Supporting Interactive Video-on-Demand With Adaptive Multicast Streaming

Ying Wai Wong, Jack Y. B. Lee, *Senior Member, IEEE*, Victor O. K. Li, *Fellow, IEEE*, and Gary S. H. Chan, *Senior Member, IEEE*

***Abstract*—Recent advances in multicast video streaming algorithms have opened up new ways to provision video-on-demand services to potentially millions of users. However, the spectacular efficiency of multicast streaming algorithms can only be realized by restricting or even prohibiting interactive playback control. Experiments reveal that the performance of current state-of-the-art multicast streaming algorithms will degrade significantly even at very low levels of interactivity (e.g., one control per five users). This study tackles this challenge by investigating the fundamental limitations of multicast streaming algorithms in supporting interactive playback control and presents a general solution—*static full stream scheduling* (SFSS)—which can be applied to many of the existing multicast streaming algorithms to substantially improve their performance when interactive playback control is to be supported. Moreover, to solve the problem of optimizing the algorithm for the often unknown client access patterns (e.g., arrival rates and interactivity rates), we present a novel *just-in-time simulation* (JTS) scheme to dynamically and automatically tune operating parameters of the SFSS algorithm while the system is online. This JTS scheme not only eliminates the need for *a priori* knowledge of the often unknown system parameters, but also can adapt to changes in the client access pattern over time. Extensive simulation results show that the proposed adaptive algorithm can reduce the admission and interactive control latencies by as much as 90%.**

***Index Terms*—Embedded simulator, interactive playback control, just-in-time, multicast streaming, video-on-demand (VoD).**

I. Introduction

THE provisioning of large-scale video-on-demand (VoD) services has attracted much attention in recent years. In current VoD systems, whenever a new client starts a video session, a dedicated stream is allocated to serve the user till the end of the viewing session. Video data are transmitted to the client in a point-to-point manner, known as *unicast* streaming. In unicast streaming, clients can control playback of the video at will, such as performing pause/resume and seeking (i.e., change to a new playback position). However, as unicast streaming requires separate streaming bandwidth for each and every client, the server and network bandwidth resources consumed will inevitably grow linearly with the user population. Therefore, unlike TV broadcasters, VoD service providers cannot benefit from the economy-of-scale in serving large user populations. Worse still, the need for specialized high-capacity streaming servers and network equipment often increases the per-client cost when scaling up such a system.

In response to this challenge, researchers have recently begun to investigate the use of network multicast for video streaming. Unlike unicast transmission, a *multicast* video stream can be shared by more than one receiver. The network switches/routers will automatically replicate the multicast data for multiple receivers without adding any extra streaming workload at the video server. Over the last decade researchers have developed a number of innovative algorithms/protocols to take advantage of network multicast to significantly reduce the resources required for video streaming. Notable examples include batching [1]–[5], patching/stream merging [6]–[13], piggybacking [14]–[17], bandwidth skimming [18], and periodic broadcasting [19]–[22]. These techniques can also be further combined to form even more efficient architectures [23]–[27].

With multiple clients sharing a multicast video stream, individual client thus cannot alter the playback sequence (e.g., performing a seek to another playback location) without affecting other clients sharing the same data stream. One solution is to dynamically allocate a separate interactive video stream to perform interactive playback control and then merge the client back to an existing multicast video stream afterwards (e.g., the Split-and-Merge Protocol first pioneered by Liao and Li [28], [29]). Our simulation study of a number of recently proposed multicast streaming algorithms [9]–[11], [13] modified to support interactive playback control reveals that performance of these algorithms will degrade significantly even at relatively low levels of interactivity (e.g., mean access latency increases from 0.11 s with no interactivity to 698.28 s with only 0.39 interactive control per viewing session for the Dyadic algorithm [11], [13]). At higher levels of interactivity, the performance degrades so much so that the performance differences between different streaming algorithms diminish as most of the system resources are then used to support interactive playback control instead of serving new clients.

This study tackles this challenge by investigating the fundamental limitation of multicast streaming algorithms in supporting interactive playback control. We present a general solution—*static full stream scheduling* (SFSS)—which can

Manuscript received November 25, 2004; revised February 23, 2006. This work was supported in part by a Direct Grant and Earmarked Grant CUHK 4328/02E from the Hong Kong Special Administrative Region (HKSAR) Research Grant Council and in part by the Area of Excellence Scheme, established under the University Grants Committee of the HKSAR, China under Project AoE/E–01/99. This paper was recommended by Editor-in-Chief C. W. Chen.

Y. W. Wong and J. Y. B. Lee are with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: yblee@ie.cuhk.edu.hk).

V. O. K. Li is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: vli@eee.hku.hk).

G. S. H. Chan is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong (e-mail: gchan@cse.ust.hk).

Digital Object Identifier 10.1109/TCSVT.2006.888839

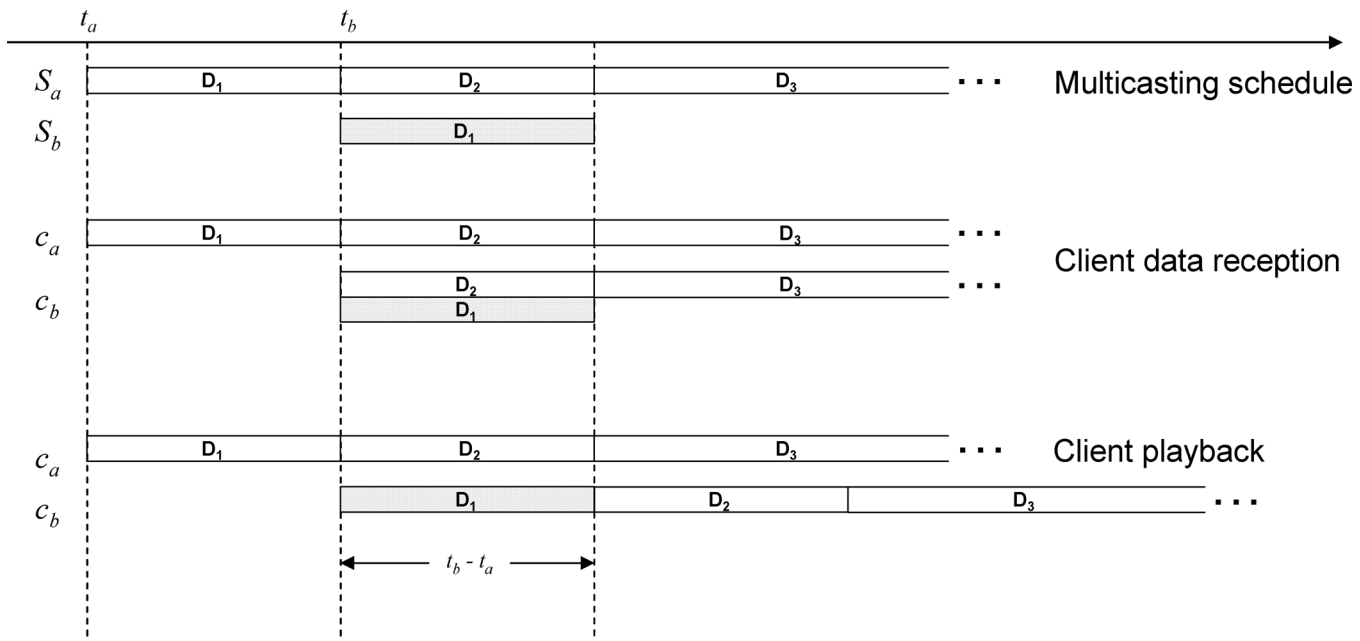


Fig. 1. Merging clients using patching and caching.

be applied to many of the existing multicast streaming algorithms to improve their performance when interactive playback control is to be supported. Moreover, to solve the problem of optimizing the algorithm for the often unknown client access patterns (e.g., arrival rates and interactivity rates), we present a novel *just-in-time simulation* (JTS) scheme to dynamically and automatically tune operating parameters of the SFSS while the system is online. This JTS scheme not only eliminates the need for *a priori* knowledge of the often unknown system parameters, but also can adapt to changes in the client access pattern over time.

The remainder of this paper is organized as follows. Section II reviews some previous works on multicast streaming algorithms and existing approaches to support interactive playback control; Section III presents a client interaction model and studies the performance impact of interactive playback control on the existing multicast streaming algorithms; Section IV presents the general SFSS technique and its applications to some existing multicast streaming algorithms; Section V presents the JTS scheme to address parameter estimation and performance optimization issues when applying the SFSS scheme in practice; Section VI presents simulation results to evaluate the performance of SFSS; and Section VII summarizes the study.

II. Background

In this section, we first review some current state-of-the-art multicast streaming algorithms and then review the existing methods to support interactive playback control in some of these algorithms.

A. Multicast Streaming Algorithms

There are two fundamental operations, namely patching and caching, common in most multicast streaming algorithms. When a client is admitted to the system, there may be one or more ongoing multicast video streams transmitting the requested video title. This newly admitted client can cache one or more of these multicast video streams to share the transmitted data—caching, but will not be able to begin playback as it has missed the initial portion of the video stream. To tackle this problem, the client can request another video stream to transmit the missing initial video portion to enable playback to begin—patching.

Fig. 1 illustrates the process of admitting new clients through patching and caching. Two clients, denoted by c_a and c_b, arrive at the system at time t_a and t_b, respectively, requesting the same video. To facilitate discussion, we divide the whole video into three logical segments D_1 to D_3, and use $[D_i, D_j]$ to denote the video data from the ith to the jth segment. We assume that the video being requested is of length L seconds encoded at a constant bit rate of R bps.

Assuming the system is idle when c_a arrives, the server will allocate a video stream to transmit to client c_a the whole video from the beginning to the end (i.e., $[D_1, D_3]$). We call this video stream a *full stream*, denoted by S_a. The cost of serving c_a is thus equal to the bandwidth-duration product *LR* bits. At time t_b, client c_b arrives and caches video data from S_a. As it has missed the initial $(t_b - t_a)$ seconds of the video (i.e., D_1), the system will need to initiate a new stream S_b, called a *partial stream*, to stream the missed portion D_1 to client c_b to enable it to begin playback (i.e., patching). Client c_b can then simultaneously cache subsequent video data $[D_2, D_3]$ from S_a for the rest of the video session (i.e., caching). In this way, the cost of serving clients c_a and c_b is reduced from 2*LR* bits as in a unicast-based VoD system to $(L + (t_b - t_a))R$ bits. The tradeoffs are the need for the client to receive two video streams simultaneously and the additional buffers needed to store the cached data for later playback.

For clients arriving after t_b, they can also be patched and eventually merged back to the full stream as long as they arrive before the full stream (S_a) ends. We can view this patching and

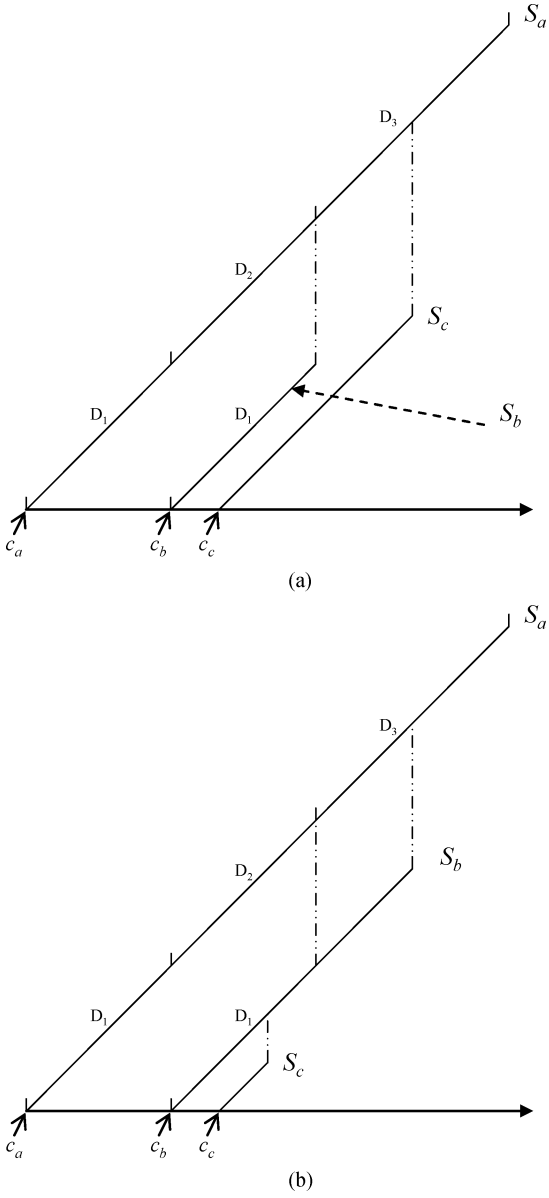


Fig. 2. Merge tree for patching and caching. (a) S_c attaches to S_a. (b) S_c attaches to S_c.

caching process as a *merge tree* with the full stream S_a as its root and the partial streams (e.g., S_b) as its leaves as shown in Fig. 2.

In Fig. 2(a), a later client c_c is admitted into the system by caching data from S_a. In other words, the assigned new stream S_c is attached as a child of the full stream S_a in the tree. Note that in general a client is not limited to caching data from only a full stream, but also ongoing partial streams as well. In our case, as the client c_c enters the system before the partial stream S_b ends, it can also cache video data from S_b as shown in Fig. 2(b). In this case client c_c is assigned to cache data from S_b while playing back data received from S_c. After that it follows the tree structure to switch to cache data from S_a in order to merge back to the full stream S_a. This may enable further resource reduction because to the client c_c the duration of initial video data missed is less when caching from the partial stream S_b instead of from the full stream S_a.

As Fig. 2(a) and (b) illustrates, there are in general more than one way to admit a new client, i.e., there are different paths to reach the root of the merge tree from the leaves. Different multicast streaming algorithms will have different ways to select the path to reach the root of the merge tree, and consequently, with different performance and tradeoffs. We review in the following some state-of-the-art algorithms in multicast video streaming.

Shi and Kuo [9] proposed two multicast streaming algorithms, called Controlled Greedy Recursive Patching (CGRP) and Cost-Aware Recursive Patching (CARP). A new partial stream is always started when admitting a client. In CGRP, a new client is assigned to cache data from the latest *reachable* stream – defined as the partial stream already in the merge tree that terminates after the newly started partial stream; and in CARP a new client is inserted at a point in the merge tree so as to minimize the additional server bandwidth consumed.

In another study [11], [13], Coffman *et al.* proposed the Dyadic algorithm for merging streams. Assume a full stream S_f was started at time t, then all subsequent clients arriving in the time interval $[t, t + L/2)$ will join the tree with S_f as the root. To construct the tree, the time interval is further divided into Dyadic intervals by the time instants $t + L/(2r^i)$, with $i = 1, 2, \ldots$, and r being the Dyadic ratio. The earliest stream in each Dyadic interval then becomes a child of S_f. The same procedure is then applied recursively to each Dyadic interval until all clients are assigned.

In the previous three algorithms, the patching and caching schedules, i.e., the merge tree, are assigned and fixed upon admission. In another study [10], Eager *et al.* proposed an Earliest Reachable Merge Target (ERMT) algorithm that relaxes this restriction and allows even ongoing patching and caching schedules to be modified when admitting new clients, thereby enabling further performance improvements.

To illustrate the efficiency of multicast streaming algorithms, we plot in Fig. 3 the access latency—defined as the mean time a client has to wait before playback can begin (ignoring network delays and prefetch buffering delays), versus client arrival rates ranging from 0.01/s up to 0.1/s for the four previously mentioned algorithms. Note that for the Dyadic algorithm we use a Dyadic ratio of 1.62 instead of 2.0 proposed in the original study [11], [13] according to the recent study by Bar-Noy *et al.* [12]. There are a total of ten streaming channels available at the server. Once all channels are occupied, subsequent arriving requests will have to wait until a server channel is released. We can observe that all four multicast streaming algorithms are very efficient, achieving relatively short latency even at heavy arrival rates. For example, with a video length of 7200 s, an arrival rate of 0.1 client/s will require on average 720 streaming channels in a unicast-based VoD system to avoid overload. Using just ten channels these multicast streaming algorithms manage to reduce the resource requirements by as much as 98%.

However, these rather spectacular performance gains are achievable only if the clients are not allowed to perform any interactive playback control. In particular, common to the Dyadic, CARP, CGRP, and many other multicast streaming algorithms, a configurable system parameter denoted by W—*full-stream restart threshold*, is used to control the minimum duration between the allocations of two consecutive full streams. De-

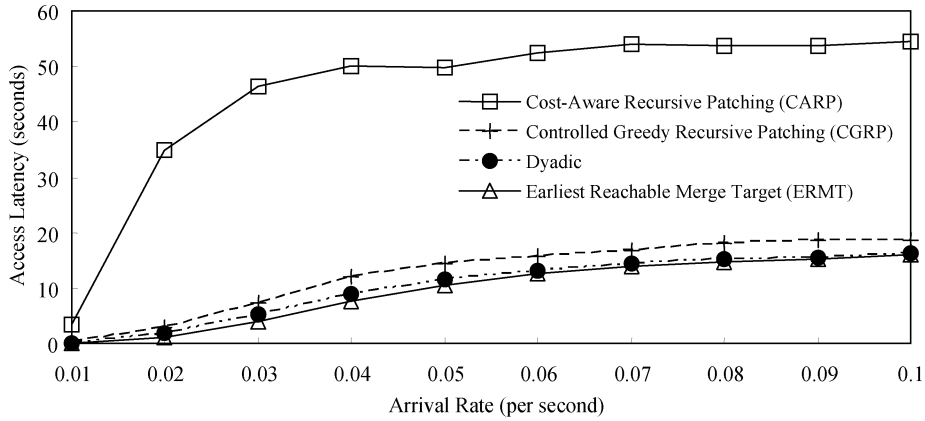


Fig. 3. Performance of different multicast streaming algorithms under zero user interactivity.

creasing W will generate full streams more frequently, thereby dividing the clients into larger number of independent merge trees. Not surprisingly, the choice of W has significant impact on the system's performance and each of the previously discussed algorithms has its own way to set the threshold W. However, in Section III-D we show that these thresholds are no longer valid when interactive playback control is supported, and performance of these multicast streaming algorithms will degrade dramatically.

B. Interactive Playback Support

In a unicast video streaming system, each video stream is dedicated to one client and any interactive playback operation can be supported by simply adjusting the video content streamed to the client. By contrast, this cannot be accomplished in the same way in a multicast video streaming system without affecting playback of other clients sharing the same multicast video stream.

One way to alleviate this problem is to support only discontinuous interactive playback control. Unlike continuous playback controls, where a client can change the playback point to any point within the video, discontinuous playback control allows the client to change to only a limited number of playback points. For example, Almeroth and Ammar [30] considered a staggered multicast video streaming system with consecutive video streams offset by K seconds. In this system, clients are allowed to change the current playback point to a new playback point iK ($i = 1, 2, \ldots$) seconds away in either forward or reverse direction. Similarly, the client can pause for durations only in multiples of K seconds. With these constraints, interactive playback operations can then be implemented simply by switching to another multicast stream that is offset by iK seconds. Additionally, if the client has the buffering capacity, it can also implement continuous pause of any duration simply by buffering the incoming video data for later playback.

Clearly, while discontinuous playback controls do not consume extra resources, the limited number of seekable playback points will significantly degrade user experiences. To achieve true VoD it is therefore necessary to support continuous playback controls. One solution is to initiate dedicated interactive streams to support interactive playback control and to merge it back to an existing multicast video stream with the patching algorithm. This approach was first proposed by Liao and Li in their Split-and-Merge (SAM) protocol for VoD systems employing batching [28], [29], and later on studied by Abram-Profeta and Shin [31]. Specifically, a break-away client after performing an interactive playback control is treated as a new client arrival, albeit not necessarily starting video playback from the beginning. The break-away client uses the patching stream to sustain video playback while caching video data from an ongoing multicast stream that the client eventually merges back to.

We illustrate the merging process with an example. At some time after admission, a client c_m may issue a forward seeking (FSEEK) or backward seeking (BSEEK) request to an offset t_m seconds relative to the beginning of the video. As there are other clients receiving data from the same multicast stream, the client c_m has to break away from the multicast stream and find a full stream S_k that has its current multicasting point p_k ($p_k > t_m$) nearest to t_m. Then the client simply merges back to this full stream S_k by patching and caching.

Fig. 4 illustrates this process. The full stream S_k has started for p_k seconds and is about to multicast the data block D_{m+1}. In the first phase which lasts for $(p_k - t_m)$ seconds, c_m resumes video playback from point t_m with video data D_m received from a newly initiated partial stream S_m. At the same time, c_m buffers data D_{m+1} from S_k. After $(p_k - t_m)$ seconds, patching ends and S_m is released. The client c_m continues its video playback with data received from S_k until another interactive playback control is requested. In the process, a buffer large enough to cache $(p_k - t_m)$ seconds of video data is required. In case an eligible full stream S_k cannot be found, a partial stream will be started to transmit video data from t_m until the end of the video.

The two studies differ in that Abram-Profeta and Shin [31] assumed the client has the buffer for caching video data while in the SAM protocol [28], [29] the buffer can be located either in the client or in a proxy shared by multiple clients. Alternatively, the server can also initiate a new video stream at twice the video bit rate to enable the client to catch up with an ongoing multicast video stream [32].

In a more recent study, Ma and Shin [33] proposed an even more sophisticated algorithm called *Best-Effort Patching* (BEP) that uses a dynamic algorithm to merge clients after interactive

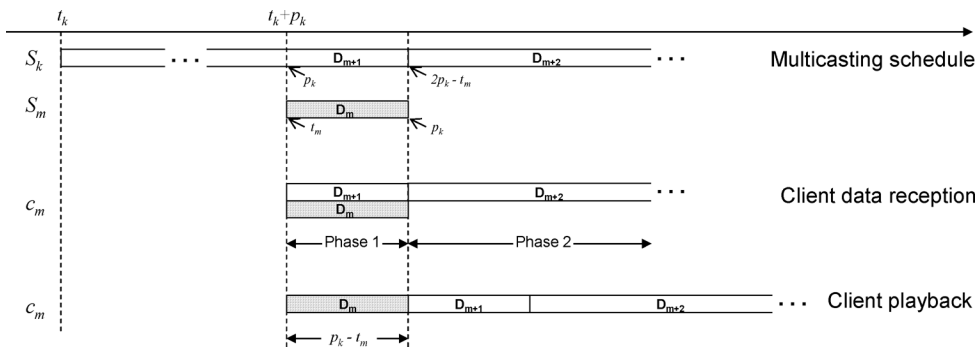


Fig. 4. Illustration of merging a break-away client that performed an interactive playback control.

playback operations. This dynamic algorithm enables clients to cache video data from more than one stream throughout the whole merging process and thus can achieve further network resource reduction compared to simple patching and caching.

Finally, researchers have also proposed other techniques to reduce the resources required in implementing interactive playback control, including trading off seeking precision [30], [34] and video quality [35]. These techniques are orthogonal to this study and thus can be combined to yield further resource reductions. We refer the interested readers to the original studies for more details [30], [34], [35].

III. Interactive Multicast Streaming

While researchers have proposed algorithms and techniques to support interactive video playback in multicast streaming algorithms, so far there is no systematic study on the performance degradation (or increase in resource requirements) that results from supporting interactive playback control. Moreover, the recent advances in multicast streaming algorithms (e.g., [9]–[11], [13]) have significantly increased the efficiency of multicast video streaming systems and thus the performance impact of supporting interactive playback control may be further amplified.

In this section, we investigate this issue by first defining a user interactivity model for issuing interactive playback control requests. Next we discuss how admission and interactive requests are scheduled and then present simulation results to evaluate the performance impact of supporting interactive playback control on the current state-of-the-art multicast streaming algorithms.

A. Interactivity Model

Interactive playback operations are commonly modeled after their counterparts in video cassette recorders (VCRs). Common playback controls include pause/resume, slow motion, frame stepping, fast forward/backward visual search, as well as forward/backward seeking. Over the years, researchers have devised a number of user interactivity models [31], [36], [37] for different applications.

For example, Branch *et al.* [37] reported user access statistics collected from a multicampus interactive educational resource system. They showed that interactive behavior can be adequately modeled by an exponential distribution, while the lognormal distribution gives an even closer match to the real statistics. Since these statistical results are collected from applications where frequent repeated viewings of certain sections are common (e.g., educational materials), they may not be suitable for other applications such as entertainment contents and movie viewing. In another study, Li *et al.* [36] devised a two-state model in which users are either in the NORMAL or the INTERACTION state, and stay in that state for a random period of time that is exponentially distributed. In a third study, Abram-Profeta *et al.* [31] distinguished between different states and devised a multi-state interaction model. We adopt this model in this study to form the model depicted in Fig. 5.

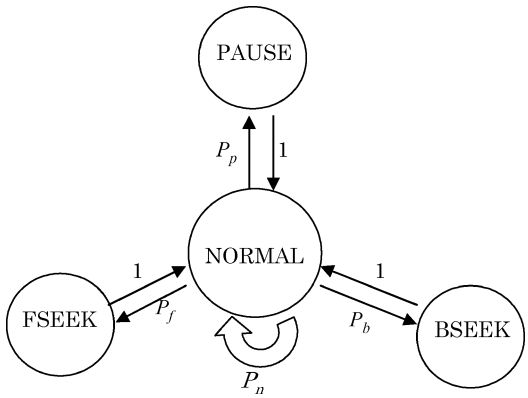


Fig. 5. Client interaction model.

Beginning at the NORMAL state, the client will transit to the NORMAL, PAUSE, FSEEK, and BSEEK states with probability P_n, P_p, P_f, and P_b, respectively (i.e., $P_n+P_p+P_f+P_b = 1$). Once in the NORMAL or PAUSE state, the client will stay there for a random duration that is exponentially distributed with mean $\mu_{\rm sd}$ seconds. For FSEEK/BSEEK states, the client will perform a random seek to a new position with a *seek distance* exponentially distributed with mean $\mu_{\rm sd}$ seconds. This interactivity model can capture several characteristics of user interactions, including the level of client interactivity, the relative frequency of individual control requests, and the duration of interactive playback control.

It is worth noting that it is possible to implement a richer set of interactive functions using the basic interactive controls such as FSEEK and BSEEK as basic building blocks. For example, one can implement two-way interactive video streaming in the sense

that the user can directly or indirectly choose between alternative sequences of video segments, i.e., different developments of the same story, to be played back. This feature can be implemented by concatenating all alternative video sequences as a single stream for periodic multicast, and then invoke FSEEK or BSEEK to jump to the desired video sequence to continue playback based on the user's interactive inputs.

On the other hand, instead of allowing the user to actively choose the video sequences for playback, the user can let the system choose the particular video sequences to playback base on popularity or even randomly. These rich interactive functions will open up new dimensions to the VoD experience.[1] The challenge, obviously, will be the capability to support interactive playback efficiently in a large-scale VoD system.

B. Request Scheduling

In interactive multicast streaming, there are two types of streaming requests, namely *admission requests* generated by newly arrived clients; and *merging requests* generated by clients performing interactive playback control. Intuitively, to the end users admission delay is far more tolerable than delay in performing interactive playback control. To account for this observation, we can separately place admission requests in an admission queue and merging requests in a merging queue. Then we give priority to waiting requests in the merging queue whenever server resources become available. However, this approach suffers from a significant problem.

Specifically, if merging requests have absolute priority over admission requests, the available resources will be dominated by them at high system load. Now as merging requests generate only partial streams, the system will eventually run out of full stream to serve as the root of merge trees for admitting new clients and merging break-away clients. Consequently, the break-away clients will not be able to merge back to an ongoing full stream and thus cannot release their assigned stream for reuse by other clients, leading to inefficient resource utilization.

Therefore, instead of simply giving absolute priority to merging requests, the system will try to schedule a full stream every W seconds, where W is the *full-stream restart threshold*. In particular, whenever a streaming channel becomes available, the server will allocate it as a new full stream (and admitting waiting admission requests in the process) if the time since the last full stream allocation is equal to or longer than W seconds. Otherwise the available streaming channel will be allocated first to waiting merging requests, and then to admission requests if the merging queue is empty.

This request scheduling algorithm together with the merging algorithm for break-away clients can be applied to the Dyadic, CARP, and CGRP algorithms. For ERMT, as it has no equivalent parameter as the full-stream restart threshold W, the above algorithm is not directly applicable. Instead, we modify the ERMT algorithm to give priority to admission request if the time lapse since the last full stream generation is equal to or longer than W seconds. This prevents the server channels from being monopolized by merging requests. For simplicity, we will refer to these modified versions of Dyadic, CARP, CGRP, and ERMT by their original names in the rest of the paper.

[1]The authors wish to thank the anonymous reviewer for this idea.

C. Client Buffer Management

During patching and caching, a client will receive data at a rate of $2R$ bps while playing back video at R bps. Thus, video data will accumulate in the client buffer at a rate of R bps. For the Dyadic, CARP, and CGRP algorithms, a client arriving t seconds after the last full stream will need to perform patching and caching for t seconds, buffering tR bits of video data in the process. Now given that a new full stream is started whenever the time lapsed since the last full stream exceeds W seconds, the maximum duration of patching and caching process in the interactive Dyadic, CARP, and CGRP algorithms is W seconds and thus they require a client buffer size of WR bits. As the parameter W does not exist in ERMT, we cannot control the starting frequency of full streams. To determine the maximum client buffer requirement, we note that video data accumulate in the client buffer at R bps when a client is performing patching and caching. During that phase, t seconds of video is received in $t/2$ seconds. Since $t \leq L$ the maximum buffer size required is equal to half of the video, i.e., $RL/2$ bits.

In case the client buffer size is limited to say B_cR bits, the maximum duration of patching and caching during admission will then be limited to B_c seconds, which implies that the parameter W is limited by B_c (i.e., $W \leq B_c$). On the other hand, due to server bandwidth constraint, consecutive full streams may be offset by more than W seconds, so is the duration of patching and caching during the merging process of break-away clients after interactive playback operations. Thus, we have to impose an additional constraint of $(p_k - t_m) \leq B_c$ on the merging operation described in Section II-B. In other words, if the constraint is not satisfied then a partial stream will be started to serve the merging client till the end of the video. In this case the stream is held for a much longer duration, and thus will incur heavier load on the system when compared to merging by patching and caching. Nevertheless this scenario only occurs at extremely low arrival rates when the time offset between two consecutive full streams is usually much larger than W seconds.

With client buffer we can further optimize the PAUSE operation. Specifically, instead of immediately breaking away from the video stream, once a PAUSE operation is executed, the client can continue to receive and cache video data at a rate of $2R$ bps if patching and caching is in progress or R bps otherwise. Thus, with a buffer of B_cR bits, no matter whether a client is in the progress of patching and caching or not, it can implement resource-free PAUSE operation if $T_p + T_{pc} \leq B_c$, where T_p and T_{pc} are duration of PAUSE and duration of patching and caching, respectively. Otherwise, the client will issue a merging request to merge back to an existing full stream.

D. Performance Impact

With the previously described modification to support interactive playback operations in Dyadic, CARP, CGRP, and ERMT, we investigate in the following the performance impact of interactive playback operations on the system performance using discrete-event simulations. The simulation results are obtained from a simulator written based on the CNCL simulation library [38] and the system parameters employed are summarized in Table I.

TABLE I
DEFAULT SYSTEM PARAMETERS

Parameters	Symbol	Value
Arrival rate (requests /s)	λ	0.01
Mean seeking distance (s)	μ_{sd}	500
Probability of FSEEK	P_f	N/A
Probability of BSEEK	P_b	N/A
Probability of PAUSE	P_p	N/A
Client buffer size	B_c	N/A
Video length (s)	L	7200
Number of server channels	N	24

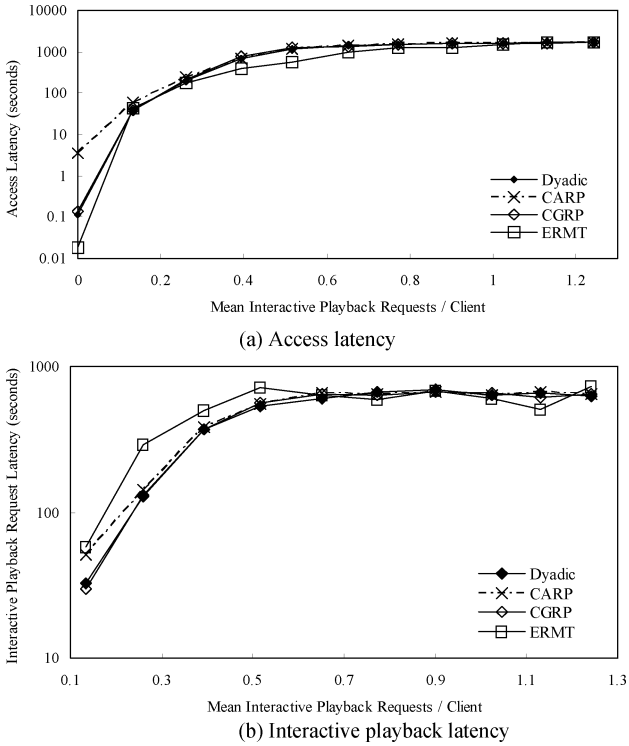


Fig. 6. Performance of different multicast streaming algorithms under various levels of client interactive control intensities.

We use two measures, namely access latency and interactive playback latency to compare the performance of different multicast streaming algorithms under various interactive playback interaction intensities. For interactive playback latency, it is measured from the time the client issues a FSEEK or a BSEEK operation to the time video playback resumes at the new playback point. For PAUSE operation, it is the waiting time measured from the time a client requests to return from the PAUSE state to the NORMAL playback state to the time video playback resumes.

Fig. 6(a) and (b) plots, respectively, the access latency and the interactive playback latency for the interactive Dyadic, CARP, CGRP, and ERMT algorithms. In this experiment, we set N (number of server channels) to 10, P_b (probability of BSEEK) and P_p (probability of PAUSE) to 0, while P_f (probability of FSEEK) varies from 0.0 to 0.1. After a client entered the system, it will issue different interactive playback commands (i.e., transiting between different states in the model depicted in Fig. 5) according to the set of transition probabilities. The horizontal axes in Fig. 6(a) and (b) are measured in *mean interactive playback requests per client* (in units of requests/client). To compute this we count the total number of interactive playback commands issued during the whole simulation, and then divide it by the total number of clients served in the simulation. This measure, referred to hereafter as the *interactive playback intensity*, represents the average frequency at which a client generates interactive playback controls. In the simulations in Fig. 6(a) and (b) each client on average issues 0 to 1.24 interactive playback requests.

At zero interactive playback intensity, all four algorithms perform extremely well with access latency well within 10 s. However, when we increase the interactive playback intensity to just 0.2 requests/client, i.e., on average one interactive playback control is performed for every five clients, the access latency increases dramatically to tens of seconds. For even higher interactive playback intensity, the performance degrades so much so that the performance differences of different algorithms (e.g., CARP versus ERMT) diminish. Similarly, the interactive playback latency also increases dramatically for higher interactive playback intensity, reaching hundreds of seconds even for intensity less than 1 request/client.

These results clearly show that these multicast streaming algorithms are not designed to support interactive playback control. Although we can extend them to support interactive playback control, the system resources will be dominated by the resulting merging requests, leading to unacceptable performance even at very low levels of interactivity (e.g., 0.1 request/client).

It may appear surprising that the performance degradation is so substantial even for interactive playback intensity of only 0.1 request/client. After all, the system can serve interactive playback requests in the same way as admission requests using batching, patching, and caching. However, one fundamental difference is that while admission request always begins video playback from the beginning of the video, interactive playback requests can request playback to resume at any playback points over the entire duration of the video. Consequently, while all the waiting admission requests can be batched and served together once a free channel becomes available, the differing playback points of waiting merging requests render batching improbable. As a result, each merging request will require a dedicated partial stream to resume playback and this significantly increases resource consumption.

To further investigate the performance impact of each type of interactive playback operations, we simulated a system with $N = 24$ channels and interactive playback transition probabilities given by $(P_f, P_b, P_p) = (0.03, 0.03, 0.03)$. Then we vary in turn the transition probability of one of the interactive playback operations from 0.01 to 0.1 and plot the results in Fig. 7(a) and (b). We observe that generally BSEEK results in higher latencies than FSEEK. This is because a BSEEK effectively extends the duration of the video session which in turn will generate more merging requests. By contrast, a FSEEK will skip some of the video and thus effectively shortening the video session, thereby also reducing the number of merging requests generated. The PAUSE operation, on the other hand, incurs significantly less overhead as most of them can be handled in a resource-free manner through client-side buffering as described in Section III-C.

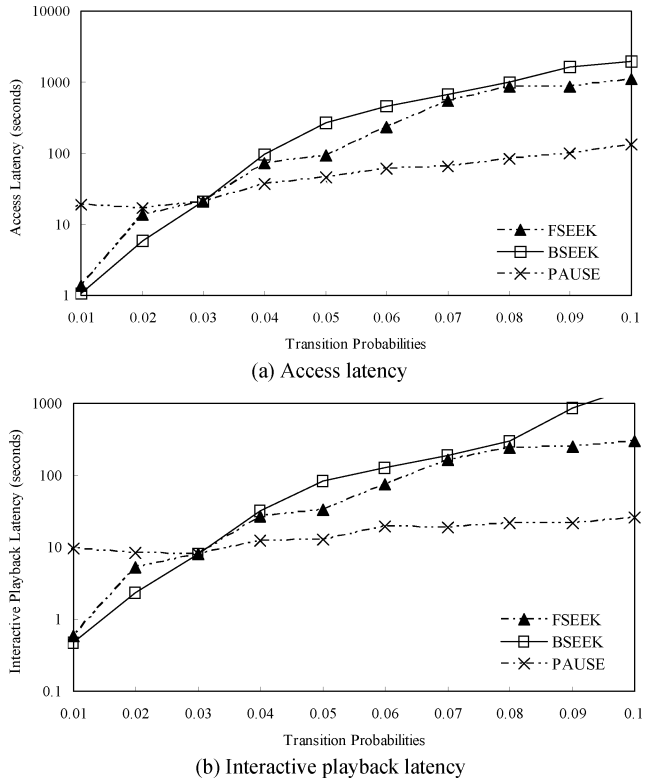


Fig. 7. Effect of interaction parameters on system performance.

Nevertheless, the previous results clearly show that current multicast streaming algorithms will suffer from significant performance degradation if interactive playback control is supported. We address this challenge in the next section by analyzing the cause of the performance degradation and then present a solution.

IV. Static Full Stream Scheduling (SFSS)

To obtain insights into the dynamics of the system when interactive playback control is present, we first formulate a simplified system model that accounts for the cost of serving interactive playback requests. Recall from Section II that there are full streams and partial streams in a multicast streaming system. Assuming full streams are generated once every W seconds, then the mean number of full streams in the system will be equal to L/W. Thus, on average, the playback point after interactive playback control of a client will be offset by $W/2$ seconds when compared to the full stream with the nearest playback point. To merge this request back to the full stream, the system will thus incur a mean merging cost of

$$r_m = \frac{W}{2}R \tag{1}$$

where $(W/2)$ is the mean merging time and R is the video bit rate.

Now as merging requests generally cannot be batched, with a merging request rate of $\mu_{\rm VCR}$ requests/second, the resulting merging operations will consume system resources at a rate of

$$R_m = \mu_{\rm VCR} r_m = \mu_{\rm VCR}\frac{WR}{2}. \tag{2}$$

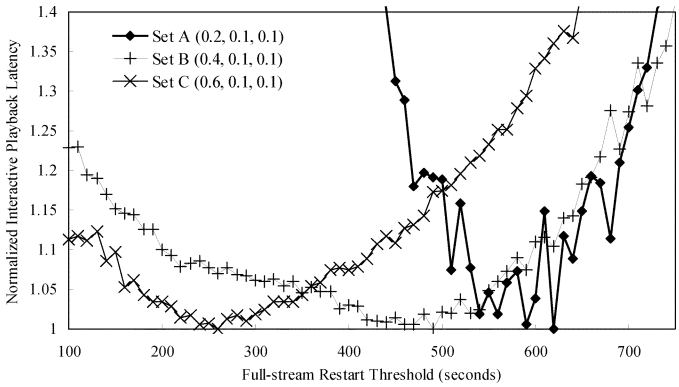


Fig. 8. Illustration of different optimal full-stream restart thresholds for different sets of client interactivity patterns.

Similarly, the resource consumption rate of full streams is equal to

$$R_f = \frac{RL}{W}. \tag{3}$$

Therefore, summing (2) and (3) we obtain the total resource consumption rate

$$R' = R_f + R_m = \frac{RL}{W} + \mu_{\rm VCR}\frac{WR}{2} = \frac{R}{2W}(2L + \mu_{\rm VCR}W^2). \tag{4}$$

Differentiating both sides with respect to W we have

$$\frac{dR'}{dW} = -\frac{RL}{W^2} + \mu_{\rm VCR}\frac{R}{2} \tag{5}$$

with the minimum R' occurring at

$$W = \sqrt{\frac{2L}{\mu_{\rm VCR}}}. \tag{6}$$

Thus W is a decreasing function of $\mu_{\rm VCR}$, suggesting that as the merging request rate increases, we should use a smaller W in order to minimize the resource consumption rate. Clearly, this simple analysis does not take into account factors such as playback request admissions and server bandwidth constraint. Nevertheless it reveals that the full-stream restart threshold employed in existing multicast streaming algorithms may no longer be optimal when interactive playback control is supported.

To verify this hypothesis, we simulated the Dyadic multicast streaming algorithm with three sets of interactive playback parameters (P_f, P_b, P_p) and plot in Fig. 8 the normalized interactive playback latency for full-stream restart threshold W ranging from 100 to 750 s. In the original Dyadic algorithm the restart threshold is derived to be $W = 3600$ s. However the results show that this threshold is no longer optimal when interactive playback is allowed. Moreover, the actual optimal threshold varies with different sets of interactive playback parameters −620 s (Set A), 490 s (Set B), and 260 s (Set C). The results also confirm that as we increase the interactive playback intensity (e.g., from $P_f = 0.2$ in Set A to $P_f = 0.6$ in Set C), the resultant optimal threshold also decreases.

Clearly, one solution to improve the algorithms' performance is to adjust the full-stream restart threshold W for a given set of operating parameter—SFSS. However given the complexity of the multicast streaming algorithms, an analytical model capturing all the essential features of the system does not appear to be tractable. Alternatively we can use simulations to find the optimal full-stream restart threshold. However, even this approach can only be done if all the system parameters, such as arrival rate and the respective transition probabilities for various interactive controls, are all known in advance. This is clearly not possible in practice and thus we investigate in the next section an adaptive approach to solve this problem.

V. Adaptive Full Stream Scheduling

It is clearly not possible to find the optimal full-stream restart threshold W without knowing all the system parameters, which themselves are not known *a priori*. To tackle this problem, we propose a JTS technique to estimate the system parameters while the system is online, and then dynamically adjust the system threshold based on results obtained from an *embedded simulator*. The embedded simulator is built into the video streaming system to perform simulations similar to those in Section IV for finding the optimal full-stream restart threshold W. Once completed, the system uses the newly obtained threshold and the JTS algorithm is restarted and the whole process repeats until the threshold converges. With this JTS technique a service operator then no longer needs to know the system parameters in advance and can let the system dynamically adjusts itself according to the actual measured system parameters. An additional advantage is that if the system parameter changes (e.g., when a new video is introduced into the system or due to time of day, day of week changes), the JTS technique can also adapt to the new system parameters automatically.

The JTS technique comprises three steps. First we need an initial full-stream restart threshold to allow the system to begin operations such that system parameters can be measured. Second, once the system parameters are measured, we need to run the embedded simulator with the measured system parameters to find the optimal threshold. This process is repeated until the obtained threshold converges. Finally, we need to detect any changes in the system parameters which may require adjustment of the threshold again, i.e., repeating step two.

When the system is first started, it clearly does not have any past statistics for the embedded simulator to optimize W. Nevertheless, we can choose an initial threshold based on offline simulation results. The JTS algorithm can then collect statistics from subsequent requests to refine the threshold to improve performance.

Specifically, the system needs to estimate five system parameters: the client arrival rate λ, probability of FSEEK P_f, probability of BSEEK P_b, probability of PAUSE P_p and mean seek distance μ_{sd}. Given a set of n sampled data values X_k $(k = 1, 2, \ldots, n)$ with sample mean M, the *confidence interval* (CI) bounding the estimated mean μ is calculated as

$$M - tS_M \leq \mu \leq M + tS_M \tag{7}$$

where

$$S_M = \frac{S}{\sqrt{n}} \tag{8}$$

$$S = \sqrt{\frac{\sum_{i=1}^{n} (X_i - M)^2}{n-1}} \tag{9}$$

is the standard deviation of the samples, and the parameter t can be found from a t-value lookup table.

Using this CI method we can obtain estimates of the system parameters for use in the embedded simulator to obtain an updated threshold W. This updated threshold is then adopted in the system for new requests while the system continues to collect access statistics for the next round of embedded simulations. The JTS scheme can be executed repetitively or we can also define a stopping condition to free up the server processor to carry out other tasks. For example, we can define a stopping condition such as

$$\Pi = \frac{95\%\mathrm{CI}}{2\mu} \tag{10}$$

where the numerator represents the 95% confidence interval. The server then continues updating the threshold W until the index Π of all five parameters fall below $K\%$, where K is a configurable parameter. After which the server completes the initiation and optimization steps.

If the system parameters do not change, then the previous initialization and optimization steps will be sufficient to optimize the full-stream restart threshold. In practice however, these parameters can change dynamically with changes in the video collection, the time of the day, or the day of the week. For example, the client arrival rate in a day can vary substantially from a low rate at the morning to a high rate at peak hours from about 7:00 pm to 1:00 am.

To address this problem, the system continues to collect access statistics even after system initialization is completed. In particular, the system uses a sliding window to collect the access statistics and periodically compute the statistical index in (10). Once the index Π of any system parameter exceeds a preset threshold, say $(K + Y)\%$, the JTS scheme will be restarted to optimize the threshold again as in the system initialization phase. This is further illustrated in the next section on performance evaluation.

VI. Performance Evaluation

In this section, we evaluate the performance gains achievable by the proposed SFSS scheme when applied to existing multicast streaming algorithms and also study the dynamic behavior of the JTS scheme when the system parameters change dynamically. The simulation results are generated from a discrete-event simulator written based on the CNCL simulation library [38]. For the multicast streaming algorithms, unless stated otherwise we adopt the configurations proposed in their original studies and use the default parameters in Table I.

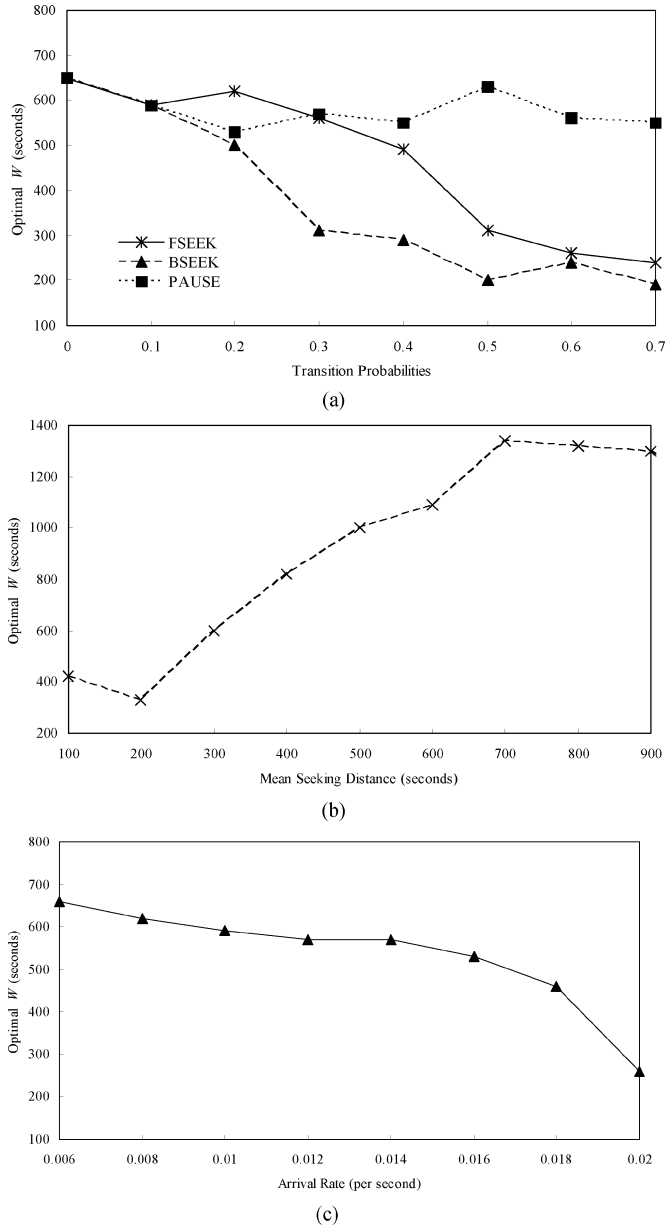


Fig. 9. Optimal full stream restart threshold versus (a) transition probabilities, (b) mean seeking distance, and (c) arrival rate.

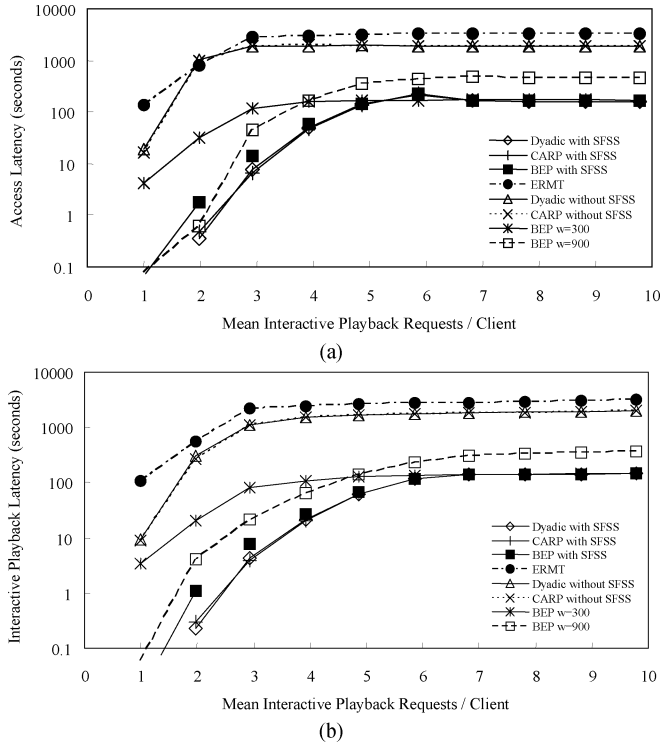


Fig. 10. Effect of interactive playback control intensity on (a) access latency and (b) interactive playback latency.

A. Optimization of the Full Stream Restart Threshold

We first study the sensitivity of the optimal full-stream restart threshold with respect to the system parameters, namely the arrival rate λ, the probability of FSEEK P_f, the probability of BSEEK P_b, the probability of PAUSE P_p, and the mean seeking distance $\mu_{\rm sd}$. For all the simulations, we set $N = 24$, default interaction parameters $(P_f, P_b, P_p) = (0.1, 0.1, 0.1)$ and make use of the interactive Dyadic algorithm for illustration.

Fig. 9(a) plots the optimal threshold versus the transition probabilities for the three types of interactions. We observe that in general the optimal threshold decreases with increases in the interaction probability, e.g., the threshold decreases from 650 to 240 s (P_f) and 190 s (P_b), respectively, when the transition probability is increased from 0.0 to 0.7. For PAUSE the optimal threshold fluctuates between 650 and 550 s, indicating that PAUSE operations incur insignificant performance impacts.

Fig. 9(b) plots the optimal threshold versus mean seek distance ranging from 100 to 900 s. We observe that the optimal threshold can vary significantly, in this case from 330 to 1340 s. Similarly, as shown in Fig. 9(c), the optimal threshold can also vary widely when the arrival rate varies from 0.006 o 0.02 client/s. These two sets of results show that it is undesirable to employ a constant full-stream restart threshold, thus confirming the need for the adaptive JTS scheme presented in Section V.

B. Latencies Comparisons

In this section, we compare the performance gains achievable by the proposed SFSS algorithm when applied to three existing multicast streaming algorithms, namely Dyadic [11], [13], CARP [9], and BEP [33].

In the first set of results, we compare the admission latency [Fig. 10(a)] and interactive playback latency [Fig. 10(b)] of two categories of algorithms. The first category comprises the interactive Dyadic, interactive CARP, interactive ERMT, and BEP algorithms using full-stream restart thresholds as proposed in their original studies [9]–[11], [13], [33]; and the second category comprises the interactive Dyadic, interactive CARP, and BEP algorithms equipped with the SFSS algorithm to determine the full-stream restart threshold.

The results in both Fig. 10(a) and (b) show that the SFSS algorithm can significantly reduce the admission and interactive playback latencies. We set $(P_f, P_b, P_p) = (0.03n, 0.03n, 0.01n)$, where $n = 1$ to 10, resulting in interactive playback intensities from 1.02 to 9.68 requests/client. Inspecting the figures, for examples, SFSS reduces the interactive playback latency of the Dyadic algorithm by 99.92%, 98.58%, and 93.41%, respectively, at interactive playback

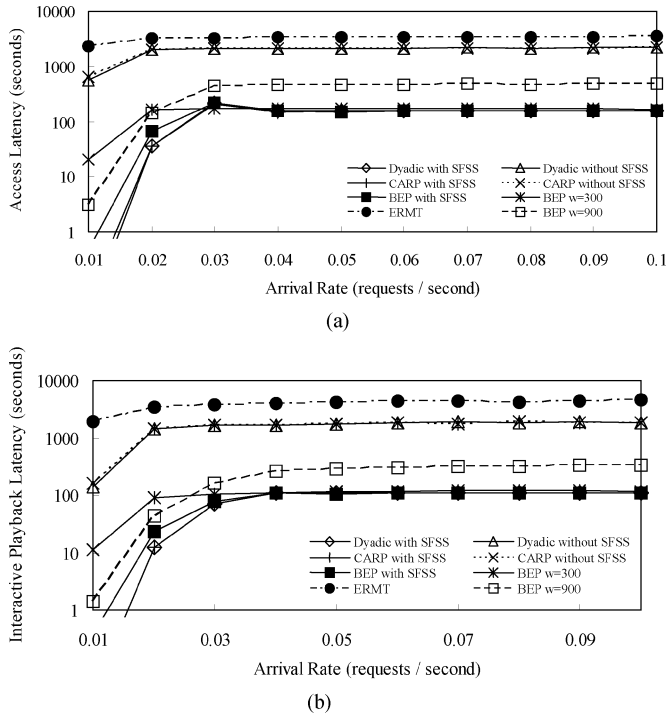


Fig. 11. Comparison of (a) access latency and (b) interactive playback latency, versus arrival rate.

intensities of 1.97, 3.92, and 5.86 requests/client. It is worth noting that the performance gains achieved by the SFSS algorithm far exceed the performance differences between different multicast algorithms, especially at high interactive playback intensities.

In the second set of results, we set $(P_f, P_b, P_p) = (0.05, 0.05, 0.05)$ and plot in Fig. 11(a) and (b) the latencies for different algorithms under different client arrival rates. The observations are consistent with those in Fig. 10(a) and (b), showing that the proposed SFSS algorithm again significantly reduces the admission latency, e.g., by 89.96%, 92.71%, and 92.88%, respectively, at arrival rates of 0.03/s, 0.06/s, and 0.09/s for the Dyadic algorithm.

C. Effect of Client Buffer Constraint

In the previous results, we have set the client buffer size to half of the video length (i.e., $B_c = L/2$) as suggested in the original studies. If we reduce the buffer size, then some admission or merging requests will force the system to generate a full stream instead of merging with a partial stream. Fig. 12 plots the interactive playback latency versus client buffer size from 100 to 7200 s, with $(P_f, P_b, P_p) = (0.1, 0.1, 0.1)$, $N = 24$, and $\lambda = 0.01/\text{s}$. The results show that the latency decreases rapidly when we increase the buffer size from 100 to around 1000 s. Beyond that the improvements are less significant. Thus, the buffer sizes suggested in the original studies (e.g., $L/2 = 3600$ s) are more than sufficient for achieving good performance in the SFSS algorithm.

D. Just-in-Time Simulation (JTS)

In this section we study the performance of the JTS scheme, in particular the time for the JTS scheme to reach the optimal

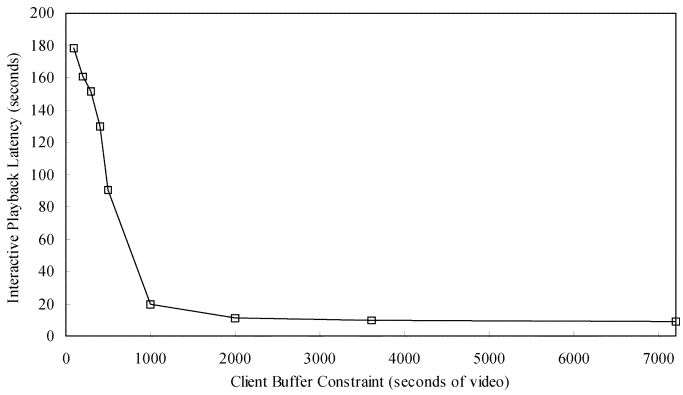


Fig. 12. Performance impact of client buffer constraint.

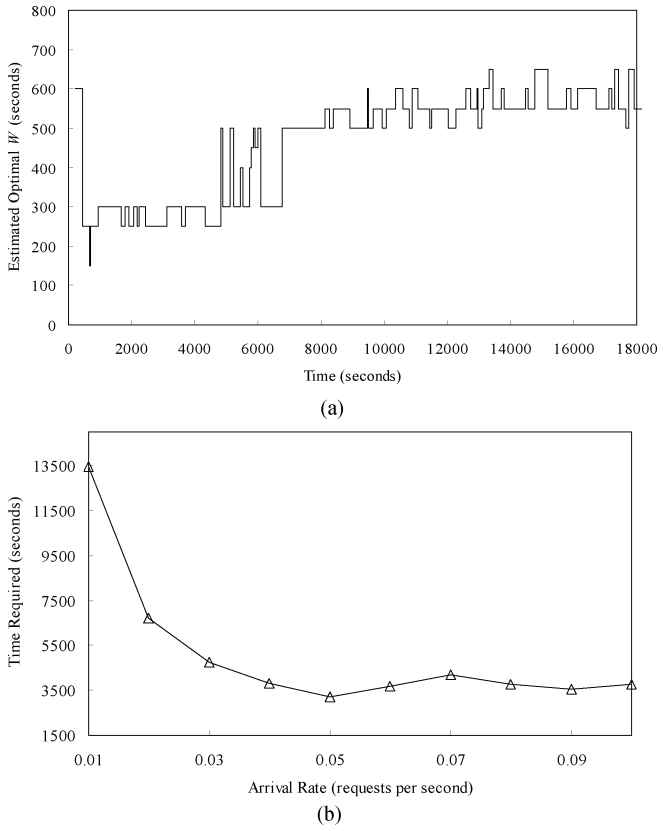


Fig. 13. Parameter estimation at system initialization. (a) Estimated optimal W. (b) Time needed for accurate estimation.

threshold, and JTS's ability to adapt to changes in system parameters.

First we consider JTS's convergence time in Fig. 13(a). At time zero the system is started with no knowledge of the system parameters. The server collects access statistics and then invokes the JTS module to continuously refine the full- stream restart threshold. With an initial threshold value of 600 s, the JTS scheme initially adjusted the threshold to a range of values around 300 s. However, as more access statistics are collected, the system parameters are better estimated and thus JTS progressively refined the threshold to approach the optimal value of 590 s. In this simulation run JTS took about 8500 s to converge to the optimal full-stream restart threshold.

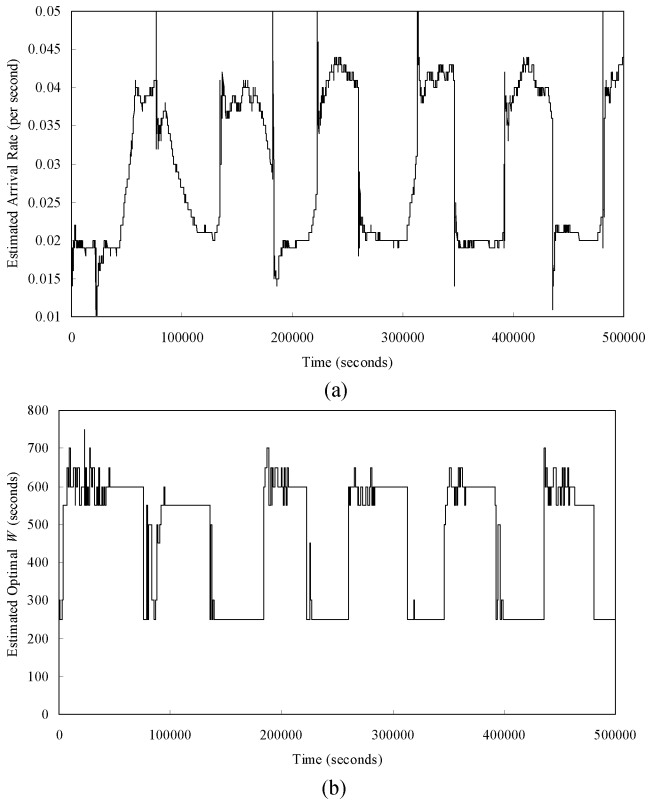


Fig. 14. Automatic adaptation to changing system parameters using JTS. (a) Estimated arrival rate. (b) Estimated optimal W.

Not surprisingly, the convergence time depends on the system parameters, and in particular the arrival rate as it determines the rate at which access statistics are collected. Fig. 13(b) illustrates this relation by plotting the convergence time against arrival rates ranging from 0.01 to 0.1 clients/second. We can observe that higher arrival rates can substantially reduce the convergent time, but in all cases the convergent time is relatively modest compared to the duration of a video session.

Next, we investigate JTS's ability to adapt to varying system parameters. In this experiment, we set $N = 24$, $(P_f, P_b, P_p) = (0.05, 0.05, 0.05)$, but vary the arrival rate λ between 0.02 and 0.04 clients/s, with a cycle time of one day. To study the adaptive ability of JTS under different variation patterns, we modulate the value of λ between 0.02 and 0.04 clients/s using three modulation patterns, namely square, triangular, and sinusoidal waves. In all cases, JTS can adapt to the variations in client access parameters. As an example, Fig. 14(a) and (b) shows, respectively, the estimated arrival rate and estimated optimal threshold against time for the square wave modulation pattern. We can observe that the system can estimate the arrival rates and obtain the corresponding optimal thresholds, which equal to 610 and 260 s for arrival rates of 0.02 and 0.04 clients/s, respectively.

To illustrate the performance gain achievable by JTS, we plot in Fig. 15 the access and interactive playback latencies for a Dyadic multicast system with and without JTS. In this simulation, we set $(P_f, P_b, P_p) = (0.05, 0.05, 0.05)$ and arrival rate λ varying between 0.02 and 0.04 clients/s, with a cycle time of one week. The results show that without JTS, both access and interactive playback latencies are substantially higher regardless

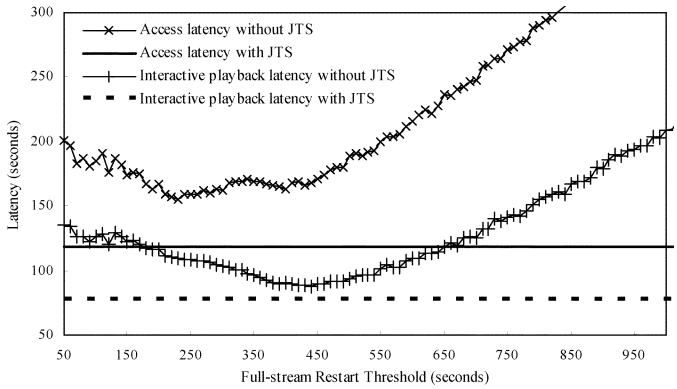


Fig. 15. Performance of JTS compared to a Dyadic multicast system enhanced with SFSS.

of the value chosen for the full-stream restart threshold. This is because the optimal restart threshold varies with time and so a fixed restart threshold cannot adapt to the changing arrival rates. By contrast, the proposed JTS scheme can dynamically optimize the restart threshold to achieve over 23% and 11% reductions in access and interactive playback latencies, respectively.

VII. Conclusions

In this study, we show that current state-of-the-art multicast streaming algorithms, while extremely efficient, all suffer from significant performance degradations when interactive playback controls are supported. To tackle this problem, we present a SFSS algorithm that schedules admission requests and merging requests using two separate queues, and optimizes the full stream restart threshold with interactive playback controls accounted for. Simulation results show that this SFSS algorithm can significantly reduce the performance degradation. Nevertheless, the SFSS algorithm requires *a priori* knowledge of system parameters that are clearly not available in practice. Thus, we present a novel JTS technique to embed a system simulator within the video streaming system to dynamically measure and estimate the required system parameters for obtaining the optimal full stream restart threshold while the system is online serving users. Results show that this JTS technique can accurately estimate the system parameters as well as adapt to changes in the system parameters to reoptimize the system automatically. This general technique can be applied to many of the current state-of-the-art multicast streaming algorithms, including but not limited to Dyadic, CARG, CGRP, and BEP to significantly improve their performance in providing true interactive VoD services for a large user population.

Acknowledgment

The authors wish to thank the anonymous reviewers for their constructive comments and suggestions in improving this paper.

References

[1] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. 2nd ACM Int. Conf. Multimedia*, 1994, pp. 15–23.

[2] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, Channel allocation under batching and interactive playback control in movie-on-demand servers IBM, Yorktown Heights, NY, 1994, Res. Rep. RC19588.

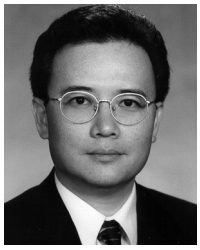
[3] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *ACM Multimedia Syst.*, no. 4, pp. 112–121, 1996.
[4] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proc. Int. Conf. Multimedia Syst.*, Jun. 1996.
[5] H. Shachnai and P. S. Yu, "Exploring wait tolerance in effective batching for video-on-demand scheduling," in *Proc. 8th Israeli Conf. Comput. Syst. Softw. Eng.*, Jun. 1997, pp. 67–76.
[6] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. 6th Int. Conf. Multimedia*, Sep. 1998, pp. 191–200.
[7] Y. Cai, K. A. Hua, and K. Vu, "Optimizing patching performance," in *Proc. SPIE/ACM Conf. Multimedia Comput. Netw.*, San Jose, CA, Jan. 1999, pp. 204–215.
[8] Y. Cai and K. A. Hua, "An efficient bandwidth-sharing technique for true video on demand systems," in *Proc. 7th ACM Int. Multimedia Conf.*, Orlando, FL, Nov. 1999, pp. 211–214.
[9] Z. Shi and C. C. J. Kuo, "Recursive patching for video-on-demand (VOD) systems with limited client buffer constraint," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2002, vol. 1, pp. 373–376.
[10] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for video-on-demand servers," in *Proc. 7th ACM Int. Multimedia Conf.*, pp. 199–203.
[11] E. G. Coffman, P. Jelenkovic, and P. Momcilovic, "Provably efficient stream merging," presented at the 6th Int. Workshop Web Caching Content Distribution, Boston, MA, 2001.
[12] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam, "Comparison of stream merging algorithms for media-on-demand," in *Proc. SPIE Multimedia Comput. Netw.*, San Jose, CA, Jan. 2002, pp. 115–129.
[13] E. G. Coffman, P. Jelenkovic, and P. Momcilovic, "The Dyadic stream merging algorithm," *J. Algorithms*, vol. 43, no. 1, pp. 120–137, Apr. 2002.
[14] L. Golubchik, J. C. S. Lui, and R. Muntz, "Reducing I/O demand in video-on-demand storage servers," in *Proc. 1995 ACM SIGMETRICS Joint Int. Conf. Meas. Modeling Comput. Syst.*, Ottawa, ON, Canada, May 1995, pp. 25–36.
[15] ——, "Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers," *ACM Multimedia Syst.*, vol. 4, no. 3, pp. 140–155, 1996.
[16] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal piggyback merging policies for video-on-demand systems," in *Proc. Int. Conf. Multimedia Syst.*, Jun. 1996, pp. 253–258.
[17] S. W. Lau, J. C. S. Lui, and L. Golubchik, "Merging video streams in a multimedia storage server: Complexity and heuristics," *ACM Multimedia Syst.*, vol. 6, no. 1, pp. 29–42, 1998.
[18] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective video-on-demand," in *Proc. IS&T/SPIE Conf. Multimedia Comput. Netw. 2000 (MMCN 2000)*, San Jose, CA, Jan. 2000, pp. 206–215.
[19] S. Viswanathan and T. Imielinski, "Pyramid broadcasting for video on demand service," in *Proc. SPIE Multimedia Comput. Netw. Conf.*, San Jose, CA, 1995, pp. 66–77.
[20] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-deman systems," in *Proc. ACM SIGCOMM '97*, Sep. 1997, pp. 89–100.
[21] J. F. Paris, S. W. Carter, and D. D. E. Long, "A low bandwidth broadcasting protocol for video on demand," in *Proc. 7th Int. Conf. Comput. Commun. Netw. (IC3N'98)*, Oct. 1998, pp. 690–697.
[22] W. C. Liu and J. Y. B. Lee, "Constrained consonant broadcasting – A generalized periodic broadcasting scheme for large scale video streaming," in *Proc. IEEE Int. Conf. Multimedia Expo*, Baltimore, MD, Jul. 2003, vol. 1, pp. 805–808.
[23] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE Int. Conf. Multimedia Comput. Syst.*, Florence, Italy, Jun. 1999, vol. 2, pp. 117–121.
[24] L. Gao, Z. L. Zhang, and D. Towsley, "Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams," in *Proc. 7th ACM Int. Multimedia Conf.*, Orlando, FL, Nov. 1999, pp. 203–206.
[25] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (Mcache): An adaptive zero-delay video-on-demand service," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 440–56, Mar. 2001.
[26] J. Y. B. Lee and C. H. Lee, "Design, performance analysis, and implementation of a super-scalar video-on-demand system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 11, pp. 983–997, Nov. 2002.
[27] C. W. Kong, J. Y. B. Lee, M. Hamdi, and V. O. K. Li, "Turbo-slice-and-patch: An algorithm for metropolitan scale VBR video streaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, pp. 338–353, Mar. 2006.
[28] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, no. 4, pp. 51–62, Apr. 1997.
[29] V. O. K. Li and W. Liao, "Interactive Video-on-Demand System," U.S. Patent 6 543 053 B1, Apr. 1, 2003.
[30] K. C. Almeroth and M. H. Ammar, "On the performance of a multicast delivery video-on-demand service with discontinuous VCR actions," in *Proc. IEEE Int. Conf. Commun.*, 1995, vol. 3, pp. 1631–1635.
[31] E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," in *Proc. IEEE Int. Conf. Multimedia Comput. Syst.*, 1998, pp. 66–75.
[32] W. F. Poon, K. T. Lo, and J. Feng, "Design and analysis of multicast delivery to provide VCR functionality in video-on-demand systems," in *Proc. 2nd Int. Conf. ATM*, 1999, pp. 132–139.
[33] H. Ma and K. G. Shin, "A new scheduling scheme for multicast true VoD service," in *Proc. PCM2001, Springer LNCS 2195*, 2001, pp. 708–715.
[34] J. B. Kwon and H. Y. Yeom, "Providing VCR functionality in staggered video broadcasting," *IEEE Trans. Consum. Electron.*, vol. 48, no. 1, pp. 41–48, Feb. 2002.
[35] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley, "Providing VCR capabilities in large-scale video servers," in *Proc. 2nd ACM Int. Conf. Multimedia*, Oct. 1994, pp. 25–32.
[36] V. O. K. Li, W. J. Liao, X. X. Qiu, and E. W. M. Wong, "Performance model of interactive video-on-demand systems," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 8, pp. 1099–1109, Aug. 1996.
[37] P. Branch, G. Egan, and B. Tonkin, "Modeling interactive behaviour of a video based multimedia system," in *Proc. IEEE Int. Conf. Commun.*, 1999, vol. 2, pp. 978–982.
[38] M. Junius, M. Steppler, M. Büter, and D. Pesch, *CNCL – Communication Networks Class Library*. Aachen, Germany: Aachen Univ. Technol., 1999.

Ying Wai Wong received the B.Eng. and M.Phil. degrees in information engineering from the Chinese University of Hong Kong, Hong Kong, in 2001 and 2003, respectively. In 2004, he joined the Department of Linguistics and Modern Languages at the Chinese University of Hong Kong to pursue graduate degrees. He studied and reported asymmetries in tone production and perception processes by means of acoustic analyses and psychoacoustic experiments. He is currently pursuing the Ph.D. degree, with research interests in production and perception phenomena related to Cantonese tones in continuous speech.

During 2001–2003, he was at the Multimedia Communications Laboratory investigating techniques to support interactive playback control in multicast video streaming systems.

Jack Y. B. Lee (M'95–SM'03) received the B.Eng. and Ph.D. degrees in information engineering from the Chinese University of Hong Kong, Hong Kong, in 1993 and 1997, respectively.

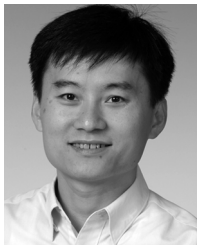
He participated in the research and development of video streaming systems from 1997 to 1998 where he and his team developed novel parallel video server architectures for building cost-effective, scalable and fault-tolerant video-on-demand systems. This work had resulted in numerous publications, two U.S. Patents, and the technologies are subsequently transferred to a spin-off technology company for commercialization. He was a faculty member at the Department of Computer Science, Hong Kong University of Science and Technology, from 1998 to 1999, and in 1999 he joined the Department of Information Engineering, Chinese University of Hong Kong, to establish the Multimedia Communications Laboratory to spearhead research in distributed multimedia systems, peer-to-peer networks, multicast communications, Internet protocols and applications.



Victor O.K. Li (S'80–M'81–SM'86–F'92) received the B.S., M.S.E.E., and D.Sc. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1977, 1979, 1980, and 1981, respectively.

He joined the University of Southern California (USC), Los Angeles, in February 1981, and became Professor of electrical engineering and Director of the USC Communication Sciences Institute. Since September 1997, he has been with the University of Hong Kong, Hong Kong, where he is Chair Professor of Information Engineering at the Department of Electrical and Electronic Engineering. He has also served as Managing Director of Versitech Ltd., the technology transfer and commercial arm of the University, and on various corporate boards. His research is in information technology, including all-optical networks, wireless networks, and Internet technologies and applications. He is a Principal Investigator of the Area of Excellence in Information Technology funded by the Hong Kong Government. Sought by government, industry, and academic organizations, he has lectured and consulted extensively around the world. He is a part-time member of the Central Policy Unit of the Hong Kong Government. He also serves on the Innovation and Technology Fund (Electronics) Vetting Committee, the Small Entrepreneur Research Assistance Programme Committee, the Engineering Panel of the Research Grants Council, and the Task Force for the Hong Kong Academic and Research Network (HARNET) Development Fund of the University Grants Committee. He was a Distinguished Lecturer at the University of California at San Diego, at the National Science Council of Taiwan, and at the California Polytechnic Institute. He has also delivered keynote speeches at many international conferences.

Prof. Li chaired the Computer Communications Technical Committee of the IEEE Communications Society 1987–1989, and the Los Angeles Chapter of the IEEE Information Theory Group 1983–1985. He co-founded the International Conference of Computer Communications and Networks (IC3N), and chaired its Steering Committee 1992–1997. He also chaired various international workshops and conferences, including most recently, IEEE INFOCOM 2004 and IEEE HPSR 2005. He has served as an Editor of *IEEE Network*, IEEE JOURNAL OF SELECTED AREAS OF COMMUNICATION (JSAC) Wireless Communications Series, and Telecommunication Systems. He also guest edited special issues of IEEE JSAC, Computer Networks and ISDN Systems, and KICS/IEEE *Journal of Communications and Networking*. He is now serving as an Editor of ACM/Springer Wireless Networks and IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He has been appointed to the Hong Kong Information Infrastructure Advisory Committee by the Chief Executive of the Hong Kong Special Administrative Region (HKSAR). He has received numerous awards, including, most recently, the Changjiang Chair Professorship at Tsinghua University, Beijing, China, from the Ministry of Education, China, U.K. Royal Academy of Engineering Senior Visiting Fellowship in Communications, the Outstanding Researcher Award of the University of Hong Kong, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of HKSAR, China. He is a Fellow of the HKIE and the IAE.



Gary S.-H. Chan (S'89–M'98–SM'03) received the B.S.E. degree (highest hons.) in electrical engineering from Princeton University, Princeton, NJ, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems, and the M.S.E. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1999, respectively, with a minor in business administration.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, and an Adjunct Researcher with Microsoft Research Asia, Beijing, China. He was a Visiting Assistant Professor in networking at the Department of Computer Science, University of California at Davis, from 1998 to 1999. During 1992–1993, he was a Research Intern at the NEC Research Institute, Princeton, NJ. His research interest includes multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks.

Dr. Chan served as a Vice-Chair of IEEE Communications Society (COMSOC) Multimedia Communications Technical Committee (MMTC) from 2003 to 2006. He is a Guest Editor for special issues on "Peer-to-Peer Multimedia Streaming" in IEEE *Communication Magazine* (2007) and "Advances in Consumer Communications and Networking" in Springer *Multimedia Tools and Applications* (2007). He has been a co-chair of multimedia symposium in IEEE GLOBECOM (2006) and IEEE ICC (2005 and 2007), and for the workshop on "Advances in Peer-to-Peer Multimedia Streaming" in ACM Multimedia Conference (2005). He is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa. He was a William and Leila Fellow at Stanford University during 1993–1994. At Princeton, he was the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence in 1993.