongoing S streams for video *i* in search of 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) precedes $t_o + d$, but not more than SB before. Thus in Figure 8, the second and the last S streams are ineligible.

If the protocol finds no eligible S stream, it initiates a new S stream to serve the user. If there is no available S stream in the system, the request will join an FCFS queue to wait for the first available S stream, and 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. At the same time, 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 synch buffer fed by an S stream serves the user originally, SAM's operation remains pretty much the same.

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

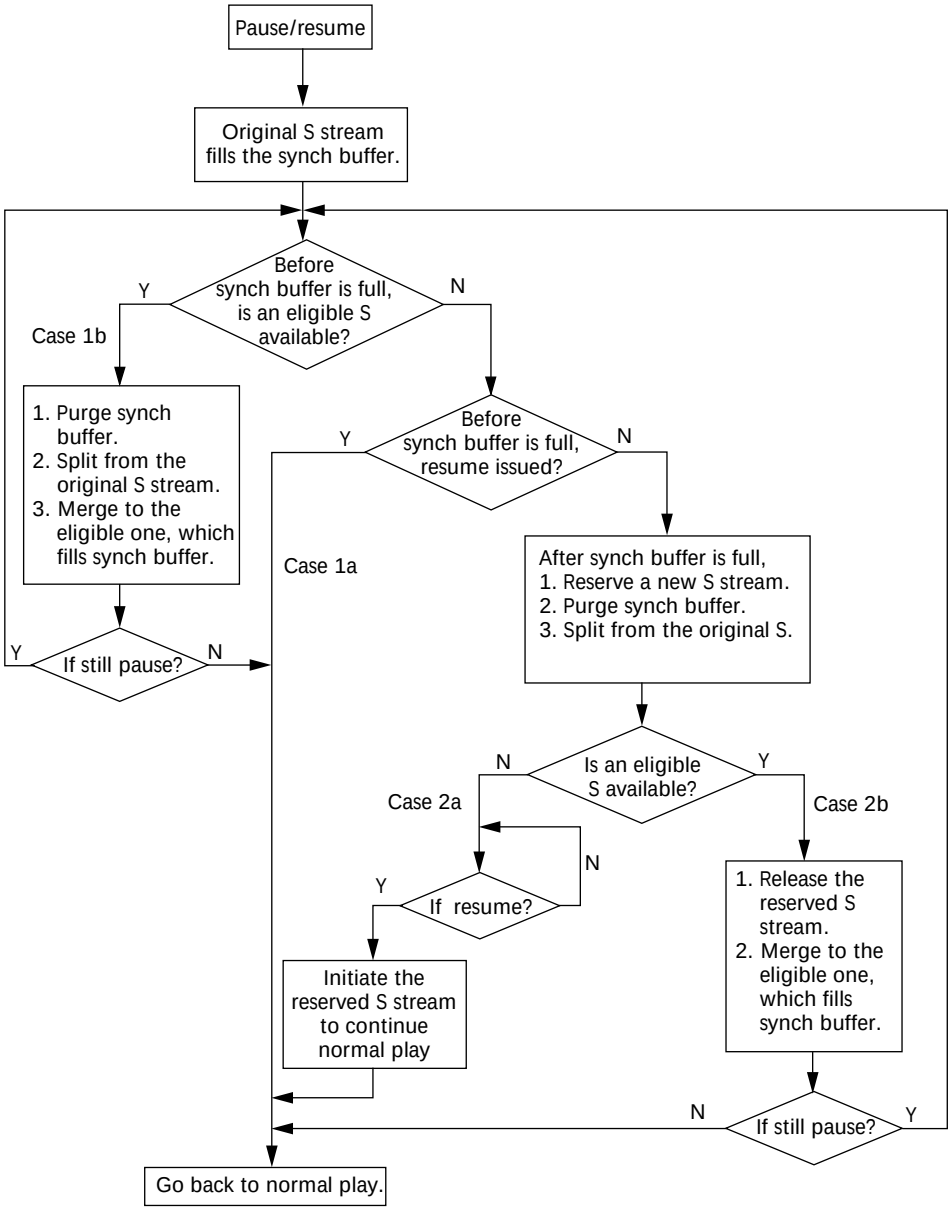
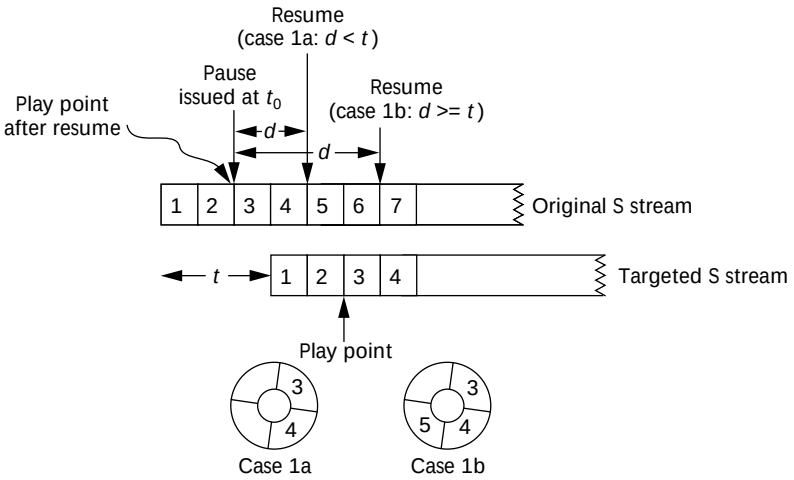


Figure 9. Flowchart of the pause and resume operation.

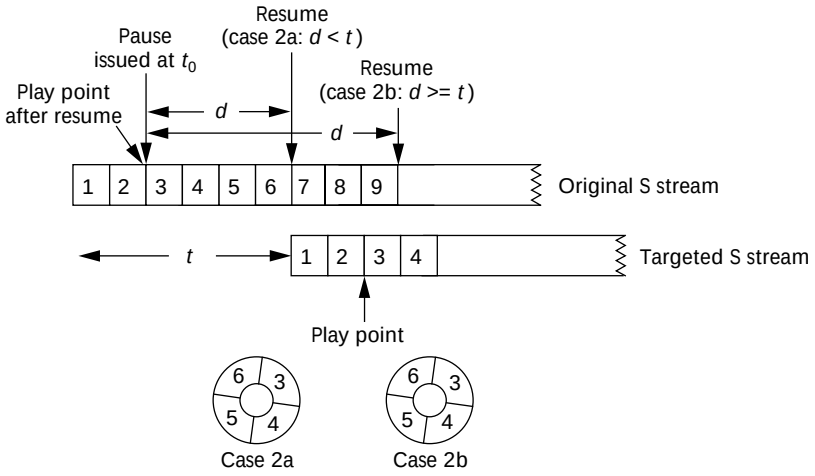
Pause and resume

The flowchart for the pause and resume operation appears in Figure 9. Again, the system allocates each user a synch buffer of maximum size SB, four in our discussion. Suppose the original S stream in Figure 10 (next page) serves the user accessing video i. As soon as the pause begins, the original S stream feeds the synch buffer. 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 in the figure, starts a duration t later. The user will try to merge into this S stream. We distinguish between two major cases. Case 1 occurs when $t \le SB$, and case 2 occurs when $t > SB$. Case 1 (shown in Figure 10a, next page) further divides into 1a and 1b. In case 1a, the pause period $d < t$. Since $d < t \le SB$, the synch buffer is not yet full when the user resumes. (In fact, it will contain segments 3 and 4 in this example.) The user can simply retrieve the video from the synch buffer. In case 1b, $d \ge t$, meaning the corresponding play point (the start of segment 3 in this example) of the closest eligible stream (the targeted S stream) will play before the pause operation terminates. In Figure 10a, since t corresponds to three segments, the corresponding play point will arrive after three segments of the original S stream—segments 3, 4, and 5—have been stored. The user



(a) Case 1: $t <= SB$



(b) Case 2: $t > SB$

Figure 10. Illustration of the pause operation.

merges (switches) to the new stream, meaning the targeted S stream starting at the play point will feed the synch buffer. If the buffer fills before the pause terminates, we have Case 2.

Case 2 (shown in Figure 10b) further divides into 2a and 2b. The system might need to initiate a new S stream, depending on whether the pause terminates before or after an ongoing S stream has reached the play point. As soon as the buffer fills, a reservation request goes out for a new S stream, the contents are purged from the synch buffer, and SAM splits the user from the original S stream. In case 2a, the pause terminates before an ongoing S stream has reached the play point, and a new S stream (the one reserved earlier) will be initiated to serve the user. Otherwise, we have case 2b.

Quite possibly no S streams are available in the system. The user's request will then join an FCFS queue to wait either for the next available S stream or until an ongoing S stream has reached the play point, at which time the user 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 play before the pause operation terminates. In Figure 10b, since t corresponds to six segments, the corresponding play point will arrive after six segments of the original stream have arrived. Since the buffer is of size 4, it can only store four segments—3, 4, 5, and 6. The protocol releases the reserved S stream and merges (switches) the user to the new stream as soon as the synch buffer becomes full. The new stream, starting at the play point, now feeds the synch buffer. In cases 1b and 2b, since the user remains in the pause mode after merging with the targeted ongoing S stream, the procedure must repeat, perhaps multiple times, until the user terminates the pause operation.

This discussion assumes that an S stream feeds the user directly. Let us now study the situation in which the synch buffer, fed by an S stream, serves the user—the user issued an interactive operation earlier. Figure 11 illustrates how SAM handles pause and resume in this situation. Again, this example assumes the system has allocated a synch buffer of size four to each user. Suppose the original S stream in Figure 10 serves the user accessing video *i*. Note that this virtual S stream is fed by an ongoing S stream that started two segments earlier. In other words, two segments of the synch buffer have already been used to synchronize with this feeding S stream, leaving only two segments available. This marks the only difference between how SAM handles the pause operation when the user is served directly versus through 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 $t_d = 2$ in our example, $SB' = 2$ and $t = 2$. The operation is described by the same flow chart (Figure 9) as when a real S stream feeds the user. The only difference is that the synch buffer size equals SB' instead of SB. In addition, a user switching to a new stream (case 1b and case 2b) can discard the existing contents in the synch buffer, effectively augmenting the synch buffer to the maximum value of SB again.

Variations of the basic scheme

Now let's consider variations of the basic SAM protocol.

No initial batching delay

As soon as the system receives a video request, it determines whether at least one eligible S stream exists. If so, an I stream serves the request immediately and the eligible S stream feeds the synch buffer for an interval t_{os}, equal to the time offset between the new request and the eligible S stream. After t_{os}, this video request effectively merges into the ongoing S stream by serving the user from the synch buffer. If no eligible S streams exist, the system immediately initiates a new S stream to serve the request.

Adjustable batching intervals

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

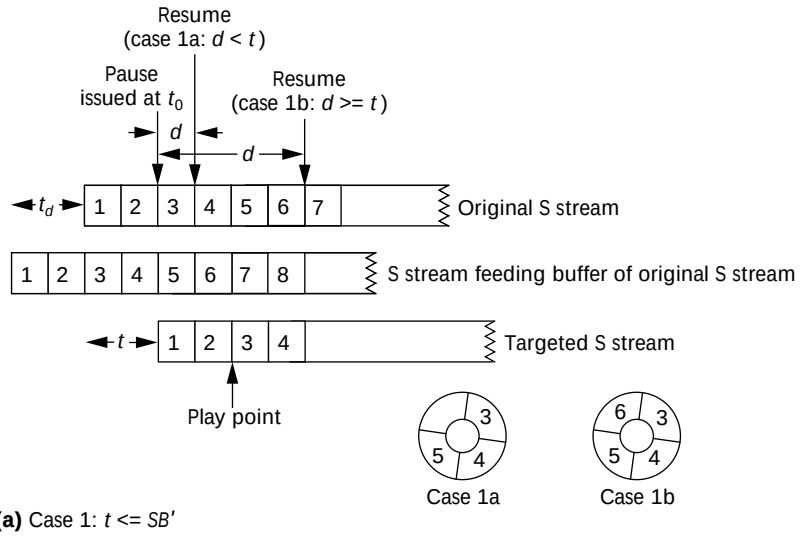
A variation of the basic batching idea

Instead of the system initiating a timer when the first request in a batch arrives, it divides time into fixed length intervals, say every five minutes. If at least one request for video *i* arrives during an interval, the system initiates an S stream at the end of that interval. This has the advantage that, for unpopular videos, the average wait equals half of the batching interval instead of the whole batching interval. Users will not notice much difference for popular videos.

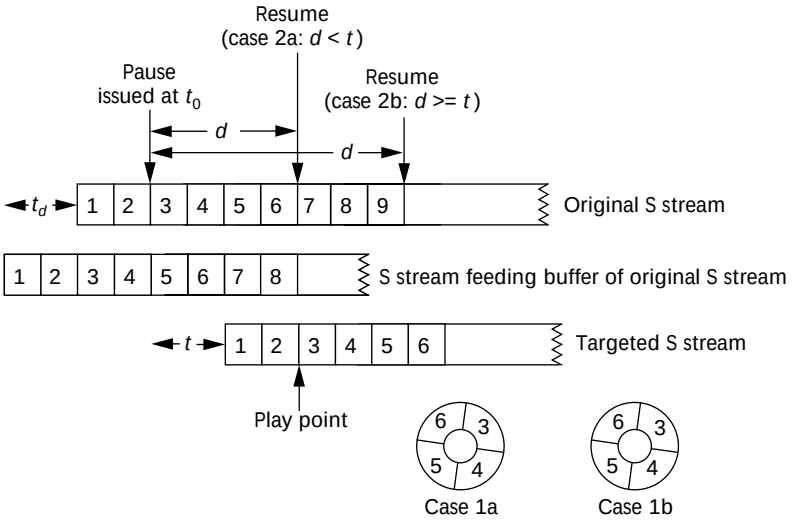
One synch buffer for multiple virtual streams

In the basic scheme, when the split and merge operations are required, the system only examines real ongoing video streams. Most likely the users can merge into virtual streams. SAM uses a synch buffer to create a virtual stream from an ongoing real stream. Each virtual stream offset in time from the real stream needs its own synch buffer. To reduce the total amount of synch buffer required, the same synch buffer may serve multiple virtual streams.

For example, virtual stream B may be one minute behind real stream A, while virtual stream C may be two minutes behind real stream A. We can have stream A feed a synch buffer that will in turn feed both streams B and C. As a result, a synch buffer may have one input stream but multiple output streams serving different users with different time offsets.



(a) Case 1: $t <= SB'$



(b) Case 2: $t > SB'$

Figure 11. Illustration of the pause operation when the user operates from the synch buffer.

A pricing mechanism for user interactivity

The more a user will pay, the closer the user gets to true VoD service. The high end imposes no initial batching delay and allows full user interactivity. Medium-cost service yields an initial batching delay and full interactivity. Finally, the least-expensive service level buys users an initial batching delay and limited interactivity.

Numerical results

This section presents simulation results generated by a C program running on a Hewlett-Packard C160. We simulated 24 hours of real time, requiring on average 30 minutes of CPU time for each arrival rate. We used the following system parameters: 30 available videos, each lasting 120 minutes, with the probability a particular movie

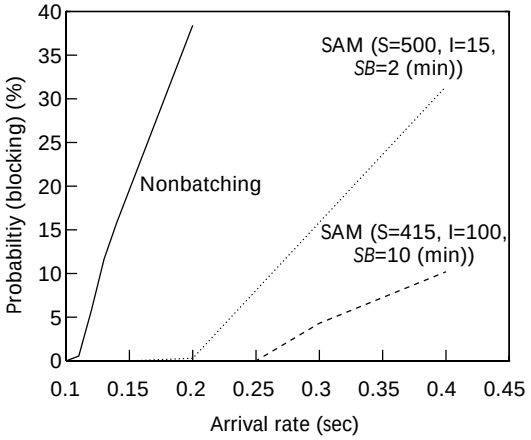


Figure 12. Blocking probability for the batching and nonbatching cases.

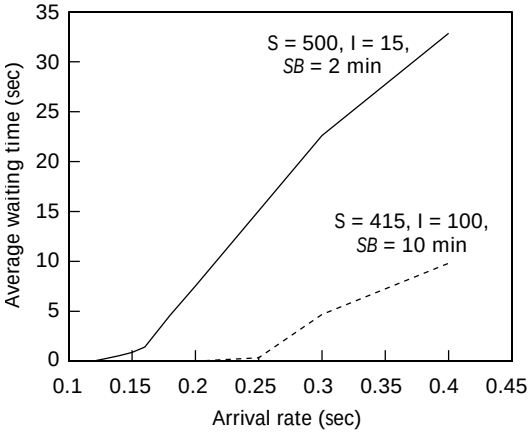


Figure 14. Average interaction delay as a function of arrival rates.

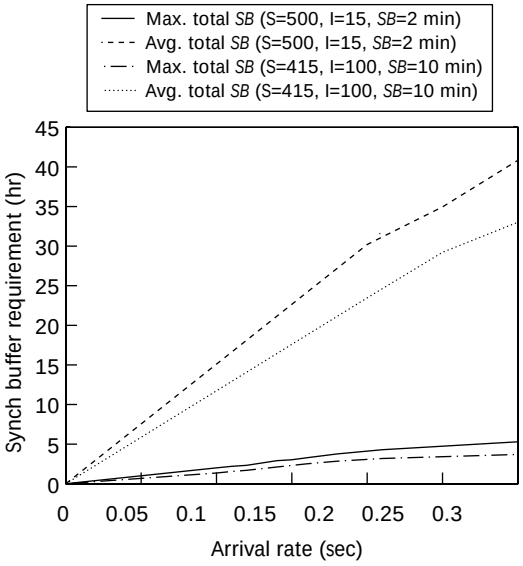


Figure 13. Average amount of synch buffer required as a function of arrival rates.

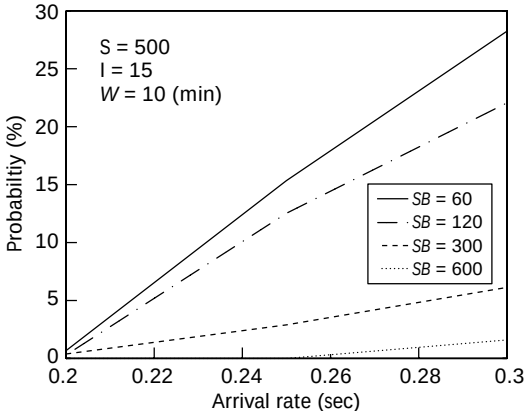


Figure 15. Blocking probability as a function of synch buffer allowed.

is accessed (p_i where i = 1, 2, …, 30) following the Zipf distribution.[8] Therefore, video 1 is the most popular, followed by video 2, and so on.

We also assume the following user activity model: A user starts in normal play mode and stays there for an exponential amount of time with a mean of 30 minutes. Then the user goes to interaction mode with probability 0.75 and quits with probability 0.25. In interaction mode, the user 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 or rewind is exponentially distributed with a mean of 0.5 minutes (the user holds down the fast-forward or rewind button for an average of 0.5 minutes). Fast-forward and rewind take the user to a point in the video offset from the original point by an interval uniformly distributed between 1 and 90 seconds. Each jump operation lasts one second and takes the user to a point in the video offset from the original point by an interval uniformly distributed between 1 and 1,000 seconds. After a user interaction, the user resumes normal play mode. This cycle repeats 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 1 to 10 minutes.

Note that because existing batching approaches only allow near-VoD services, in our simulation we compare SAM with the nonbatching approach only. For fair comparison, in the nonbatching case we use all of the 515 available video streams to serve users in a dedicated fashion. Each user also follows the same user activity model.

Figure 12 shows how the blocking probability

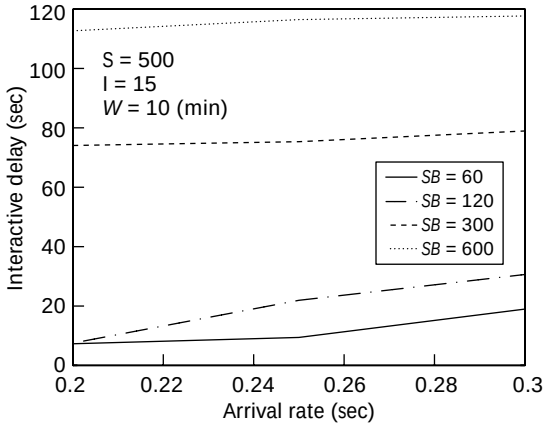


Figure 16. Average interaction delay as a function of synch buffer allowed.

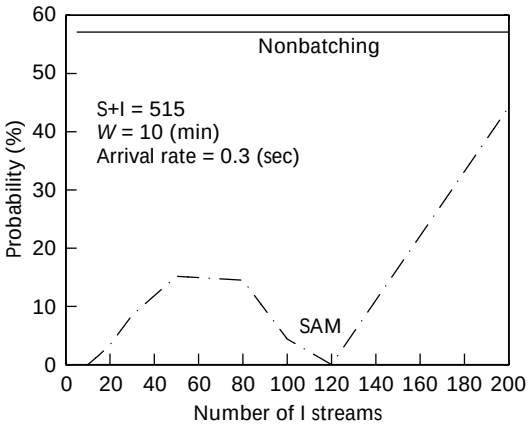


Figure 17. Blocking probability as a function of number of I streams.

changes as the video request rate increases for the cases of batching (SAM) and nonbatching (point-to-point connection for each user). As expected, a large reduction in the blocking probability occurs with SAM. (Actually, the improvement for popular videos is much more dramatic than indicated, since the figure shows the reduction in blocking probabilities averaged over all videos.) The price we pay is the initial batching delay, which is bounded by 10 minutes and averages 5 minutes.

In addition, we need a synch buffer, and the user may experience some interaction delay. Figure 13 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 8.3 seconds per user when the arrival rate is 0.3 requests per second, for $S = 500$, $I = 15$, and $SB = 2$ minutes. (Note that 0.3 requests per second translates to $0.3 \times 120 \times 60 = 2{,}160$ average number of users.) Figure 14 shows the average interaction delay—acceptable except in a highly loaded situation. (To ensure that the blocking probability remains small, we would not operate the system at such a high load anyway.) This leads us to conclude that our proposed SAM protocol makes an excellent VoD video delivery protocol.

We also investigated how the blocking probability (see Figure 15) and the interaction delay (see Figure 16) change as the maximum synch buffer allowed for each user changes. As expected, the blocking probability decreases as the maximum synch buffer allowed increases. At the same time, the interaction delay increases because more users are admitted into the system.

Finally, we studied the effect of increasing the number of I streams while keeping the total number of streams constant: on the blocking probability (see Figure 17), the synch buffer required (see Figure 18), and the interaction delay (see Figure 19). For the system parameters specified in

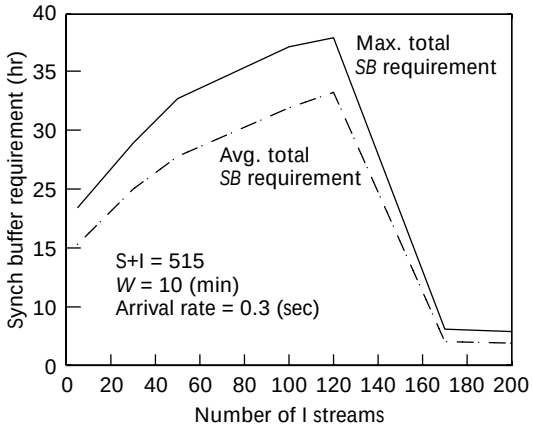


Figure 18. Average amount of synch buffer required as a function of the number of I streams.

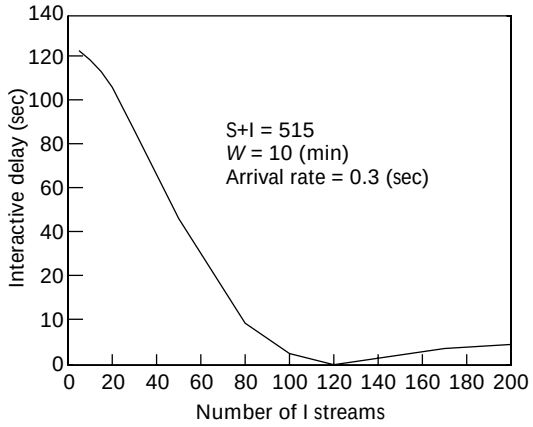


Figure 19. Average interaction delay as a function of the number of I streams.

the figures, we find that the blocking probability and the interaction delay are minimized with 120 I streams, at which time the synch buffer required reaches its maximum.

Conclusions

VoD will be one of the most important commercial applications of distributed multimedia systems. It provides an electronic video rental service, which gives users the ultimate flexibility in selecting any video programs, at any time, and in performing any VCR-like user interactions.

To achieve commercial success, however, VoD must be priced competitively with existing video rental services. Approximately half of video rental revenues go to the program providers. That means the other half goes toward the cost of delivering the video and for the service provider's profits. In existing video rental stores, the users bear the major cost of delivery, and the service provider incurs only the costs of shelf space. For VoD, the costs of video delivery include the costs of the high-capacity video server and the high-speed network, both substantial. Our proposed protocol lets multiple users share the same video stream, dramatically increasing the capacity of the system and greatly reducing the costs per user. At the same time, the price—batching delay, interaction delay, and so forth—remains tolerable. This leads us to conclude that our proposed SAM protocol makes an excellent candidate for deployment in interactive VoD systems. MM

Acknowledgments

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

References

1. T.S. Perry, "The Trials and Travails of Interactive TV," *IEEE Spectrum*, Vol. 33, No. 4, Apr. 1996, pp. 22-28.
2. A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. ACM Multimedia 94*, ACM Press, New York, 1994, pp. 15-23.
3. R.O. Banker et al., "Method of Providing Video-on-Demand with VCR-like Functions," *U.S. Patent 5,357,276*, 1994.
4. K.C. Almeroth and M.H. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE J. Selected Areas in Comm.*, Vol. 14, No. 6, Aug. 1996, pp. 1110-1122.
5. 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," *ACM/Springer Multimedia Systems*, Vol. 3, No. 4, 1995, pp. 137-150.
6. V.O.K. Li et al., "Performance Model of Interactive Video-on-Demand Systems," *IEEE J. Selected Areas in Comm.*, Vol. 14. No. 6, Aug. 1996, pp. 1099-1109.
7. L. Golubchik, J.C.S. Lui, and R.R. Muntz, "Reducing I/O Demand in Video-on-Demand Storage Servers," *Proc. ACM SIGMetrics*, ACM Press, New York, 1995, pp. 25-36.
8. D. Knuth, "The Art of Computer Programming," *Sorting and Searching,* Addison-Wesley, New York, Vol. 3, 1973.

Wanjiun Liao is an assistant professor in the Electrical Engineering Department at National Taiwan University, Taipei, Taiwan. Her research interests include multimedia communications, networking, and distributed database systems. She received her BS and MS from National Chiao Tung University, Taiwan, in 1990 and 1992, respectively, and a PhD in electrical engineering from the University of Southern California, Los Angeles. A member of the IEEE and the Phi Tau Phi scholastic honor society, she is a recipient of the Acer Long-Term thesis award, the Chinese IEE (CIEE) graduate student thesis award, and the outstanding research paper award at USC. She was recently selected by *EE Times* (Oct. 1997) as an Outstanding Young Electrical Engineer.

Victor O.K. Li is Chair Professor of Information Engineering at the Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong, and Managing Director of Versitech, the University Contract Research Company. His research interests include high-speed communication networks, personal communication networks, and distributed multimedia systems. He received his SB, SM, and ScD in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, Massachusetts, in 1977, 1979, and 1981, respectively. He was elected an IEEE Fellow in 1992.

Readers may contact Li at the Dept. of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, e-mail vli@eee.hku.hk.