Design, Performance Analysis, and Implementation of a Super-Scalar Video-on-Demand System

Jack Y. B. Lee, *Member, IEEE,* and C. H. Lee

***Abstract*—Despite the availability of video-on-demand (VoD) services in a number of cities around the world, large-scale deployment of VoD services in a metropolitan area is still economically impractical. This study presents a novel super-scalar architecture for building very large-scale and efficient VoD systems. The proposed architecture combines the use of static multicast, dynamic multicast, and intelligent client-side caching to vastly reduce server and network resource requirement. Moreover, in sharp contrast to conventional VoD systems where the system cost increases at least linearly with the system scale, the proposed architecture becomes more efficient as the system scales up and can ultimately be scaled up to serve any number of users while still keeping the startup latency short. This paper presents this new architecture, proposes methods to support interactive playback controls without the need for additional server or client resources, and derives an approximate performance model to relate the startup latency with other system parameters. The performance model is validated using simulation and the architecture is evaluated under various system settings. Lastly, a system implementation is presented and benchmarking results obtained to further verify the architecture, the performance model, and the simulation results.**

***Index Terms*—Multicast, performance analysis, super-scaler video-in-demand (SS-VoD), super-scalar, video-on-demand.**

I. INTRODUCTION

VIDEO-ON-DEMAND (VoD) systems have been commercially available for many years. However, except for a few cities, large-scale deployment of VoD service is still uncommon. One of the reasons is the high cost in provisioning large-scale interactive VoD service. The traditional true-VoD (TVoD) model calls for a dedicated channel, both at the server and at the network for each active user during the entire duration of the session (e.g., 1–2 h for movies). In a city with potentially millions to tens of millions of subscribers, the required infrastructure investment is immense.

To tackle this challenge, a number of researchers have investigated various innovative architectures to improve the scalability and efficiency of large-scale VoD systems [1]–[37]. Notable examples include batching [1]–[5], chaining [6], [7], periodic broadcasting [8]–[21], patching [22]–[26], and piggybacking [27]–[30]. Moreover, these techniques are often complementary and hence can be combined into even more sophisticated architectures to further improve efficiency [31]–[37]. A more detailed review of these works will be presented in Section II.

The central theme in these pioneering studies is the use of network-level multicast to enable sharing of transmitted data among a large number of users, thereby drastically reducing resource requirements when scaling up the system. The challenges to applying multicast to VoD applications are threefold. First, one needs to design a multicast transmission schedule to maximize resource sharing while at the same time minimize startup latency. Second, as users arrive at random time instants, one would need ways to group them together so that they can share just a few multicast transmissions. Third, to provide service comparable to traditional TVoD service, one would also need to find ways to support interactive controls such as pause-resume, slow motion, seeking, etc., during video playback.

This study investigates a super-scalar architecture—Super-Scalar VoD (SS-VoD) that addresses the previous three challenges. Specifically, SS-VoD integrates ideas from batching, static multicasting, dynamic multicasting, and client-side caching to form a simple yet efficient architecture. Static multicast channels in SS-VoD are scheduled using a simple staggered schedule similar to a near VoD (NVoD) system. While more sophisticated multicast schedules [10], [11], [13]–[15], [17]–[21] can achieve better resource savings, they often require more client-side bandwidth and client-side buffer. More importantly, these multicast schedules require the client to switch between multiple multicast channels during a video session to achieve the resource savings. For large-scale systems comprising millions of users, the channel switching overhead can present a significant burden to the network.

Let us consider IP multicast as an example. A client wishing to switch from one multicast channel to another will need to send an Internet Group Management Protocol (IGMP) [38] message to the edge router to stop it from forwarding data in the current multicast group. Another IGMP message will then be sent to request the edge router to start forwarding data from the new multicast group. Unlike processing data packets, these control messages and group management processing are performed in software running on the router CPU. Hence the more channel switching it requires, the more chance that a router could become overloaded. This could lead to missed schedule and/or data loss, resulting in client playback hiccup and/or visual quality degradation.

Another advantage of using the simple staggered multicast schedule in SS-VoD is for the support of interactive playback control. In particular, we show in Section IV that interactive controls such as pause–resume, slow motion, and seeking can be supported in SS-VoD without incurring any additional resource

Manuscript received August 19, 2001; revised August 15, 2002. This work was supported in part by the Hong Kong Special Administrative Region (HKSAR) Research Grant Council under Grant CUHK6095/99E and Grant CUHK4328/02E, and by the Area-of-Excellence in Information Technology. This paper was recommended by Associate Editor H.-J. Zhang.

Tha authors are with the Department of Information Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: jacklee@computer.org; chlee9@ie.cuhk.edu.hk).

Digital Object Identifier 10.1109/TCSVT.2002.805510

or processing at the video server nor any additional buffer at the client.

The dynamic multicast channels in SS-VoD, on the other hand, are scheduled online by an admission controller. These dynamic multicast channels are used to allow a user to start and sustain playback while caching data from another static multicast channel. Users sharing a dynamic multicast channel will eventually catch up with their cached data and the dynamic channel can then be released. Observing that in practice most users are willing to wait for a small startup delay of, say, a few seconds, we design an admission controller for SS-VoD to enable more users requesting the same movie to share the same dynamic multicast channel. This dramatically reduces the system resource required to achieve the same startup latency.

Another contribution of this study is in performance modeling. Systems of this complexity are inherently difficult to analyze and many studies have therefore resorted to using simulations to obtain performance results. A number of previous works [32], [34], [36] have successfully analyzed system performance by deriving the average number of channels needed given other system parameters, assuming that server and network bandwidth can be allocated whenever needed. These performance models have provided important insights into the relations between different system parameters. This study takes a different approach and begins with a fixed total number of channels and then proceeds to derive the startup latency given other system parameters. We believe that this model can better address the need of VoD service providers as server and network bandwidth are often fixed during system installation.

The rest of the paper is organized as follows. Section II reviews some previous works and discusses the differences of our approach. Section III presents the SS-VoD architecture, with details of the transmission schedule, admission control algorithm, and channel merging algorithm. Section IV presents algorithms to support three interactive controls, namely pause–resume, slow motion, and seeking under the SS-VoD architecture. Section V presents a performance model of SS-VoD where the startup latency is related to other system parameters. Section VI evaluates performance of the SS-VoD architecture using results obtained from analysis and simulation. Section VII presents the implementation of a system prototype and the experimental results obtained from benchmarking. Finally, we conclude the study in Section VIII.

II. Related Work

In this section, we briefly review the related works and compare them with this study. As mentioned in Section I, there are five common approaches for improving VoD system efficiency, namely batching, chaining, periodic broadcasting, patching, and piggybacking. These approaches can be used individually or combined to form even more sophisticated architectures.

The first approach, *batching*, groups users waiting for the same video data and then serves them using a single multicast channel [1]–[5]. This batching process can occur passively while the users are waiting or actively by delaying the service of earlier users to wait for later users to join the batch. Various batching policies have been proposed in recent years, including first-come-first-serve (FCFS) and maximum queue length (MQL) proposed by Dan *et al.* [1], maximum factored queue (MFQ) proposed by Aggarwal *et al.* [4], and Max_Batch and Min_Idle proposed by Shachnai *et al.* [5].

The second approach, called *chaining* or *virtual batching*, as proposed by Sheu *et al.* [6], [7], builds upon batching and exploits client-side disk and network bandwidth to reduce the batching delay. Specifically, clients from the same batch form a logical chain where the first client of the batch starts playback immediately, caches the video data, and then forwards them to the next client in the chain. This chaining process repeats for subsequent clients in the batch. The primary advantage of this approach is that earlier clients are not penalized with a longer wait due to the batching process. The tradeoff is that the clients and the access network must have bandwidth to stream video data to other clients.

The third approach, called *periodic broadcasting*, schedules the transmissions of a video over multiple multicast channels in a fixed pattern [8]–[21]. For the simplest example, NVoD repeatedly transmits a video over multiple channels at fixed time intervals so that an arriving user can simply join the next upcoming multicast cycle without incurring additional server resource. More sophisticated broadcasting schedules such as pyramid broadcasting [10], [11], skyscraper broadcasting [13], and Greedy Disk-Conserving Broadcasting (GDB) [17] have been proposed to further reduce the resource requirement by trading off client-side access bandwidth, buffer requirement, and multicast channel switching overhead as discussed in Section I.

The fourth approach, called *patching*, exploits client-side bandwidth and buffer space to merge users on separate transmission channels onto an existing multicast channel [22]–[26]. The idea is to let the client caches data from a nearby (in playback time) multicast transmission channel while sustaining playback with data from another transmission channel—called a patching channel in [23].[1] This patching channel can be released once video playback reaches the point where the cached data began, and playback continues via the cache and the shared multicast channel for the rest of the session.

The fifth approach, called *piggybacking*, merges users on separate transmission channels by slightly increasing the playback rate of the latecomer (and/or slightly decreasing the playback rate of the early starter) so that it eventually catches up with another user and hence both can then be served using the same multicast channel [27]–[30]. This technique exploits users' tolerance on playback rate variations and does not require additional buffer on the client side as in the case of patching.

The previous five approaches are complementary and hence can be combined to form even more sophisticated architectures. For example, Liao *et al.* [22], Hua *et al.* [23], and Cao *et al.* [24] have investigated integrating batching with patching to avoid the long startup delay due to batching. Oh *et al.* [31] have proposed an adaptive hybrid technique which integrated batching with Skyscraper Broadcasting. They proposed a new batching policy called Largest Aggregated Waiting Time First (LAW), which

[1] In [23], the term *patching* referred to the whole architecture that combined patching and batching. In this paper we treat these two techniques separately.

is a refinement of the MFQ policy [4]. Unlike most studies which assume stationary video popularity, their approach estimates video popularity using an online algorithm and revises the Skyscraper Broadcasting schedule from time to time to adapt to video popularity changes.

More recently, Gao *et al.* [32] proposed a *controlled multicast* technique that integrated patching with dynamically scheduled multicasting. This is further refined by Gao *et al.* [34] in their *catching* scheme, which employed the GDB schedule for the periodic broadcasting channels. Their study found out that catching outperforms controlled multicast at high loads but is otherwise not as good as controlled multicast. This motivated them to further combine catching with controlled multicast to form a *selective catching* scheme that dynamically switch between catching and controlled multicast depending on the system load.

In another study, Ramesh *et al.* [36] proposed and analyzed the multicast with cache (*Mcache*) approach that integrated batching, patching, and prefix caching. They proposed placing regional cache servers close to the users to serve the initial portion (prefix) of the videos. In this way, a client can start video playback immediately by receiving prefix data streamed from a regional cache server. The server will then dynamically schedule a patching channel for the client to continue the patching process beyond the prefix and also identify an existing multicast channel for the client to cache and eventually merge into. This architecture has been shown to outperform prefix-assisted versions of dynamic skyscraper, GDB, and selective catching.

The SS-VoD architecture proposed in this study differs from the previous studies in four major ways. First, we combine both static and dynamic multicast with ideas from patching and batching. In particular, we employ the simple staggered periodic multicast schedule for the static multicast channels. Compared to dynamically scheduled multicast schedules [12], [19], [32], [36], this simple multicast schedule enables us to implement interactive playback control such as pause–resume, slow motion, and seeking in a simple yet efficient way. Compared to more sophisticated periodic broadcasting schedules [10], [11], [13]–[15], [17]–[21], the staggered schedule enjoys lower client buffer requirement and, more importantly, eliminates the need to switch multicast channels during a video session as discussed in Section I.

Second, we take advantage of users' tolerance to a small startup latency to enable multiple users to share a dynamic multicast channel for patching [35]. This technique achieves the benefits of batching patching requests similar to Mcache [36] but without the need for regional cache servers. As will be shown in later sections, this technique greatly reduces the startup latency at high arrival rates.

Third, instead of using simulation or deriving the average channel requirement assuming server and network bandwidth can be allocated whenever needed, we modeled the system performance by beginning with a given total number of multicast channels and then derive the startup latency accordingly. We believe this model will be useful for the VoD service providers to dimension the system requirement and to estimate system performance under different user arrival rates.

Last but not least, we successfully implemented the proposed architecture, complete with pause–resume, slow motion, and seeking controls. Using this system implementation, we obtained benchmark results that further validated the analytical and simulation results.

In a previous work by the author [33], [37], we proposed a UVoD architecture that also integrated patching with static periodic multicast. SS-VoD improves upon UVoD in two ways. First, SS-VoD employs multicast patching channels instead of unicast patching channels in UVoD, which dramatically improves the system performance at high loads. Second, SS-VoD employs intermediate admission controllers to admit and consolidate client requests for transmission to the servers. This three-tier architecture ensures that the server will not be overloaded with client requests when one scales up the system to millions of users. These two architectures will be compared in more detail in Section VI.

III. Super-Scalar Architecture

In this section we present the design of the SS-VoD architecture. As depicted in Fig. 1, the system comprises a number of service nodes connected via a multicast-ready network to the clients. The clients form clusters according to their geographical proximity. An admission controller in each cluster performs authentication and schedules requests for forwarding to the service nodes.

Each service node operates independently from each other, having its own disk storage, memory, CPU, and network interface. Hence a service node is effectively a mini video server, albeit serving a small number of video titles to the *entire* user population. This modular architecture can simplify the deployment and management of the system. For example, since the configuration of each service node is decoupled from the scale of the system and each service node carries just a few movies, a service provider simply deploys the right number of service nodes according to the desired video selections. Additional service nodes can be added when more movie selections are needed, with the existing nodes remain unchanged.

SS-VoD achieves scalability and bandwidth efficiency with two techniques. The first technique is through the use of multicast to serve multiple clients using a single multicast channel. However, simple multicast such as those used in an NVoD system limits the time for which a client may start a new video session. Depending on the number of multicast channels allocated for a video title, this startup delay can range from a few minutes to tens of minutes. To tackle this initial delay problem, we employ patching to enable a client to start video playback at any time using a dynamic multicast channel until it can be merged back onto an existing multicast channel. Sections III-A through III-C present these techniques in detail.

A. Transmission Scheduling

Each service node in the system streams video data into multiple multicast channels. Let M be the number of video titles served by each service node and let N be the total number of multicast channels available to a service node. For simplicity, we assume N is divisible by M and hence each video title

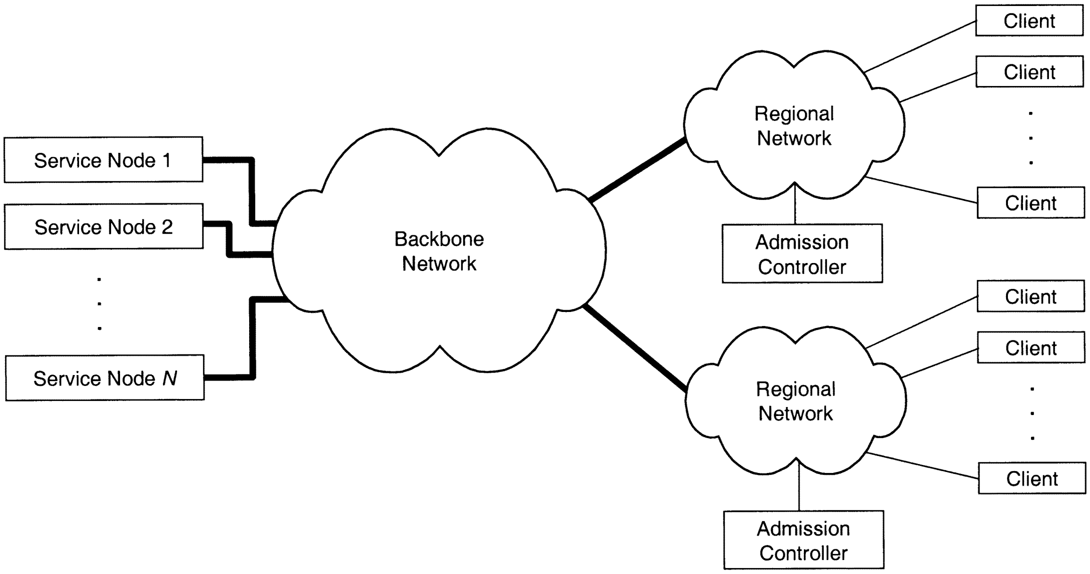

Fig. 1. SS-VoD architecture.

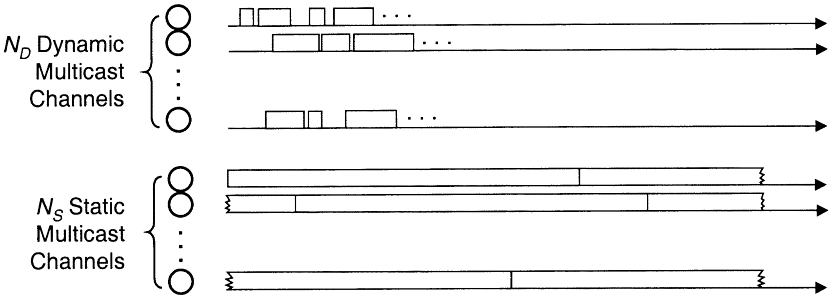
N_D Dynamic Multicast Channels
N_S Static Multicast Channels

Fig. 2. Transmission schedules for static and dynamic multicast channels.

is served by the same number of multicast channels, denoted by $N_M = N/M$. These multicast channels are then divided into two groups of N_S static multicast channels and $N_D = N_M - N_S$ dynamic multicast channels.

The video title is multicast repeatedly over all N_S static multicast channels in a time-staggered manner as shown in Fig. 2. Specifically, adjacent channels are offset by

$$T_R = \frac{L}{N_S} \tag{1}$$

seconds, where L is the length of the video title in seconds. Transmissions are continuously repeated, i.e., restart from the beginning of a video title after transmission completes, regardless of the load of the server or how many users are active. These static multicast channels are used as the main channel for delivering video data to the clients. A client may start out with a dynamic multicast channel but it will shortly be merged back to one of these static multicast channels. Section III-B presents the admission procedure for starting a new video session, and we explain in Section III-C how the client is merged back to one of the static multicast channels.

B. Admission Control

To reduce the response time while still leveraging the bandwidth efficiency of multicast, SS-VoD allocates a portion of the multicast channels and schedules them dynamically according to the request arrival pattern.

Specifically, a new request always goes to the admission controller. Knowing the complete transmission schedule for the static multicast channels, the admission controller then determines if the new user should wait for the next upcoming multicast transmission from a static multicast channel or should start playback with a dynamic multicast channel. In the former case, the client just waits for the next multicast cycle to begin, without incurring any additional load to the backend service nodes. In the latter case, the admission controller performs additional processing to determine if a new request needs to be sent to the appropriate service node to start a new dynamic multicast stream.

Fig. 3 depicts the state-transition diagram for the admission procedure. Beginning from the IDLE state, suppose that a new request arrives at time a_i, which is between the start time of the previous multicast cycle, denoted by t_m, and the start time of the next multicast cycle, denoted by t_{m+1}. Now a predefined

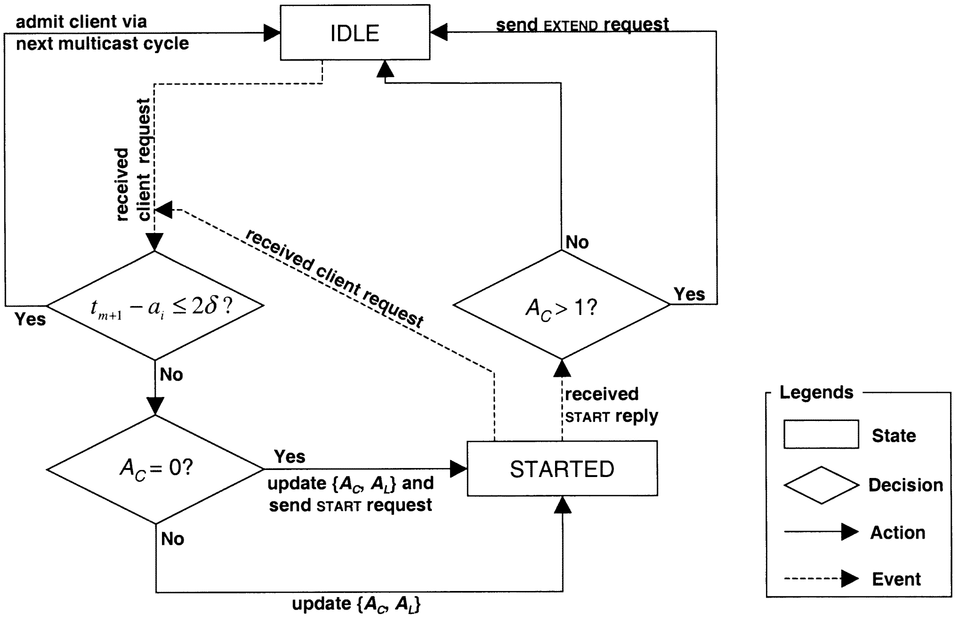

Fig. 3. State-transition diagram for the admission controller.

admission threshold, denoted by δ, determines the first admission decision made by the admission controller: the new request will be assigned to wait for the next multicast cycle to start playback if the waiting time, denoted by w_i, is equal to or smaller than 2δ, i.e.,

$$w_i = t_{m+1} - a_i \le 2\delta. \tag{2}$$

We call these requests *statically admitted* and the admission controller returns to the IDLE state afterwards. This admission threshold is introduced to reduce the amount of load going to the dynamic multicast channels. Optimization procedure for this admission threshold will be presented in Section V-C.

If (2) does not hold, then the admission controller will proceed to determine if a request needs to be sent to the appropriate service node to start a new dynamic multicast stream—*dynamically admitted*. The service nodes and admission controllers each keeps a counter-length tuple: $\{A_C, A_L\}$, where $A_C = \{0, 1\}$ is the counter, and $A_L, 0 \le A_L \le (T_R - 2\delta)$, is the length of service for each video title being served. Therefore, each service node will have M such admission tuples and each admission controller will have MK such admission tuples, where K is the total number of service nodes in the system. Both the counter and the length fields are initially set to zero.

Now with the admission tuples, the admission procedure proceeds as follows. For requests that cannot be statically admitted, the admission controller will first check the counter in the admission tuple for the requested video title. If the counter A_C is zero, then the counter is incremented by one, and the length field is set according to

$$A_L = a_i - t_m \tag{3}$$

which is the length of time passed since the beginning of the last multicast. In other words, this particular client will occupy the dynamic channel for a duration of A_L seconds for patching purpose. At the same time, a START request carrying the requested video title and the length field A_L will be sent to a service node and the admission controller enters the STARTED state.

If another request for the same video title arrives during the STARTED state, say at time a_{i+1}, the admission controller will not send another request to the service node, but just update the local length field according to

$$A_L = a_{i+1} - t_m. \tag{4}$$

This process repeats for all subsequent requests arrived during the STARTED state. As a result, only one START request will be sent to the service node regardless of how many requests are received during the STARTED state, thereby significantly reducing the processing overhead at the service node.

At the service node side, upon receiving a START request from the admission controller, the service node will wait for a free channel from the N_D dynamic multicast channels to start transmitting the video title for a duration of A_L seconds as shown in Fig. 4. Once a channel becomes available, a START reply will be sent back to all admission controllers to announce the commencement of the new transmission.

The admission controllers, upon receiving the START reply, will do one of two things. If the local counter value is one, then both the counter and the length fields are zeroed and the admission process is completed. If the counter is larger than one, then the admission controller will send an EXTEND request to the service node to extend the transmission duration according to the value of the local length field A_L. Note that, in this case, the length field at the admission controller will be larger than the length field at the service node because only the length field at the admission controller is updated for subsequent requests for the same video title. The length field at the service node is always the one for the first request. After receiving EXTEND requests from the admission controllers, the service node will update the transmission duration to the largest one among all EXTEND requests. Transmission will stop after the specified

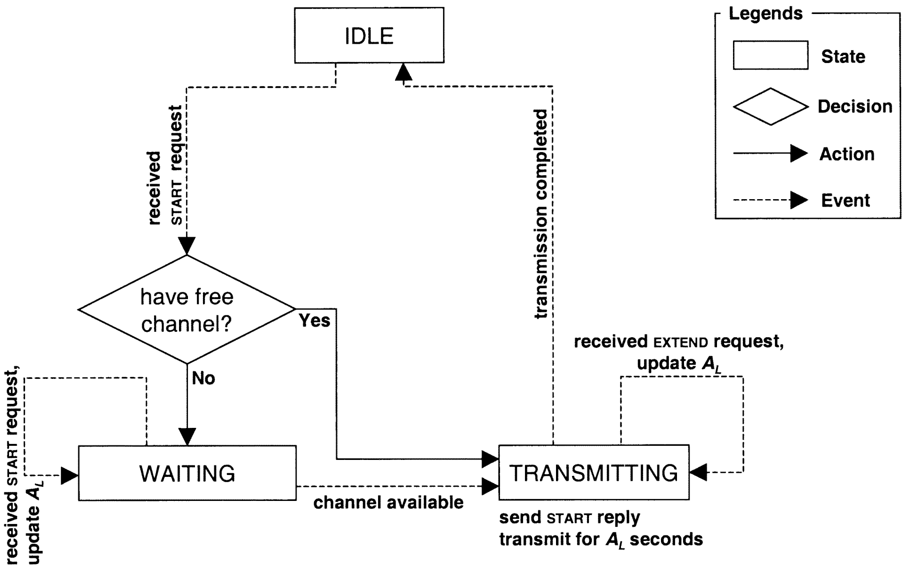

Fig. 4. State-transition diagram for a service node.

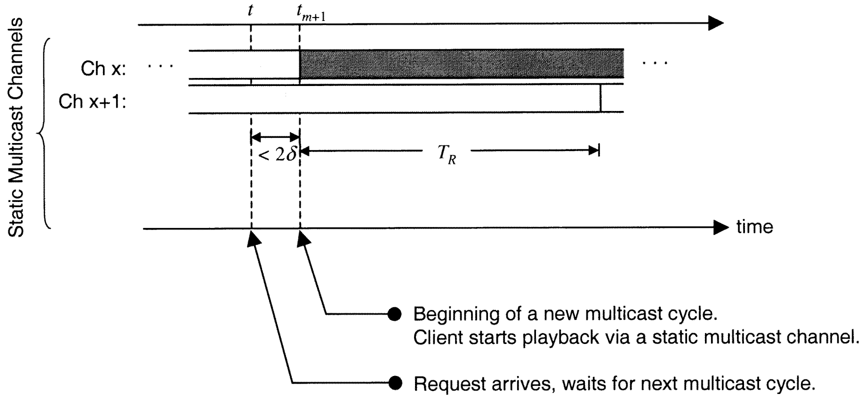
Static Multicast Channels
t
t_{m+1}
Ch x:
Ch x+1:
$< 2\delta$
T_R
time
Beginning of a new multicast cycle.
Client starts playback via a static multicast channel.
Request arrives, waits for next multicast cycle.

Fig. 5. Timing diagram for a statically admitted client.

transmission duration expires. Note that the service node does not need to wait for any EXTEND request to begin streaming. Streaming will begin as soon as a free dynamic channel becomes available. The purpose of the EXTEND request is to increase the transmission time of the dynamic channel to cater for subsequent requests in the same batch that require a longer patching duration.

It may appear that the previous admission procedure is unnecessarily complex and the clients are better off sending requests directly to the service nodes. However, this direct approach suffers from poor scalability. In particular, recall that each service node serves a few video titles to the entire user population. Therefore, as the user population grows, the volume of requests directed at a service node will increase linearly and eventually exceed the service node's processing capability.

By contrast, an admission controller generates at most two requests, one START request and one EXTEND request, for each dynamically started multicast transmission, irrespective of the actual number of client requests arrived in an admission cycle (i.e., from receiving the first request in a batch to sending the EXTEND request). Given that the number of admission controllers is orders of magnitude smaller than the user population, the processing requirement at the service nodes is substantially reduced. For extremely large user populations where even requests from admission controllers can become overwhelming, one can extend this request-consolidation strategy into a hierarchical structure by introducing additional layers of admission controllers to further consolidate requests until the volume becomes manageable by the service nodes.

C. Channel Merging

According to the previous admission control policy, a statically admitted client starts receiving streaming video data from a static multicast channel for playback as depicted in Fig. 5. For dynamically admitted clients, video playback starts with video data received from a dynamically allocated multicast channel. To merge the client back to an existing static multicast channel, the client concurrently receives and caches video data from a nearby (in time) static multicast channel as illustrated in the timing diagram in Fig. 6. Eventually, playback will reach the point where the cached data began and the client can then release the dynamic multicast channel. Playback then continues using data received from the static multicast channel.

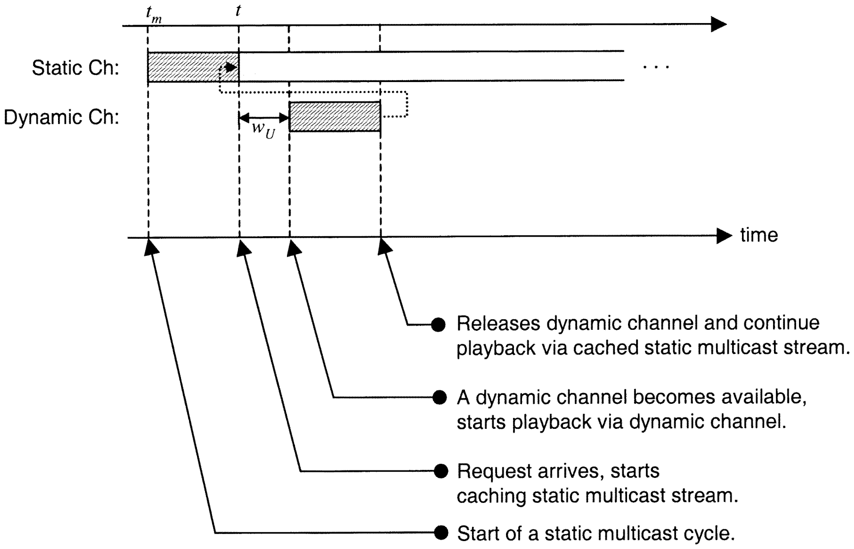
t_m
t
Static Ch:
Dynamic Ch:
w_U
time
Releases dynamic channel and continue playback via cached static multicast stream.
A dynamic channel becomes available, starts playback via dynamic channel.
Request arrives, starts caching static multicast stream.
Start of a static multicast cycle.

Fig. 6. Timing diagram for a dynamically admitted client.

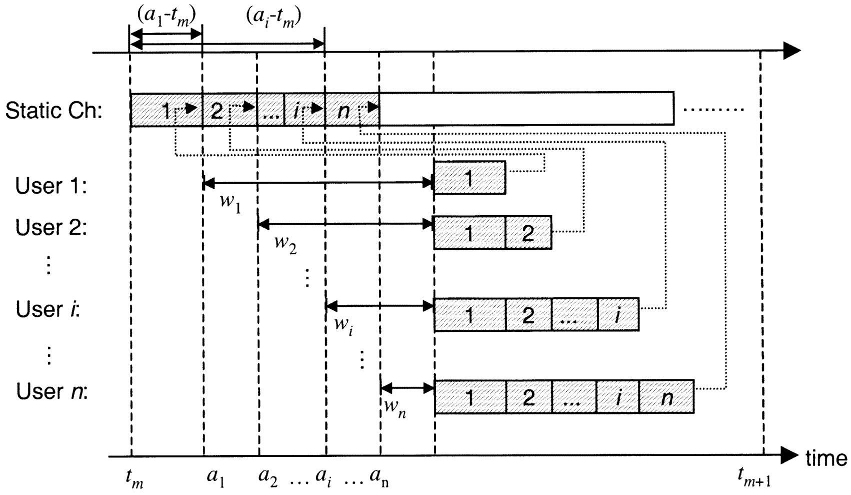

Fig. 7. Timing diagram for admitting a group of dynamically admitted users.

As an illustration, consider a dynamic multicast channel serving n dynamically admitted clients as shown in Fig. 7. Let a_i be the time client i arrives at the system and the nearest multicast cycle starts at t_m and t_{m+1}, respectively, where $t_m < a_1 < a_2, \ldots, < a_n < (t_{m+1} - 2\delta)$. Each client upon arrival will begin caching data from a static multicast channel while waiting for an available dynamic channel to begin playback. Note that the later a client arrives in the batch, the longer it must receive data from the dynamic multicast channel to make up for the missed data transmitted by the static multicast channel. Eventually all clients in the batch will reach their cached data position and the dynamic multicast channel is released. Therefore, the channel holding time of the dynamic multicast channel is equal to $(a_n - t_m)$, i.e., dominated by the last client joining the batch.

Compared to TVoD systems, a SS-VoD client must have the capability to receive two multicast channels concurrently and have a local buffer large enough to hold up to T_R seconds of video data. Given a video bit rate of 3 Mb/s (e.g., high-quality MPEG4 video), a total of 6 Mb/s downstream bandwidth will be needed for the initial portion of the video session. For a 2-h movie served using 25 static multicast channels, the buffer requirement will become 108 MB. This can easily be accommodated today using a small hard disk on the client side, and in the near future simply using memory as technology improves.

IV. Interactive Controls

To provide a complete VoD service, interactive playback controls such as pause–resume, slow motion, seeking, etc. will also need to be supported. Among these, pause–resume is likely the control most frequently performed in typical movie-on-demand applications. Intuitively, performing an interactive control in SS-VoD essentially breaks the client away from the current static multicast channel and then restarts it at another point within the video stream. Hence, a simple method to support interactive control is to treat them just like a new request. Clearly, this approach will increase loads at the dynamic multicast channels and result in increased waiting time for both new session and interactive control requests. As there is no generally accepted user-activity model, we do not attempt to quantify the performance impact of this approach in this study.

In Sections IV-A through IV–C, we present algorithms that take advantage of the staggered static multicast schedule to support pause–resume, slow motion, and seeking in SS-VoD in a resource-free way. In other words, no additional server resource or client buffer is needed to support these interactive controls in SS-VoD.

A. Pause-Resume

We use a simple channel-hopping algorithm to implement pause–resume in SS-VoD. Specifically, since a client has a buffer large enough to cache T_R seconds of video, it can just continue buffering incoming video data after the user paused playback. If the user resumes playback before the buffer is full, then no further action is required. By contrast, if the buffer becomes full, then the client simply stops receiving data and enters an idle state.

When the user resumes playback, the client can resume playback immediately and at the same time determine the nearest multicast channel that is currently multicasting the video. Since a movie is repeated every T_R seconds and the client buffer already contains T_R seconds worth of video data, we can guaranteed that the client can locate and merge back to an existing static multicast channel.

B. Slow Motion

Slow motion is playback at a rate lower than the normal playback rate. As video data are always being transmitted and received at the normal video bit rate, it is easy to see that once slow motion is started data will begin to accumulate in the client buffer. Now, if the user resume normal speed playback before the buffer is full, then no additional action needs to be done.

However, if playback continues in the slow motion state for a sufficiently long duration, the client buffer will eventually be fully filled up with video data. Note that at the instant when the buffer becomes full, the buffer will contain T_R seconds worth of video data. This is equivalent to the buffer full state in performing a pause operation. The only difference is that, in performing pause, the client will stop receiving data until the user resumes playback, at which time a nearby (in time) multicast channel will be located to merge back into. For slow motion, however, playback continues at that instant and hence it is necessary to immediately locate a nearby multicast channel other than the current one to merge back into. As any play point is at most T_R seconds away due to the staggered static multicast schedule, the T_R seconds worth of data in the buffer guarantees that the client can locate and merge back into a static multicast channel. If slow motion continues after merging, then data will begin to accumulate in the buffer again and the cycle repeats until normal playback speed is resumed.

Using this algorithm, slow motion at any rate slower than the normal playback rate can be supported without the need for any additional resource from the server. Client buffer requirement also remains the same.

C. Seeking

Seeking is the change from one playback point to another. Typically, the user initiates seeking either by giving a new destination time offset or by means of using a graphical user interface such as a slider or a scroll bar. SS-VoD can support different types of seeking depending on the seek direction, seek distance, and the state of the client buffer and static multicast channels. Specifically, due to patching, the client buffer typically has some advance data cached. Moreover, some past video data will also remain in the client buffer until being overwritten with new data. Hence, if the new seek position is within the range of video data in the client buffer, seeking can be implemented simply by changing the playback point internally.

Now, if the seek position, denoted by t_s, lies outside the client buffer, then the client may need to switch multicast channels to accomplish the seek. Let $t_i, i = 0, 1, \ldots, N_S - 1$ be the current playback points of the N_S static multicast channels and assume the client is currently on channel x. Then the client will choose the nearest channel to restart playback by finding the channel j such than the seek error $\varepsilon = \min\{|t_j - t_s|\}$ is minimized. Note that the current channel may happen to be the nearest channel and, in this case, the client simply seeks to the oldest data in the buffer if t_s is earlier than the current playback point or seeks to the newest data in the buffer otherwise.

Clearly, in the previous case, the seek operation may not end up in the precise location specified by the user and the seek error can be up to $T_R/2$ seconds. In return, this seeking algorithm can be supported without incurring server overhead or additional client buffer. If more precise seeking is needed, then one will need to make use of a dynamic multicast channel to merge the client back to an existing static multicast channel. Further research will be needed to develop efficient yet precise seeking algorithms.

V. Performance Modeling

In this section, we present an approximate performance model for the SS-VoD architecture. While an exact analytical solution does not appear to be tractable, we were able to derive an approximate model that can be solved numerically. The purpose of this performance model is to assist system designers to quickly evaluate various design options and to perform preliminary system dimensioning. Once the approximate system parameters are known, one can resort to a more detailed simulation to obtain more accurate performance results.

The primary performance metric we use in this study is startup latency, defined as the time a client submitted a request to the admission controller to the time the beginning of the requested video starts streaming. For simplicity, we assume there is a single video title stored in a service node and ignore network delay, transmission loss, and processing time at the admission controller.

In Sections V-A through F, we will first derive the average waiting time for statically admitted clients and dynamically admitted clients and then investigate the configuration of the admission threshold and the channel partitioning policy. We will compare results computed using this approximate performance model with simulation results in Section VI-A.

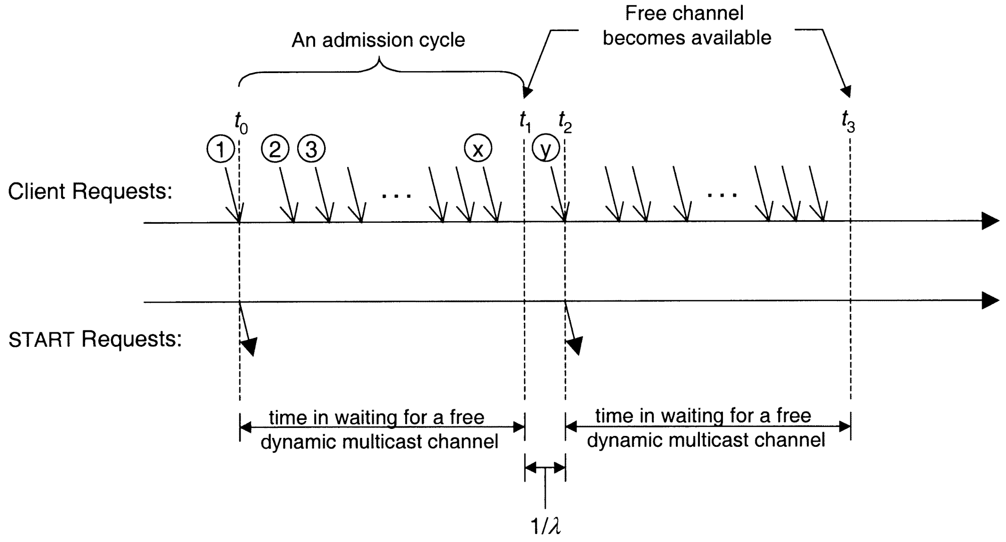
An admission cycle
Free channel becomes available
t_0
t_1
t_2
t_3
1
2
3
x
y
Client Requests:
...
START Requests:
time in waiting for a free dynamic multicast channel
time in waiting for a free dynamic multicast channel
1/λ

Fig. 8. Classification of dynamically admitted users.

A. Waiting Time for Statically Admitted Clients

As described in Section III-B, there are two ways to admit a client into the system. The first way is admission through a static multicast channel as shown in Fig. 5. Given that any clients arriving within the time window of 2δ seconds will be admitted this way, it is easy to see that the average waiting time for statically admitted clients, denoted by $W_S(\delta)$, is equal to half of the admission threshold

$$W_S(\delta) = \delta \tag{5}$$

assuming it is equally probable for a request to arrive at any time within the time window.

B. Waiting Time for Dynamically-Admitted Clients

The second way to admit a new client is through a dynamic multicast channel as shown in Fig. 6. Unlike static multicast channels, dynamic multicast channels are allocated in an on-demand basis according to the admission procedure described in Section III-B. Specifically, if there are one or more free channels available at the time a request arrives, a free channel will be allocated to start transmitting video data to the client immediately and the resultant waiting time will be zero.

On the other hand, if there is no channel available at the time a request arrives, then the resultant waiting time will depend on when a request arrive and when a free dynamic multicast channel becomes available. Specifically, requests arriving at the admission controller will be consolidated using the procedure described in Section III-B where the admission controller will send a consolidated START request to a service node to initiate video transmission.

Fig. 8 illustrates this admission process. This example assumes that there is no request waiting and all dynamic multicast channels are occupied before client request 1 arrives. After receiving request 1, the admission controller sends a START request to a service node to initiate a new multicast transmission for this request. However, as all channels are occupied, the transmission will not start until a later time t_1 when a free channel becomes available. During this waiting time, additional client requests such as request 2, 3, and so on arrive but the admission controller will not send additional START requests to the service node. This process repeats when a new request arrives at time t_2.

Based on this model, we first derive the average waiting time experienced by a START request at the service node. For the arrival process, we assume that user requests form a Poisson arrival process with rate λ. The proportion of client requests falling within the admission threshold is given by

$$P_S = \frac{2\delta}{T_R} \tag{6}$$

and these clients will be statically admitted.

Correspondingly, the proportion of dynamically admitted clients is equal to $(1 - P_S)$. We assume that the resultant arrival process at the admission controller is also Poisson, with a rate equal to

$$\lambda_D = (1 - P_S)\lambda. \tag{7}$$

Referring to Fig. 8, we observe that the time between two adjacent START requests is composed of two parts. The first part is the waiting time for a free dynamic multicast channel; and the second part is the time until a new dynamically admitted client request arrives. For the first part, we let $W_C(\delta)$ be the average waiting time for a free dynamic multicast channel given δ. To derive the second part, we first note that the mean interarrival time between the two requests (request x and y in Fig. 8) immediately before and after a free dynamic channel becomes available, called event E, is equal to $2/\lambda_D$, or *twice* the normal mean interarrival time. This counterintuitive result is due to the fact that longer interval is more likely to be encountered by the event E. With an interarrival time that is exponentially distributed with mean $1/\lambda_D$, the length-biased mean interarrival time as observed by the event E will become $2/\lambda_D$ [41]. Next we observe that the event E is equally likely to occur within the interval between the two requests, thus the mean time until the next arrival is simply half the length of the interval or $1/\lambda_D$.

Therefore, the interarrival time for START requests is given by

$$\frac{1}{\lambda_S} = W_C(\delta) + \frac{1}{\lambda_D} \tag{8}$$

where λ_S is the arrival rate for START requests. For simplicity, we assume that the arrival process formed from START requests is also a Poisson process.

For the service time of start request, it depends on the last user joining the batch (Fig. 7). In particular, the service time of the last user equals to the arrival time a_n minus the time t_{m-1} for the previous multicast of the requested video title. The service time, denoted by s, can range from 0 to $(T_R - 2\delta)$. We assume the service time s is uniformly distributed between

$$0 < s < T_R - 2\delta. \tag{9}$$

Therefore, the dynamic multicast channels form a multi-server queueing system with Poisson arrival and uniformly distributed service time. As no closed-form solution exists for such a queueing model, we resort to the approximation by Allen and Cunneen [39] for G/G/m queues to obtain the average waiting time for a dynamic multicast channel

$$W_C(\delta) = \frac{E_C(N_D, u)}{N_D(1-\rho)} \left(\frac{C_A^2 + C_S^2}{2}\right) T_S \tag{10}$$

where $C_A^2 = 1$ is the coefficient of variation for Poisson process and

$$C_S^2 = \frac{(T_R - 2\delta)^2}{12} \left(\frac{2}{T_R - 2\delta}\right)^2 = \frac{1}{3} \tag{11}$$

is the coefficient of variation for uniformly distributed service time and T_S is the average service time, given by

$$T_S = \frac{T_R - 2\delta}{2}. \tag{12}$$

Additionally, $u = \lambda_S T_S$ is the traffic intensity, $\rho = u/N_D$ is the server utilization, and $E_C(N_D, u)$ is the Erlang-C function

$$E_C(N_D, u) = \frac{\frac{u^{N_D}}{N_D!}}{\frac{u^{N_D}}{N_D!} + (1-\rho) \sum_{k=0}^{N_D-1} \frac{u^k}{k!}}. \tag{13}$$

Since the traffic intensity depends on the average waiting time, and the traffic intensity is needed to compute the average waiting time, (10) is in fact recursively defined. Due to (13), (10) does not appear to be analytically solvable. Therefore, we apply numerical methods to solve for $W_C(\delta)$ in computing the numerical results presented in Section VI.

Now that we have obtained the waiting time for a START request, we can proceed to compute the average waiting time for dynamically admitted client requests. Specifically, we assume the waiting time for a START request is exponentially distributed with mean $W_C(\delta)$. We classify client requests into two types. A Type-1 request is the first request that arrives at the beginning of the admission cycle. Type-2 requests are the other requests that arrive after a Type-1 request. For example, request 1 in Fig. 8 is a Type-1 request, and requests 2 and 3 are Type-2 requests.

We first derive the average waiting time for Type-2 requests. Let $W_2(\delta)$ be the average waiting time for Type-2 requests which can be shown to be (please refer to the Appendix)

$$W_2(\delta) = W_C(\delta) \left(1 - \left(\frac{1 + \frac{(T_R - 2\delta)}{2W_C(\delta)}}{1 - e^{(-(T_R - 2\delta)/W_C(\delta))}}\right) \times \frac{(T_R - 2\delta)}{W_C(\delta)} e^{(-(T_R - 2\delta)/W_C(\delta))}\right). \tag{14}$$

Next, for Type-1 requests, the average waiting time, denoted by $W_1(\delta)$, is simply equal to $W_C(\delta)$. Therefore, the overall average waiting time, denoted by $W_D(\delta)$, can be computed from a weighted average of Type-1 and Type-2 requests

$$W_D(\delta) = \frac{W_1(\delta) + M_2(\delta) W_2(\delta)}{1 + M_2(\delta)} \tag{15}$$

where $M_2(\delta)$ is the expected number of Type-2 requests in an admission cycle and can be computed from

$$M_2(\delta) = W_C(\delta) \lambda_D. \tag{16}$$

C. Admission Threshold

In the previous derivations, we have assumed that the admission threshold value is given *a priori*. Consequently, the resultant average waiting time for statically admitted and dynamically admitted users may differ. To maintain a uniform average waiting time for both cases, we can adjust the admission threshold such that the average waiting time difference is within a small error ε

$$\delta = \min\left\{x \middle| (W_S(x) - W_D(x)) \le \varepsilon, T_R \ge x \ge 0\right\}. \tag{17}$$

As adjusting the admission threshold does not affect existing users, the adjustment can be done dynamically while the system is online. In particular, the system can maintain a moving average of previous users' waiting time as the reference for threshold adjustment. This enables the system to maintain a uniform average waiting time for both statically admitted and dynamically admitted users. The term *latency* in this paper refers to this uniform average waiting time.

D. Channel Partitioning

An important configuration parameter in SS-VoD is the partitioning of available channels for use as dynamic and static multicast channels. Intuitively, having too many dynamic multicast channels will increase the traffic intensity at the dynamic multicast channels due to increases in the service time [see (1) and (12)]. On the other hand, having too few dynamic multicast channels may also result in higher load at the dynamic multicast channels. We can find the optimal channel partitioning policy by enumerating all possibilities, which in this case is of $O(N)$. Unlike the case in UVoD [33], [37] where the optimal channel partition policy is arrival-rate-dependent, we found that the optimal channel partitioning policy is relatively independent of the user arrival rate in SS-VoD. This will be studied in more detail in Section VI-B.

VI. Performance Evaluation

In this section, we present simulation and numerical results to evaluate performance of the SS-VoD architecture. We first validate the analytical performance model using simulation results and then proceed to investigate the effect of the channel partitioning policy to compare latency and channel requirement between TVoD, NVoD, UVoD, with SS-VoD, and finally investigate the performance of SS-VoD under extremely light loads. The focus of the comparisons is on the server and backbone network resource requirements, represented by the number of channels required to satisfy a given performance metric such as latency. Note that for simplicity we do not distinguish between unicast and multicast channels and assume that they have the same cost. In practice, a multicast channel will incur higher costs in the access network where the network routers will need to duplicate and forward the multicast video data to multiple recipients. Nevertheless, this additional cost is not present at the server (e.g., using IP multicast) and at the backbone network before fanning out to the access subnetworks and therefore will be ignored in this study.

A. Model Validation

To verify accuracy of the performance model derived in Section V, we developed a simulation program using the Communications Network Class Library (CNCL) [40] to obtain simulation results for comparison. A set of simulations is run to obtain the latency over a range of arrival rates. Each run simulated a duration of 1440 h (60 days), with the first 24 h of data skipped to reduce initial condition effects. There is one movie in the system, with a length of 120 min. We divide available multicast channels equally into static-multicast and dynamic-multicast channels. We do not simulate user interactions and assume that all users playback the entire movie from start to finish.

Fig. 9 shows the latency versus arrival rate ranging from 1×10^{-3} to 5.0 requests per second. We observe that the analytical results are reasonable approximations for the simulation results. At high arrival rates (e.g., over 1 request/s), the analytical results overestimate the latency by up to 5%. As discussed in the beginning of Section V, the analytical model is primarily used for preliminary system dimensioning. Detailed simulation, while lengthy (e.g., hours), is still required to obtain accurate performance results.

B. Channel Partitioning

To investigate the performance impact of different channel allocations, we conducted simulations with the proportion of dynamic multicast channels, denoted by r, ranging from 0.3 to 0.7. The results are plotted in Fig. 10. Note that we use a normalized latency instead of actual latency for the y axis to facilitate comparison. Normalized latency is defined as

$$\frac{w(r)}{\min\{w(r),\ \forall\, r\}} \tag{18}$$

where $w(r)$ is the latency with $r \times N$ dynamic multicast channels.

We simulated three sets of parameters with $N = 20$, 30, and 50 for two arrival rates, namely heavy load at 5 requests/s and light load at 0.5 requests/s. Note that normalized latency obtained from two different values of N cannot be compared directly as the denominator in (18) is different.

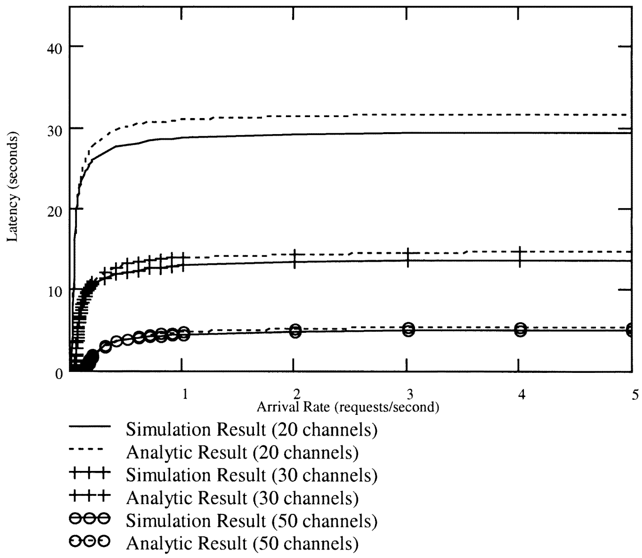

Fig. 9. Comparison of latency obtained from analysis and simulation.

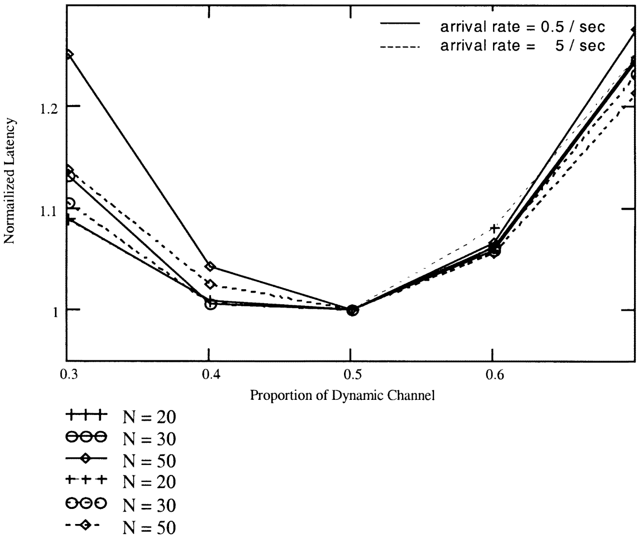

Fig. 10. Effect of channel partitioning on latency.

Surprisingly, the results show that in all cases the latency is minimized by assigning half of the channels to dynamic multicast and the other half to static multicast. For comparison, UVoD exhibits a different behavior and requires more channels allocated for static multicast to minimize latency at high loads as shown in Fig. 11 for a 50-channel configuration.

UVoD's behavior is explained by the observation that, at higher arrival rates, the waiting time for a free unicast channel increases rapidly near full utilization. Therefore, it is desirable to allocate more multicast channels to reduce the traffic intensity (arrival rate $\times T_R$) routed to the unicast channel to prevent operating the unicast channels near full utilization. By contrast, the same situation does not occur in SS-VoD because a dynamic multicast channel can batch and serve multiple waiting

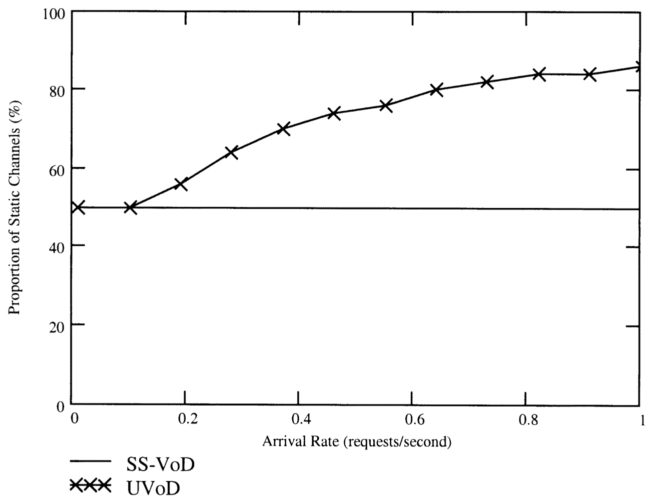

Fig. 11. Comparison of optimal channel allocation in SS-VoD and UVoD.

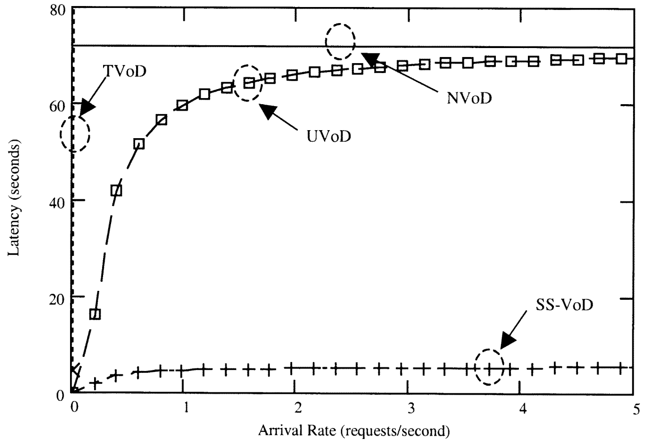

Fig. 12. Comparison of latency for different arrival rates.

requests. Moreover, the batching efficiency increases for longer waiting time, thus compensating for the increases in the arrival rate. This remarkable property of SS-VoD greatly simplifies system deployment as one will not need to reconfigure the system with a different channel partition policy in case the user demand changes.

C. Latency Comparisons

Fig. 12 plots the latency for SS-VoD, UVoD, TVoD, and NVoD for arrival rates up to 5 requests/s. The service node (or video server for TVoD/NVoD) has 50 channels and serves a single movie of length 120 min. The first observation is that, except for NVoD, which has a constant latency of 72 s, the latency generally increases with higher arrival rates as expected. For TVoD, the server overloads for arrival rates larger than 1.16×10^4 requests/s. UVoD performs significantly better with the latency asymptotically approaches that of NVoD. SS-VoD performs even better than UVoD, and the latency levels off and approaches 5.6 s or a 92% reduction compared to UVoD.

It is also worth noting that the performance gain of SS-VoD over UVoD does not incur any tradeoff at the client side. Specifically, the buffer requirement and bandwidth requirement are the same for both SS-VoD and UVoD. The only differences are the replacement of the dynamic unicast channels in UVoD with dynamic multicast channels in SS-VoD and the addition of the more complex admission procedure in the admission controller.

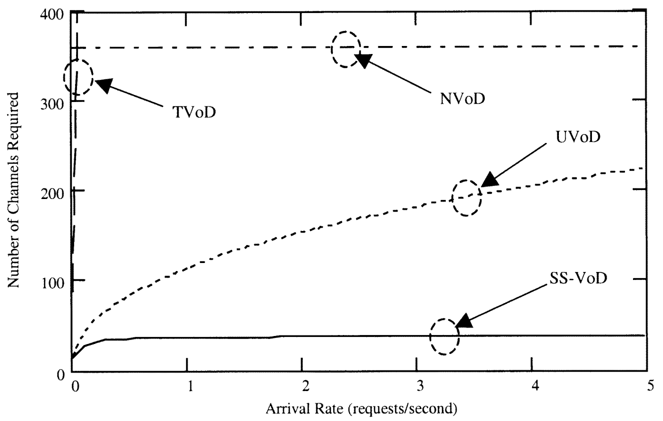

Fig. 13. Comparison of channel requirement for different arrival rates.

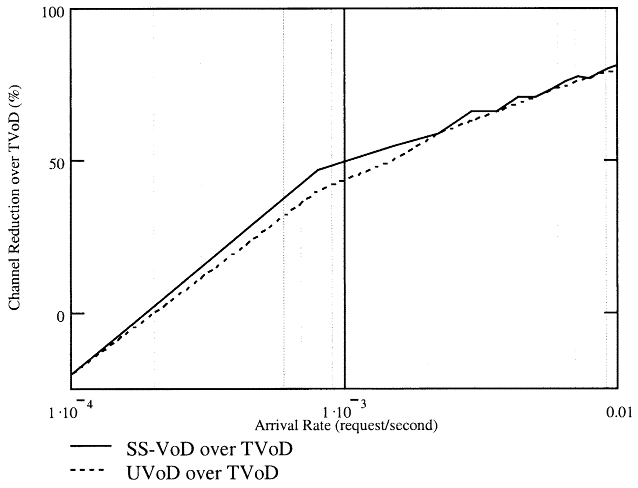

Fig. 14. Channel reduction over TVoD at very low arrival rates.

D. Channel Requirement

Channel requirement is defined as the minimum number of channels needed to satisfy a given latency constraint at a certain arrival rate. Fig. 13 plots the channel requirements of SS-VoD, UVoD, TVoD, and NVoD versus arrival rates from 0.01 to 5 requests/s. The latency constraint is set to 10 s.

The number of channels required for NVoD is a constant value and equal to 360. The channel requirement of TVoD increases with the arrival rate and quickly exceeds that of NVoD. The channel requirements of SS-VoD and UVoD are significantly lower than both TVoD and NVoD. For higher arrival rates, SS-VoD outperforms UVoD by a wide margin. For example, the channel requirements at 1 request/s are 114 and 36 for UVoD and SS-VoD, respectively, and the channel requirements at five requests per second are 225 and 38 for UVoD and SS-VoD, respectively. This result demonstrates the performance gain achieved by replacing the dynamic unicast channels in UVoD with dynamic multicast channels in SS-VoD.

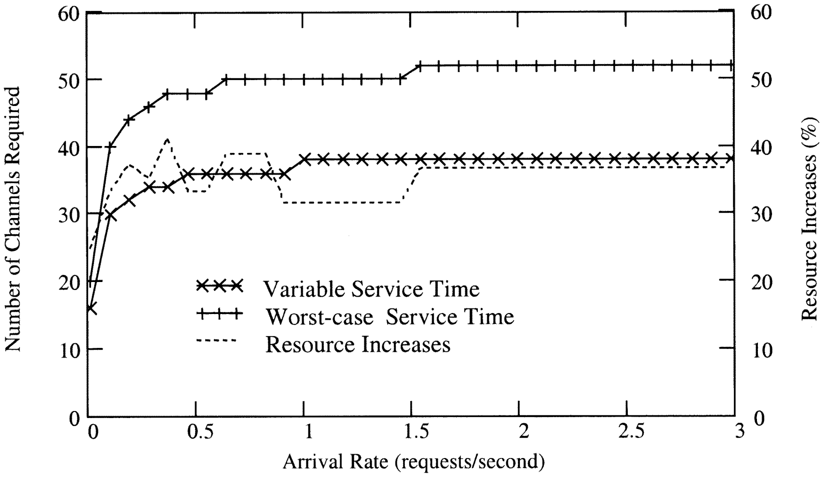

Fig. 15. Performance tradeoff for using worst case service time for dynamic channels.

E. Performance at Light Loads

The previous results are computed using relatively high arrival rates. Intuitively, the performance gains will decrease at lower arrival rates as fewer requests will be served by a dynamic multicast channel. To investigate this issue, we define a percentage of channel reduction over TVoD, denoted by G, as shown in (19), shown at the bottom of the page, where $W_{\mathrm{TVoD}}(n)$ and $W(n)$ are the latency given there are n channels, for TVoD and SS-VoD/UVoD, respectively.

Fig. 14 plots the channel reduction for arrival rates from 1×10^{-4} to 0.01 for SS-VoD and UVoD. The results show that SS-VoD requires fewer channels than TVoD as long as arrival rates are over 1.8×10^{-4} requests per second. Note that at this low arrival rate both TVoD and SS-VoD require only six channels. This suggests that SS-VoD will likely outperform TVoD in practice.

F. Simplicity Versus Performance Tradeoff

The admission controller is among the more complex components in the SS-VoD architecture. One way to simplify the admission controller is to use a constant service time of $T_R - 2\delta$ seconds for the dynamic channels. As this is the worst case service time, the admission controller no longer needs to maintain the counter-length tuple $\{A_C, A_L\}$ and also does not need to send an EXTEND update request to the service node. The tradeoff for this simplification is increased channel requirement as the dynamic channel will be occupied for longer than necessary. Fig. 15 compares the two cases, showing that using the worst-case service time of $T_R - 2\delta$ seconds results in resource increases of over 30%. This shows that the more complex admission procedure is still desirable unless system complexity must be minimized.

VII. Implementation and Benchmarking

We implemented a SS-VoD prototype using off-the-shelf software and hardware. There are three components in the prototype: service node, admission controller, and video clients. Both the service node and the admission controller are implemented using the C++ programming language running on Red Hat Linux 6.2. Two client applications have been developed: one is implemented using the Java programming language and the Java Media Framework (JMF) 2.1 while the other is implemented using C++ on the Microsoft Windows platform. Both the service node and the admission controller are video-format-independent. The Java-version client supports MPEG1 streams, while the Windows-version client supports MPEG1, MPEG2, as well as basic MPEG4 streams. We also implemented the interactive playback controls presented in Section IV, namely pause–resume, slow motion, and seeking.

With the SS-VoD prototype, we conducted extensive experiments to obtain measured benchmark results to verify against the analytical and simulation results. We developed a traffic generator in order to simulate a large number of client requests. The service node runs on a Compaq Proliant DL360 serving one movie of length 120 min with 30 channels, each at 1.5 Mb/s. The clients are ordinary PCs and all machines are connected using a layer-3 IP switch with hardware IP multicast support. We measured the startup latency for arrival rates ranging from 1 to 5 requests/s. Each benchmark test runs for a total of six hours. Benchmark data collected during the first hour is discarded to reduce initial condition effect.

Fig. 16 compares the startup latencies obtained from analysis, simulation, and benchmarking, respectively. We observe that the benchmarking results agree very well with the analytical results and simulation results. Note that the latencies obtained from benchmarking are consistently larger than those obtained from

$$G = \frac{\min\{n|W_{\mathrm{TVoD}}(n) \le 1,\ \forall\, n = 0, 1, \ldots\} - \min\{n|W(n) \le 1,\ \forall\, n = 0, 1, \ldots\}}{\min\{n|W_{\mathrm{TVoD}}(n) \le 1,\ \forall\, n = 0, 1, \ldots\}} \times 100\% \tag{19}$$

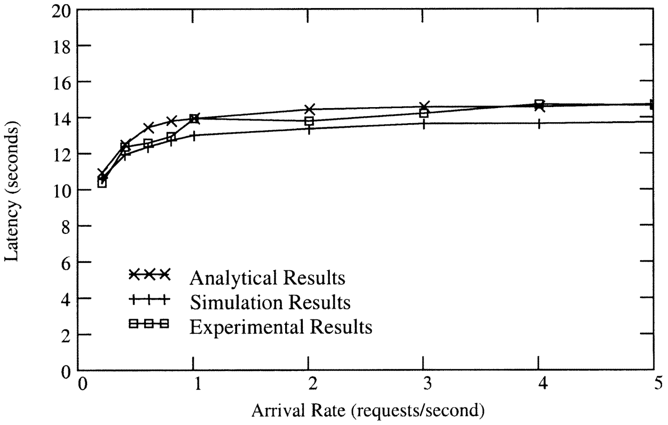

Fig. 16. Comparison of latencies obtained from analysis, simulation, and benchmarking.

simulation. We believe that this is due to the nonzero processing delay and network delay in the system, both of which have been ignored in the simulation model.

VIII. Conclusion

In this study, we investigated a SS-VoD architecture that can achieve super-linear scalability by integrating static multicast, dynamic multicast, and client-side caching. This SS-VoD architecture is particularly suitable for metropolitan-scale deployment as resource savings increase dramatically at higher arrival rates. In fact, there is no inherent scalability limit to this SS-VoD architecture provided that the network is multicast-ready and has sufficient bandwidth to connect all customers. With more and more existing residential broad-band networks being upgraded to support native multicast, the SS-VoD architecture could provide a cost-effective yet simple-to-implement and easy-to-deploy solution for interactive VoD services.

Appendix

In this appendix, we derive the mean waiting time for Type-2 users, denoted by $W_2(\delta)$. The complication is due to length biasing as a Type-2 user is more likely to observe a longer Type-1 wait than a shorter Type-1 wait. First, we compute the waiting time distribution for Type-1 users, denoted by $f'_C(t)$, *as observed by a Type-2 user* using the results from Kleinrock [41]

$$f'_C(t) = \frac{t f_C(t)}{W_C(\delta)} \tag{A1}$$

where $f_C(t)$ and $W_C(\delta) = E[f_C(t)]$ is the actual waiting time distribution and mean waiting time of Type-1 users, respectively. Let $W'_C(\delta)$ be the mean of $f'_C(t)$ as

$$W'_C(\delta) = \int_{-\infty}^{\infty} t f'_C(t) dt. \tag{A2}$$

Substituting (A1) into (A2), we then have

$$W'_C(\delta) = \int_{-\infty}^{\infty} \frac{t^2 f_C(t)}{W_C(\delta)} dt. \tag{A3}$$

We note that the waiting time can only range from zero to $(T_R - 2\delta)$, so we can rewrite (A3) as

$$W'_C(\delta) = \int_0^{T_R - 2\delta} \frac{t^2 f_C(t)}{W_C(\delta)} dt. \tag{A4}$$

Motivated by simulation results, we assume that $f_C(t)$ is truncated exponentially distributed

$$f_C(t) = \left(\left(1 - e^{(-(T_R-2\delta)/W_C(\delta))}\right) W_C(\delta)\right)^{-1} e^{(-t/W_C(\delta))}. \tag{A5}$$

Substituting (A5) into (A4), we have

$$W'_C(\delta) = \int_0^{T_R-2\delta} \frac{t^2 e^{(-t/W_C(\delta))}}{(1 - e^{(-(T_R-2\delta)/W_C(\delta))}) W_C(\delta)^2} dt. \tag{A6}$$

Solving the integral and after a series of simplifications (A6) becomes

$$W'_C(\delta) = 2W_C(\delta) \left(1 - \left(\frac{1 + \frac{(T_R - 2\delta)}{2W_C(\delta)}}{1 - e^{(-(T_R-2\delta)/W_C(\delta))}}\right) \times \frac{(T_R - 2\delta)}{W_C(\delta)} e^{(-(T_R-2\delta)/W_C(\delta))}\right). \tag{A7}$$

Finally, as a Type-2 user is equally likely to arrive anytime during a Type-1 wait, the mean waiting time is simply equal to half of the Type-1 mean wait as

$$\begin{aligned} W_2(\delta) =& \frac{W'_C(\delta)}{2} \\ =& W_C(\delta) \left(1 - \left(\frac{1 + \frac{(T_R - 2\delta)}{2W_C(\delta)}}{1 - e^{(-(T_R-2\delta)/W_C(\delta))}}\right) \times \frac{(T_R - 2\delta)}{W_C(\delta)} e^{(-(T_R-2\delta)/W_C(\delta))}\right). \end{aligned} \tag{A8}$$

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 VCR Control in, Movie-on-Demand Servers IBM Res. Rep.,”, Yorktown Heights, NY, RC19588, 1994.

[3] A. Dan, D. Sitaram, and P. Shahabuddin, “Dynamic batching policies for an on-demand video server,” *ACM Multimedia Syst.*, vol. 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 Systems*, June 1996, pp. 253–258.

[5] ——, “Exploring wait tolerance in effective batching for video-on-demand scheduling,” in *Proc. 8th Israeli Conf. Computer Systems and Software Engineering*, June 1997, pp. 67–76.

[6] S. Sheu and K. A. Hua, “Virtual batching: A new scheduling technique for video-on-demand servers,” in *Proc. 5th Int. Conf. Database Systems for Advanced Applications*, Melbourne, Australia, Apr. 1997, pp. 481–490.

[7] S. Sheu, K. A. Hua, and W. Tavanapong, “Chaining: A generalized batching technique for video-on-demand systems,” in *Proc. Multimedia Computing and Systems ’97*, Ottawa, ON, Canada, June 3–6, 1997, pp. 110–117.

[8] H. C. De-Bey, “Program Transmission Optimization,” U.S. Patent 5 421 031, Mar. 1995.

[9] T. C. Chiueh and C. H. Lu, "A periodic broadcasting approach to video-on-demand service," *Proc. SPIE*, vol. 2615, pp. 162–9, 1996.

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

[11] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *Proc. Int. Conf. Multimedia Computing and Systems*, June 1996, pp. 118–26.

[12] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1110–1122, Aug. 1996.

[13] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for meteropolitan video-on-demand system," in *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, Cannes, France, Sept. 1997, pp. 89–100.

[14] L. S. Juhn and L. M. Tseng, "Harmonic broadcasting for video-on-demand service," *IEEE Trans. Broadcasting*, vol. 43, pp. 268–271, Sept. 1997.

[15] ——, "Staircase data broadcasting and receiving scheme for hot video service," *IEEE Trans. Consumer Electron.*, vol. 43, pp. 1110–1117, Nov. 1997.

[16] E. L. Abram-Profeta and K. G. Shin, "Scheduling video programs in near video-on-demand systems," in *Proc. ACM Conf. Multimedia '97*, Seattle, WA, Nov. 1997, pp. 359–369.

[17] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proc. NOSSDAV '98*, Cambridge, U.K., July 1998.

[18] J. F. Pâris, S. W. Carter, and D. D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proc. 6th Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)*, July 1998, pp. 127–132.

[19] D. L. Eager and M. K. Vernon, "Dynamic skyscraper broadcasts for video-on-demand," in *Proc. 4th Int. Workshop on Multimedia Information Systems (MIS'98)*, Istanbul, Turkey, Sept. 1998, pp. 18–32.

[20] J. F. Pâris, S. W. Carter, and D. D. E. Long, "A low bandwidth broadcasting protocol for video on demand," in *Proc. 7th Int. Conf. Computer Communications and Networks (IC3N'98)*, Oct. 1998, pp. 690–697.

[21] Y. Birk and R. Mondri, "Tailored transmissions for efficient near-video-on-demand service," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, Florence, Italy, June 1999.

[22] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, pp. 51–62, 1997.

[23] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. 6th Int. Conf. Multimedia*, Sept. 1998, pp. 191–200.

[24] Y. Cai, K. Hua, and K. Vu, "Optimizing patching performance," in *Proc. SPIE/ACM Conf. Multimedia Computing and Networking*, San Jose, CA, Jan. 1999, pp. 204–215.

[25] S. W. Carter, D. D. E. Long, K. Makki, L. M. Ni, M. Singhal, and N. Pissinou, "Improving video-on-demand server efficiency through stream tapping," in *Proc. 6th Int. Conf. Computer Communications and Networks*, Sept. 1997, pp. 200–207.

[26] 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 Computing and Networking 2000 (MMCN 2000)*, San Jose, CA, Jan. 2000, pp. 206–215.

[27] 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. Measurement and Modeling of Computer Systems*, Ottawa, ON, Canada, May 1995, pp. 25–36.

[28] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers," *ACM Multimedia Syst.*, vol. 4, no. 30, pp. 14–55, 1996.

[29] S. W. Lau, J. C. S. Lui, and L. Golubchik, "Merging video streams in a multimedia storage server: Complexity and heuristics," *Multimedia Syst.*, vol. 6, no. 1, pp. 29–42, 1998.

[30] 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 Systems*, June 1996, pp. 253–8.

[31] J. H. Oh, K. A. Hua, and K. Vu, "An adaptive hybrid technique for video multicast," in *Proc. Int. Conf. Computer Communication and Networks*, Lafayette, LA, Oct. 1998.

[32] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, vol. 2, Florence, Italy, June 1999, pp. 117–121.

[33] J. Y. B. Lee, "UvoD—A unified architecture for video-on-demand services," *IEEE Commun. Lett.*, vol. 3, pp. 277–279, Sept. 1999.

[34] 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.

[35] V. C. H. Lee and J. Y. B. Lee, "Improving UVoD system efficiency with batching," in *Proc. Int. Conf. Software, Telecommunications and Computer Networks—SoftCOM*, Croatia, Oct. 2000.

[36] 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, pp. 440–56, Mar. 2001.

[37] J. Y. B. Lee, "On a unified architecture for video-on-demand services," *IEEE Trans. Multimedia*, vol. 4, pp. 38–47, Mar. 2002.

[38] W. Fenner, *Internet Group Management Protocol, Version 2, IETF RFC 2236*, 1997.

[39] A. O. Allen, *Probability, Statistics, and Queueing Theory With Computer Science Applications*, 2nd ed. New York: Academic, 1990.

[40] [Online]. Available: http://www.comnets.rwth-aachen.de/doc/cncl.html

[41] L. Kleinrock, *Queueing Systems Vol I: Theory*. New York: Wiley-Interscience, 1975, p. 171.

Jack Y. B. Lee (M '95) is an Associate Professor of the Department of Information Engineering at the Chinese University of Hong Kong. He directs the Multimedia Communications Laboratory and conducts research in distributed multimedia systems, fault-tolerant systems, multicast communications, and Internet computing.

C. H. Lee is a Researcher at the Department of Information Engineering at the Chinese University of Hong Kong. His research interest is in multicast communications and video streaming systems.