On a Unified Architecture for Video-on-Demand Services

Jack Y. B. Lee

***Abstract*—Current video-on-demand (VoD) systems can be classified into two categories: 1) true-VoD (TVoD) and 2) near-VoD (NVoD). TVoD systems allocate a dedicated channel for every user to achieve short response times so that the user can select what video to play, when to play it, and perform interactive VCR-like controls at will. By contrast, NVoD systems transmit videos repeatedly over multiple broadcast or multicast channels to enable multiple users to share a single video channel so that system cost can be substantially reduced. The tradeoffs are limited video selections, fixed playback schedule, and limited or no interactive control. TVoD systems can be considered as one extreme where service quality is maximized, while NVoD systems can be considered as the other extreme where system cost is minimized. This paper proposes a novel architecture called Unified VoD (UVoD) that can be configured to achieve cost-performance tradeoff anywhere between the two extremes (i.e., TVoD and NVoD). Assuming that a video client can concurrently receive two video channels and has local buffers for caching a portion of the video data, the proposed UVoD architecture can achieve significant performance gains (e.g., 400% more capacity for a 500-channel system) over TVoD under the same latency constraint. This paper presents the UVoD architecture, establishes a performance model, and analyzes UVoD's performance via numerical and simulation results.**

***Index Terms*—Near-video-on-demand (NVoD), performance analysis, true-video-on-demand (TVoD), unified architecture, UVoD, video-on-demand (VoD).**

I. Introduction

ALTHOUGH video-on-demand (VoD) systems have been around for many years, large-scale deployment is still uncommon. To provide a true-VoD (TVoD) service where the user can watch any movie at any time and with interactive VCR-like controls, the system must reserve dedicated video channels at the video server and the distribution network for the entire video-playback duration. As high-quality video consumes a large amount of transmission bandwidth even after compression, the cost of providing such a TVoD service for a large number of users is often prohibitive.

On the other hand, another type of VoD service commonly called near-VoD (NVoD) [1], [2] did find many successful applications, such as video services in hotel and paid-movies in cable TV. Unlike TVoD, NVoD transmits videos repeatedly over multiple broadcast or multicast channels to enable multiple users to share a single video channel so that system cost can be substantially reduced. The tradeoffs are limited video selections, fixed playback schedule (e.g., restarts every 15 min), and limited or no interactive control.

TVoD systems can be considered one extreme where service quality is maximized, while NVoD systems can be considered the other extreme where system cost is minimized. This paper proposes a novel architecture called Unified VoD (UVoD) that can be configured to achieve cost-performance tradeoff anywhere between the two extremes (i.e., TVoD and NVoD). Assuming that a video client can concurrently receive two video channels and has local buffers for caching a portion of the video data, the proposed UVoD architecture can achieve significant performance gains (e.g., 400% more capacity for a 500-channel system) over TVoD under the same latency constraint.

The rest of this paper is organized as follows. Section II discusses previous works in this area and compares them to UVoD; Section III presents the UVoD architecture in detail; Section IV derives an analytical model for performance evaluation; Section V presents performance results obtained numerically from the performance model; Section VI presents simulation results to validate the derived performance model and proposes an admission-rescheduling algorithm that can further improve the performance gain; Section VII discuss interactive-control issues and proposes an efficient way to support pause-resume; and Section VIII concludes the paper.

II. Background

In recent years, researchers have proposed various approaches to improve VoD system efficiency so that deploying large-scale VoD services can become more cost-effective. In Section II-A, we will briefly review the related approaches, and in Section II-B, we will discuss the differences of our approach.

A. Previous Works

One well-studied approach to improve VoD system efficiency is called *batching*. This approach has been proposed and studied by various researchers, including Dan *et al.* [3], Shachnai *et al.* [4], and Li *et al.* [5]. Dan *et al.* proposed serving queueing users requesting the same movie using a single multicast channel instead of multiple independent unicast channels to reduce resource requirement. They studied the performance of two batching policies (First Come First Served and Maximum Queue Length), representing tradeoffs between fairness and efficiency in [3]. Their simulation results showed that resource reductions of up to 70% can be realized

Manuscript received February 8, 1999; revised June 29, 2001. This work was supported in part by a Competitive Earmarked Grant (CUHK6095/99E) and a Direct Grant for Research from the HKSAR Research Grant Council and the Area-of-Excellence in Information Technology, a research grant from the HKSAR University Grants Council. The associate editor coordinating the review of this paper and approving it for publication was Prof. Shih-Fu Chang.

The author is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: jacklee@computer.org).

Publisher Item Identifier S 1520-9210(02)01394-9.

in large systems serving around 5000 concurrent users, with an average waiting time of around 1 min. Shachnai *et al.* refined the batching policies by incorporating knowledge of the next stream-completion time as well as viewer wait tolerance profile to reduce the server capacity required to achieve similar throughput and viewer turn-away probability.

Note that batching does not directly support interactive VCR-like controls as a multicast channel is shared by multiple users. To tackle this limitation, one can set aside some contingency channels to serve those users performing interactive controls as proposed and studied in Dan *et al.* [3], Li *et al.* [5], and Almeroth *et al.* [6].

Another innovative way to support interactive controls under batching is proposed by Li *et al.* [7]. Their Split-and-Merge protocol makes use of unicast channels and buffering to merge users broken away from a multicast channel (due to interactive control) back to an existing multicast channel. Specifically, a synchronization buffer is introduced between the server and the set-top box, where video stream from a nearby (in time) multicast channel can be cached while the user is temporary served by a unicast channel. Eventually, playback from the unicast channel will reach the point where the multicast channel is cached at the synchronization buffer. From this moment on, the playback is switched to the cached data and the unicast channel can then be released. The synchronization buffer essentially adds time delay to an existing multicast channel so that broken-away users can be merged back without a long delay as in traditional batching. Similar approachs have also been studied by Park *et al.* [8] and Abram-Profeta *et al.* [9].

Finally, Carter *et al.* [10] proposed another approach called *stream tapping* to improve video server efficiency. Unlike batching, where the efficiency gain is obtained through merging new video sessions, stream tapping employs active caching (or tapping) of video data from other concurrent video streams so that future video transmissions can be reduced. Their simulation results showed that stream tapping can achieve lower latency under the same load when compared to simple batching.

There are still other approaches to improving VoD system efficiency, such as pyramid broadcasting [11], [12], piggybacking [13], [14], and asynchronous multicasting [15], [16]. These approaches are not directly related to our work so we refer the interested readers to the cited literature.

B. Comparisons and Contributions

The primary contribution of this work is the unification of TVoD and NVoD into a single architecture. While TVoD and NVoD represent two extremes in cost-performance tradeoffs, UVoD enables one to trade off cost with performance on a continuous scale. Moreover, as a more general architecture, UVoD performs at least as good as, and often significantly better than, TVoD and NVoD under equivalent system parameters.

Second, existing batching approaches incur a substantial amount of delay during session start-up (in minutes) to achieve good performance gain. By contrast, one can achieve significant performance gain using UVoD at latencies as small as a few seconds. This enables UVoD to provide service qualities comparable to TVoD systems.

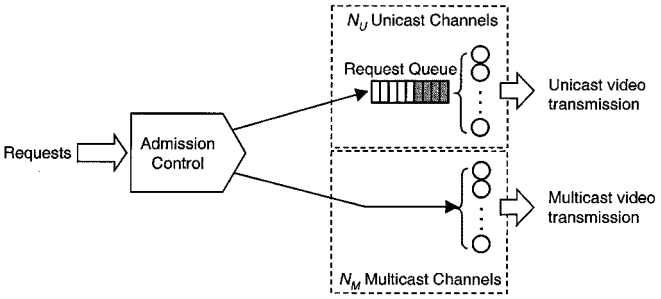

Fig. 1. Architecture of the UVoD system.

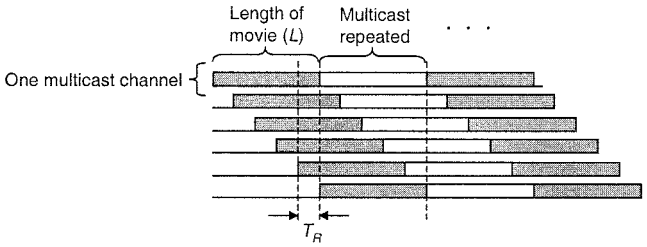

Fig. 2. Scheduling of multicast cycles for a movie.

Third, while most existing approaches employ some form of dynamic allocation of channels to movies, we propose a static allocation policy in UVoD. This static allocation approach not only simplifies system design and implementation, but also guarantees fairness and latency bounds for all movies. By contrast, performance of existing batching algorithms on a short time scale depends heavily on the arrival patterns as well as the particular movies being requested.[1]

Fourth, UVoD can support pause-resume user control without any additional resource requirement (e.g., contingency channels) at the server and network. This is possible due to the static channel-allocation and movie-scheduling policies.

Fifth, UVoD allows the service provider to safely dimension for a more conservative system size during initial deployment, knowing that the system can sustain any amount of additional loads (with tradeoff in latency) in case the actual demand exceeds the anticipated value.

Finally, this paper establishes a performance model for UVoD, derives a near-optimal channel partition policy for dividing available channels between unicast and multicast, and proposes a procedure to automatically adapt the admission threshold to maintain uniform latency. The derived performance model is then validated through simulation and shown to be reasonably accurate.

III. UVoD Architecture

Fig. 1 depicts the UVoD architecture. Let there be in total N available channels, of which N_U of them are unicast channels and $N_M = N - N_U$ are multicast channels. Let there be M movies of length L s each. For each multicast channel, the assigned movie is repeated over and over as in an NVoD system.

[1]For example, a short burst of requests for the same movie will result in good batching efficiency; while a sequence of requests (with inter-arrival time slightly longer than batching threshold) for an unpopular movie can lead to very poor resource utilization.

We divide the N_M multicast channels equally among those M movies so that each movie is multicasted over N_M/M multicast channels, assuming N_M is divisible by M. For multicast channels streaming the same movie, adjacent channels are offset by

$$T_R = \frac{L}{\left\lfloor \frac{N_M}{M} \right\rfloor} \tag{1}$$

seconds, as shown in Fig. 2.

The N_U unicast channels share the same request queue and serve incoming requests in a First-Come-First-Serve manner. Incoming requests will have to wait in the queue if all N_U unicast channels are occupied. For the video clients, we assume that they can receive up to two video channels simultaneously and have additional storage to cache up to T_R s of video data for later playback. Alternatively, an intermediate proxy, as proposed in [7], can be employed to perform the caching function for multiple clients.

When a user requests a new video session, e.g., at time t, the system first checks the multicast channels for the next upcoming multicast of the requested video. Let t_m be the time for the next upcoming multicast, then the system will assign the user to wait for the upcoming multicast (henceforth referred as *Admit-via-Multicast*) if the waiting time is smaller than a predetermined admission threshold δ

$$(t_m - t) \leq \delta. \tag{2}$$

Otherwise, the system will assign the user to wait for a free unicast channel to start playback (henceforth referred as *Admit-via-Unicast*). The admission threshold is introduced to reduce the load of the unicast channels and to maintain uniform latency between Admit-via-Multicast and Admit-via-Unicast users (see Section IV-B).

For Admit-via-Multicast users, the operation is essentially the same as in an NVoD system. The client just joins the upcoming multicast channel at time t_m and then continues receiving video stream data from that multicast channel.

For Admit-via-Unicast users, the client first starts caching video data from the previous multicast of the requested movie. Then it waits for a free unicast channel to start playback. For example, assume that the request arrives at time t and let t_{m-1} and t_m be the nearest epoch times of multicast channel $m-1$ and channel m, respectively, for which $t_{m-1} < t < (t_m - \delta)$. Then at time t, the client starts caching video data from channel $m-1$ into the client's local storage, as shown in Fig. 3(a). At the same time, the client enters the request queue and starts video playback using unicast once a free unicast channel becomes available.

The admission process is not yet completed as the client still occupies one unicast channel. As the client concurrently caches multicasted video data for the movie starting from movie time [2] $(t - t_{m-1})$, the unicast channel can be released after a time $(t - t_{m-1})$ and the client can continue video playback using the local cache, as shown in Fig. 3(b). Hence, similar to [7], the local cache is used to add time delay to the multicasted video stream so that it can be synchronized with the client playback.

[2]Movie time is the time offset relative to the beginning of the movie.

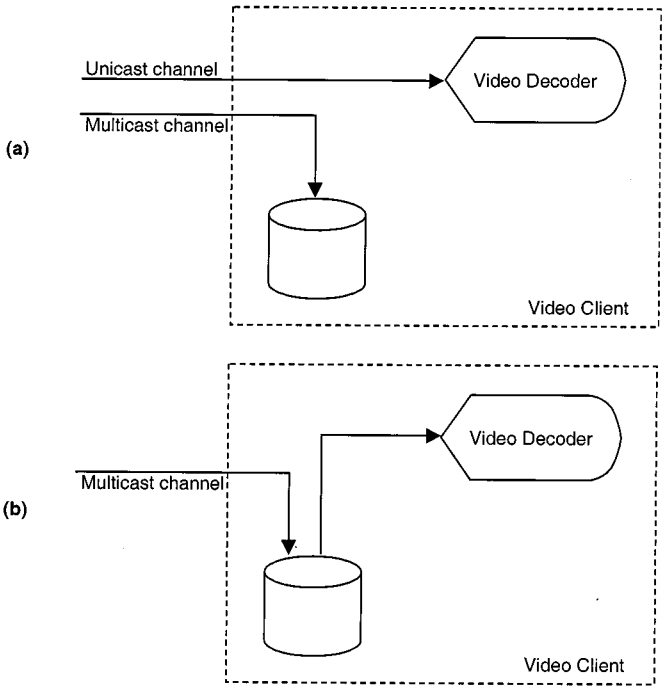

Fig. 3. (a) Simultaneous caching and playback during the start-up phase. (b) Playback via cache after the start-up phase.

This UVoD architecture achieves resource reduction over TVoD in two ways. First, a portion of the users will be admitted using multicast channels. As the number of multicast channels is fixed regardless of how many users are being served, these Admit-via-Multicast users will not result in additional load to the system. To further improve performance, one can increase the admission threshold δ and more users will be admitted to the multicast channels instead of the unicast channels, at the expense of larger latency. Second, for Admit-via-Unicast users, since $0 < (t - t_{m-1}) < (T_R - \delta) \ll L$, we can see that the unicast channels are occupied for a much shorter duration compared to TVoD. For example, with a movie-to-channel ratio of 0.1, the channel-holding time for Admit-via-Unicast users is no longer than 24 min, compared to 120 min for TVoD. This reduction in the channel-holding time substantially reduces the load at the unicast channels and allows far more requests to be served using the same number of channels.

IV. Performance Modeling

This section presents a performance model for the proposed UVoD system. Some existing works [3], [4] use user turn-away probability as a metric for performance evaluation. The motivation being that users are impatient and some of them would leave the system if the waiting time is too long. While the turn-away probability is an important parameter for the service provider, the user behavior model is generally unknown and hence renders accurate evaluation of the system difficult. Therefore in this paper, we employ another common performance metric for analysis—latency (or average waiting time), defined as the average time from a request arriving at the system to the time when the beginning of the video stream is transmitted.

TABLE I
LIST OF SYSTEM PARAMETERS

Parameters	Symbol	Value
Total number of available channels	N	100, 500
Number of multicast channels	N_M	N/A
Number of unicast channels	N_U	N/A
Number of movies	M	N/A
Length of each movie	L	120 minutes
Skewness for Zipf-distributed movie popularity profile	θ	0.271

Specifically, for Admit-via-Multicast users, the latency is the average waiting time for the next upcoming multicast of the requested movie. For Admit-via-Unicast users, the latency is the average time spent waiting at the unicast queue. Note that the actual waiting time experienced by the user is likely to be longer due to network delay, prefetch buffering, etc. However, these minor complexities will be ignored in this paper as they equally apply to UVoD, TVoD, and NVoD.

We first derive a model for the average waiting time in the next two sections and then determine the selection of the admission threshold in Section IV-B and the partition of the available channels between unicast and multicast in Section IV-C.

A. Waiting Times

For an Admit-via-Multicast user, the waiting time can range from 0 to δ s. Assuming requests are equally probable to arrive at any time, then the average waiting time, denoted by $w_M(\delta)$, is equal to half of the admission threshold

$$w_M(\delta) = \frac{\delta}{2}. \tag{3}$$

On the other hand, for an Admit-via-Unicast user, the waiting time is equal to the waiting time at the queue if all unicast channels are occupied; or zero otherwise. Clearly, the waiting time depends on the arrival process, the service time distribution, as well as the load of the unicast channels. For the arrival process, we can assume that video requests form a Poisson arrival process with rate λ. This is justified by the fact that users initiate requests independently from each other.

For a Poisson arrival process, the probability that an arriving request falls within the admission threshold is given by

$$P_m = \frac{\delta}{T_R} \tag{4}$$

where T_R is the repeating interval for the multicast channels. This P_m is also the probability that a user is admitted via multicast. Correspondingly, the probability for Admit-via-Unicast is $(1 - P_m)$. Assuming that this *splitting* process is probabilistic, then the resultant arrival process at the unicast channels is also Poisson, with a reduced rate equal to

$$\lambda_u = (1 - P_m)\,\lambda. \tag{5}$$

For the service time, it depends on the arrival time t and the time t_{m-1} for the previous multicast of the requested movie. Since $0 < (t - t_{m-1}) < (T_R - \delta)$, the service time s for requests entering the unicast-channel queue is uniformly distributed between

$$0 < s < T_R - \delta. \tag{6}$$

Therefore, the unicast channels form a multiserver queue with Poisson arrival and uniformly distributed service time. As no close-form solution exists for such a queueing model, we resort to the approximation by Allen and Cunneen [17] for G/G/m queues to obtain the average waiting time

$$w_U(\delta) \approx \frac{E_C\left(N_U, u\right)}{N_U(1-\rho)} \left(\frac{C_A^2 + C_S^2}{2}\right) T_S \tag{7}$$

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

$$C_S^2 = \frac{(T_R - \delta)^2}{12} \left(\frac{2}{(T_R - \delta)}\right)^2 = \frac{1}{3} \tag{8}$$

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

$$T_S = \frac{T_R - \delta}{2}. \tag{9}$$

Additionally, $u = \lambda_u T_S$ is the traffic intensity; $\rho = u/N_M$ is the server utilization; and $E_C(N_U, u)$ is the Erlang-C function, as given by

$$E_C(m, u) = \frac{\frac{u^m}{m!}}{\frac{u^m}{m!} + (1-\rho)\sum_{k=0}^{m-1} \frac{u^k}{k!}}. \tag{10}$$

B. 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 Admit-via-Unicast and Admit-via-Multicast users may differ. To maintain a uniform average waiting time in both cases, we can adjust the admission threshold according to the average waiting time at the unicast channels

$$\delta = \min\left\{x \middle| \left(w_M(x) - w_U(x)\right) \le \varepsilon, T_R \ge x \ge 0\right\} \tag{11}$$

so that the waiting-time differences are less than some small value ε.

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 waiting time, referred to as latency thereafter, for both Admit-via-Multicast and Admit-via-Unicast users.

C. Channel Partitioning

Another important parameter in the UVoD architecture is the proportion of channels allocated for multicast transmissions. Intuitively, too many multicast channels will leave too few chan-

nels for unicast, which may lead to overflow at the unicast channels. On the other hand, too few multicast channels will increase channel-holding time [cf. (6)] for requests entering the unicast channels, which again may lead to overflow. Hence, careful partitioning of available channels between unicast and multicast is crucial for achieving optimal system performance.

In terms of channel partitioning, the two extremes are 0 and N multicast channels, which represents the special cases of TVoD and NVoD, respectively. The problem of channel partitioning then becomes one of finding the optimum number of multicast channels between zero and N, such that the resultant latency is minimized. This translates into minimizing $w_U(\delta)$ in (7) with respect to N_U.

Unfortunately, minimizing (7) does not appear to be tractable analytically. Therefore, one would have to use numerical methods, which may take a long time if N is large. To tackle this problem, we take advantage of the observation that minimizing the load at the unicast channels will reduce the average waiting time $w_U(\delta)$. Therefore, instead of minimizing (7) directly, we could use the load at the unicast channels for optimization to obtain the optimal partition policy, presented in Theorem 1.

Theorem 1: Assuming each movie is allocated with at least one multicast channel and the admission threshold δ is smaller than the multicast cycle T_R, then the optimal proportion of available channels assigned to multicast that minimizes the load at the unicast channels is given by

$$N_M^{opt} = \left\langle \frac{LN}{2LM - \delta N} \right\rangle \frac{M}{N} \tag{12}$$

where the $\langle\rangle$ operator rounds the input to the nearest integer.

Proof: Please refer to the Appendix. ■

To integrate this channel-partition policy into the previous derivations, we can simply replace the variable N_U in (7) by $(N - N_M^{opt})$ and then use (11) to obtain the optimal admission threshold (and in turn, the latency) accordingly. We will discuss the effect of channel partitioning using numerical results in Section V-B.

V. Numerical Results

In this section, we evaluate the performance of the proposed UVoD architecture and contrast it against TVoD and NVoD using numerical results. As the primary performance metric is latency, we first derive the corresponding latency formula for NVoD and TVoD. For NVoD, the latency is simply equal to half of the repeating interval

$$W_{NVoD} = \frac{L}{2\left\lfloor \frac{N}{M} \right\rfloor}. \tag{13}$$

As the latency is constant given L, N, and M, NVoD can in theory support an unlimited number of users. By contrast, the latency for TVoD depends on the traffic intensity. Similar to Section IV, we could model TVoD as a G/G/m queue with $m = N$ servers. The arrival process is Poisson, but the service time would become constant if we ignore interactive control and assume constant movie length. Hence, applying the Allen-Cuneen approximation again, we can obtain the latency for TVoD from

$$W_{TVoD} \approx \frac{E_C(N,u)}{N(1-\rho)} \cdot \frac{L}{2} \tag{14}$$

with $C_A^2 = 1$ for Poisson process, $C_S^2 = 0$ and $T_S = L$ for constant service time.

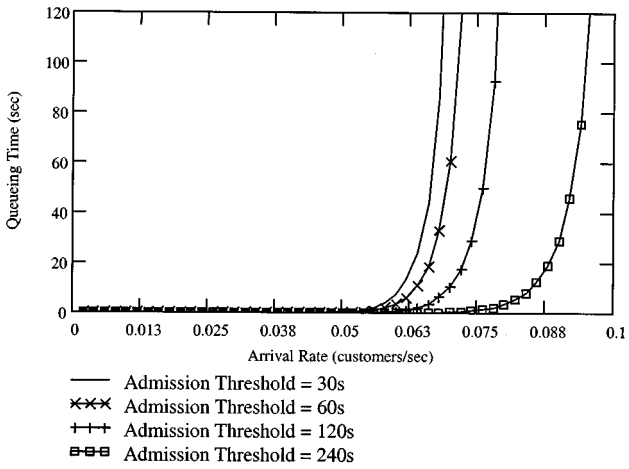

Fig. 4. UVoD queueing time versus arrival rate (100 channels, ten movies).

The numerical results presented in the following subsections are calculated using the performance models derived in Section IV and the parameters in Table I.

A. Effect of Admission Threshold on Queueing Delay

Fig. 4 plots the waiting time at the unicast channels versus arrival rate under four different admission threshold settings (30, 60, 120, and 240 s) in a 100-channel system. The system exhibits typical queueing-system characteristics (i.e., delay rises rapidly at near-saturation point). As the admission threshold is increased, the delay is reduced correspondingly. This is because the proportion of requests routed to the unicast channels is inversely proportional to the admission threshold value [cf. (4) and (5)] and increasing the threshold value also decreases the average service time for requests routed to the unicast channels [cf. (9)]. Similar results are observed for a 500-channel system.

B. Effect of Channel Partition on Latency

Fig. 5 plots the latency versus the proportion of channels assigned for multicast for three arrival rates ($\lambda = 0.055, 0.060,$ and 0.065 customers/s) in a 100-channel system with ten movies. In all three cases, we can clearly observe that an allocation of 0.5 achieves the lowest latency. This observation matches the prediction in Theorem 1.

Fig. 6 further investigates the partition policy with respect to the arrival rate. The figure shows how the partition policy assigns more channels for multicast as the arrival rate (i.e., load) increases. Note that the system ultimately degenerates into an NVoD system with all channels assigned to multicast to cope with the heavy load.

C. Latency Comparison With TVoD and NVoD

To contrast the performance with TVoD and NVoD, we plot the latency versus arrival rate for UVoD and TVoD for a 100-

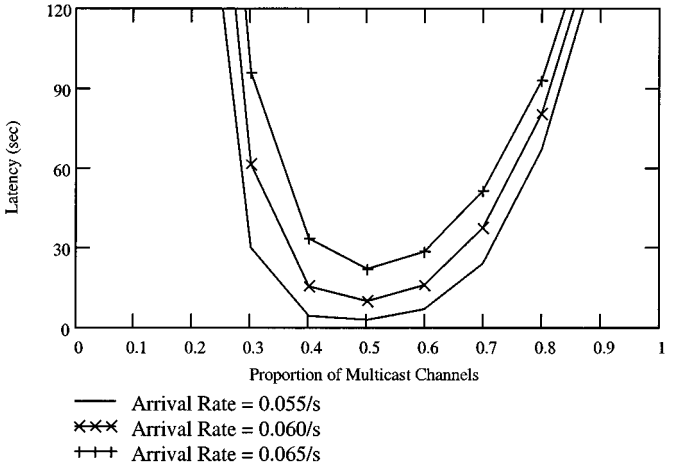

Fig. 5. Latency versus proportion of multicast channels (100 channels, 10 movies).

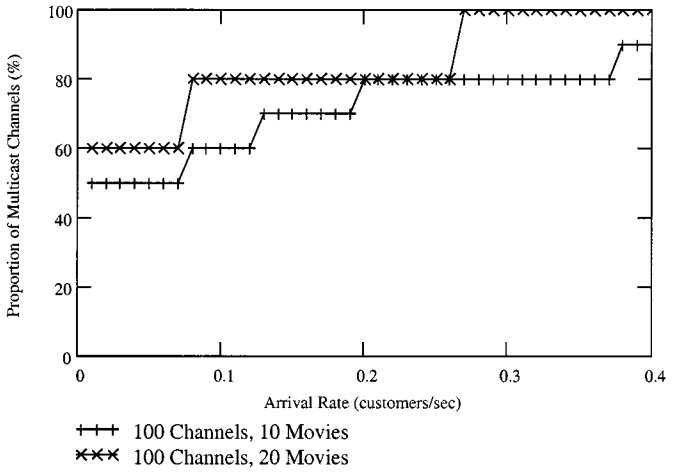

Fig. 6. Near-optimal channel partition with respect to arrival rate.

channel system in Fig. 7(a) for arrival rates up to 0.07 customers/s. Clearly, UVoD outperforms TVoD by a very wide margin. The performance improvement narrows when we increase the number of movies to 20. This is expected as fewer multicast channels are available per movie ($100 \times 0.5/25 = 2$ multicast channels per movie only). NVoD is not plotted, as the latency is constant at 360 s. Another observation is that there is no difference between using fixed channel partition of 50% and using the near-optimal channel partition policy. This is because the latency under the given load is relatively small (< 60 s) and as Fig. 6 shows, the near-optimal assignment is simply equal to 50%.

We plot in Fig. 7(b) a similar graph for heavier loads up to 0.2 customers/s. The primary observation is that for the curve with near-optimal channel partition, the latency levels off for very heavy loads. In particular, the latency approaches 360 s for the 10-movie configuration and 900 s for the 25-movie configuration. These are precisely the latencies for NVoD, indicating that UVoD gradually transforms into NVoD under heavy loads. Note that for the fixed 50% allocation policy, the latencies can exceed the NVoD bound under heavy load as not all channels are allocated for multicast. Hence, the channel-partition policy is essential in supporting a continuous cost-performance tradeoff from the TVoD extreme to the NVoD extreme.

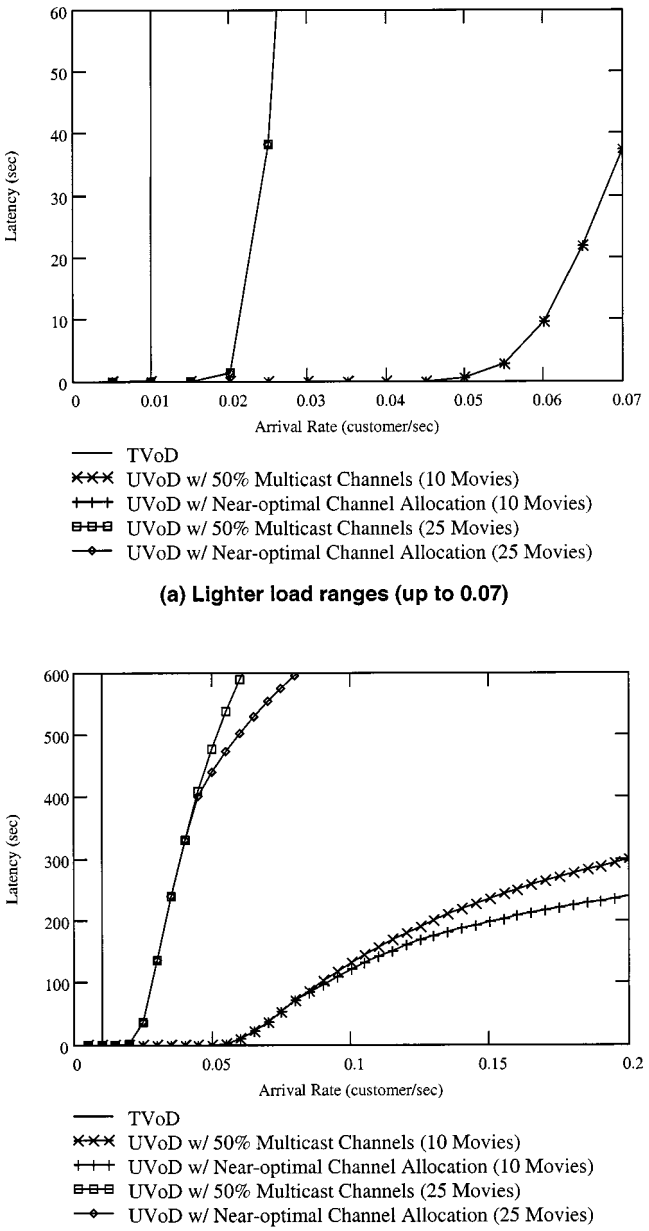

Fig. 7. Latency comparison of UVoD and TVoD.

D. System Capacity and Scalability

In system dimensioning, one would want to set a constraint for the latency and then determine the maximum arrival rate that can be supported by the system. In Fig. 8, we plot the capacity relative to TVoD versus given latency constraints. The relative capacity in the y-axis, denoted by G, is calculated from

$$G = \frac{\max\{\lambda | W_{UVoD} \leq \mu, \forall \lambda \geq 0\}}{\max\{\lambda | W_{TVoD} \leq \mu, \forall \lambda \geq 0\}} \times 100\% \qquad (15)$$

where

λ	arrival rate in customers/s;
μ	latency constraint in seconds;
W_{UVoD}	latency for UVoD;
W_{TVoD}	latency for TVoD.

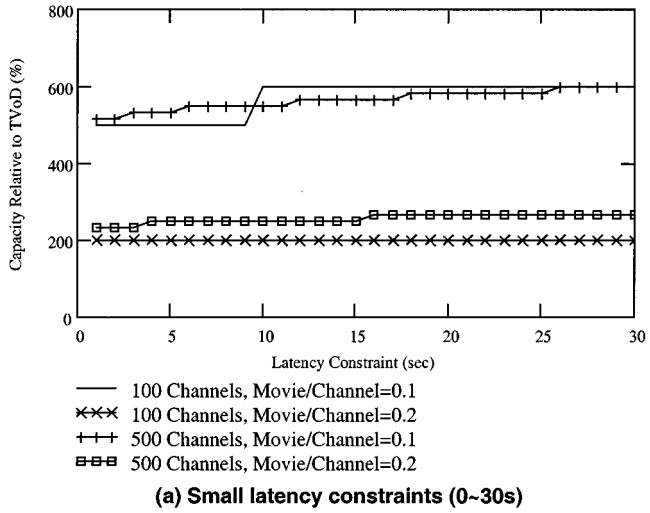

Fig. 8. Performance gain over TVoD versus latency constraint.

The results in Fig. 8 show that the performance gain is relatively constant for small latency constraints (0∼30 s) and we can achieve very good performance gain even at very small latencies. For example, with a latency constraint of only 1 s, UVoD already achieves a capacity gain of 500% for a 100-channel, 10-movie configuration and 516% for a 500-channel, 50-movie configuration. Larger movie-to-channel ratio will reduce the performance gain, e.g., doubling the number of movies would reduce the relative capacity to 200% and 233%, respectively, for the previous two configurations.

Fig. 8(b) is a similar plot albeit for a wider range of latency constraints. The primary observation is that the relative capacity increases exponentially for latencies near the NVoD bound and reaches infinity at the NVoD bound. This is because the channel-partition policy incrementally assigns more channels for multicast until the system degenerates into an NVoD system.

Clearly, if the number of movies in the system becomes comparable to the number of channels, performance gain will diminish. In the extreme case where there are more movies than channels, the UVoD architecture will not be directly applicable. On the other hand, large movie-to-channel ratio will also result in large buffering requirement at the client. For example, if the movie-to-channel ratio is increased to 0.25 (e.g., 100 channels, 25 movies), then the client buffer requirement will increase to 60 min, or 1.8 GB for 4-Mb/s MPEG2 video streams. This means a hard disk will be required at the client for buffering purpose.

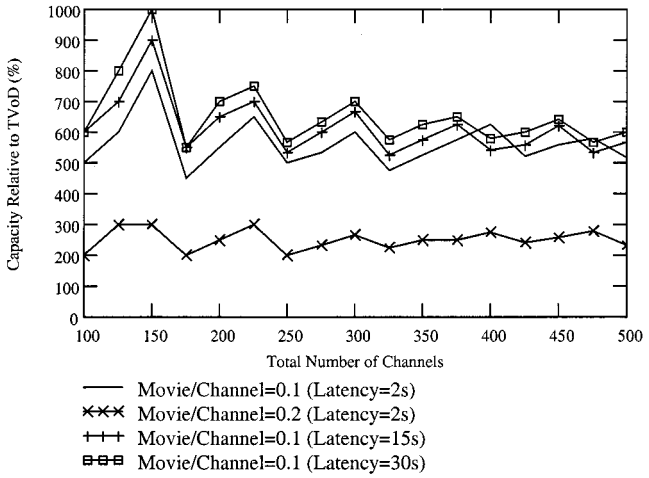

Fig. 9. Performance gain over TVoD versus system scale.

Fortunately, most real-world services share one characteristic: a small portion of the movies accounts for most of the user traffic. Therefore, one can implement a two-level architecture in which the few popular movies are served by UVoD while the less popular movies are served by TVoD. This will enable one to maintain a low movie-to-channel ratio for the UVoD servers to keep performance gains high and client buffer requirement low. The problem of optimal partitioning between these two levels is beyond the scope of this paper and will be left for future work.

Fig. 9 shows another evaluation of the performance gain with respect to different system scale (i.e., number of channels). The gain fluctuates across the scale due to divisibility between the number of multicast channels and the number of movies. In broader ranges, the performance gain is relatively constant. This suggests that UVoD is not limited to large-scale systems and can be applied to systems of all scale.

VI. Simulation Results

Results in the previous section are based on the performance model developed in Section IV. In this section, we present simulation results to validate the numerical results and propose a rescheduling algorithm to further improve the performance gain. The simulation program is developed in C++ using CNCL version 1.10 (ported to the Windows platform). Each simulation run simulates a duration of 744 h (31 days) with statistics for the first day skipped to reduce initial-condition effects.

A. Model Validation

To assess the accuracy of the performance model derived in Section IV, we simulated a 100-channel configuration and obtained the average waiting time over a range of arrival rates. We compare the analytical results with simulation results under

three admission threshold settings in Fig. 10 and under three channel partition settings in Fig. 11. It is clear that the simulation results match the performance model closely, thereby validating the derived analytical model. Note that in generating the results, we have used uniform movie-popularity profile as well as Zipf-distributed movie-popularity profile. Both profiles give the same results, verifying that the architecture is independent of the movie popularity profile.

B. Admission Rescheduling

Under the admission process described in Sections III and IV, once a user is routed to the unicast channels (i.e., Admit-via-Unicast), it will wait until a free unicast channel becomes available. For heavy system loads, it is possible that the waiting time incurred could exceed the time to the next multicast of the requested movie. In this case, the user would be better off quitting the unicast queue to start playback using the multicast channel instead—we call this technique *admission rescheduling*. In addition to reducing the waiting time of the rescheduled user, admission rescheduling can also reduce the load at the unicast channels because some users will be removed from the queue without being served.

To evaluate the effect of this admission rescheduling technique, we performed simulations using a modified admission scheduler with admission rescheduling. We also added a simple adaptive algorithm to automatically configure the admission threshold on-the-fly so that a uniform latency is maintained for both Admit-via-Unicast and Admit-via-Multicast requests. As each simulation run is executed with a constant arrival rate, the adaptation algorithm simply adjusts the admission threshold periodically according to the average waiting time of past Admit-via-Unicast requests. The problem of designing good adaptation algorithms that can cope with variable arrival rates (e.g., due to time-of-day changes) is beyond the scope of this paper and is left for future work.

We summarized the results in Fig. 12, plotting latency versus arrival rate. The results clearly show that the admission rescheduling technique can indeed reduce latency over a wide range of arrival rates. Moreover, as the multicast schedule is known *a priori*, the worst-case waiting time for an incoming request can then be determined as well. This can improve the system's user-friendliness, as the user does not need to wait endlessly without knowing when a movie will start playing.

VII. Interactive Controls

The system model in Section IV assumes that each user watches a movie from start to finish and hence does not account for interactive control requests. To provide a service comparable to TVoD, interactive viewing controls must also be supported in UVoD. Possible interactive controls include pause-resume, seeking, visual search (forward and backward), frame stepping (forward and backward), and slow motion. Among these controls, pause-resume is probably the most important (e.g., pausing playback to answer a telephone call, doorbell, etc.), especially in movie-on-demand applications. The two subsections below present two different approaches

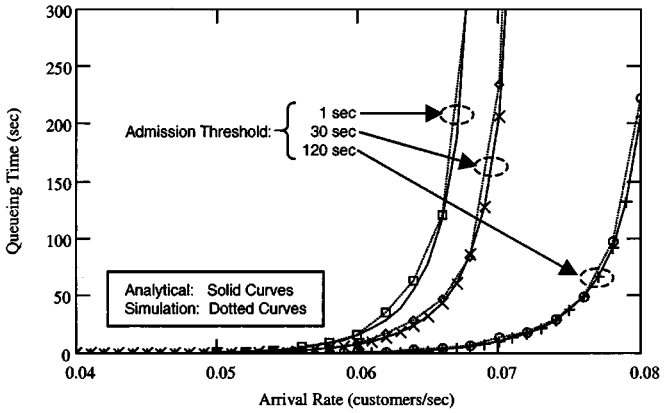

Fig. 10. Simulated versus analytical results for three admission threshold settings.

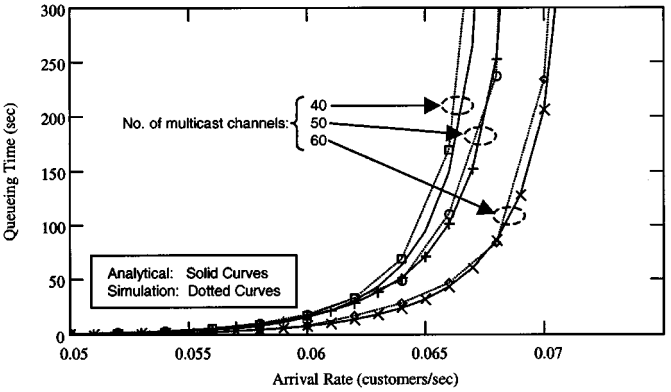

Fig. 11. Simulated versus analytical results for three channel-partition settings.

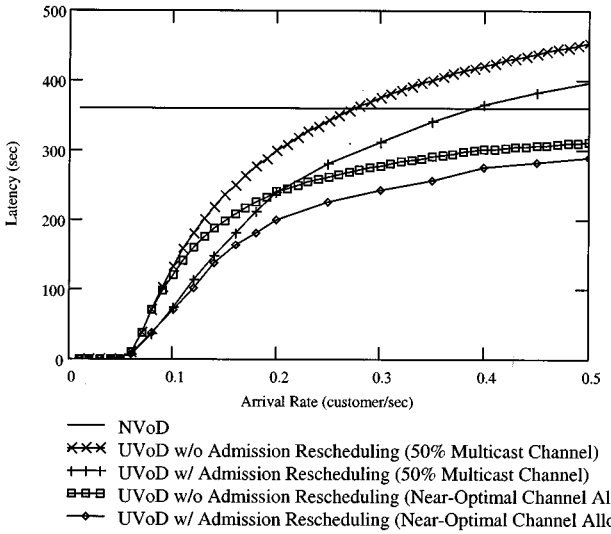

Fig. 12. Performance of UVoD under heavy loads (100 channels, ten movies).

to support interactive control under UVoD, each achieving a different cost-performance tradeoff.

A. Using Unicast Channels

Intuitively, performing an interactive control essentially breaks the client away from the current multicast video stream

and then restarts it at some point within the video stream. Under this view, interactive control is no different from a new request and hence can be served the same way as for a new-video request. Hence, a straightforward way to support interactive control requests is to treat them as new-video requests, albeit starting at the middle of a movie. In this way, interactive controls such as pause-resume, seeking, can be supported. Visual search can be supported by encoding a special version of the movie at higher playback speed (i.e., skipping frames) and then delivered through a unicast channel to the client during the search. Obviously, this approach will increase loads at the unicast channels, which could increase waiting time for both new and interactive requests. As there is no generally accepted user-activity model, we do not attempt to quantify the performance impact of this approach and refer the interested readers to [5]–[7] and [9].

B. Channel Hopping

Due to the static channel allocation employed in UVoD, we can devise a channel-hopping algorithm to support pause-resume control without incurring additional load at the unicast channels. Specifically, each movie is multicasted every T_R s and the client has a buffer large enough to cache T_R s of video. When a user pauses, say, at a movie time t_p, the client just continues to buffer the incoming video data. If the user resumes playback before buffer overflows, then nothing needs to be done. Otherwise, the client just stops buffering and enters an idle state once the buffer is full [i.e., storing the movie segment from t_p to $(t_p + T_R)$]. When the user later resumes playback, the client can resume playback immediately and at the same time determine the nearest multicast channel that is currently multicasting the movie at movie time $t_m \geq t_p$. Since a movie is repeatedly multicasted every T_R s, we have $(t_m - t_p) \leq T_R$. Hence, the client just needs to start buffering again after the selected channel reaches movie time $(t_p + T_R)$.

This channel-hopping algorithm is unique in the sense that no additional resource is required at the server. Pause-resume is simply supported by buffering and switching of multicast channel at the appropriate time. Hence, UVoD is particularly suitable for movie-on-demand applications where pause-resume is the primary interactive control needed.

VIII. Conclusions

This paper proposes and analyzes an architecture that unifies the existing TVoD and NVoD architectures. Through dynamic admission-threshold adaptation and channel partitioning, one can achieve continuous cost-performance tradeoffs between the TVoD extreme and the NVoD extreme. The proposed UVoD architecture not only encompasses TVoD and NVoD as special cases, but also achieves significant performance gains with little tradeoff. In particular, results show that performance gain as large as 500% can be achieved at a latency of only 1 s.

The proposed UVoD architecture is particularly suitable for movie-on-demand applications. First, a service provider can deploy a large-scale VoD system incrementally and make conservative assumptions during system dimensioning. This is possible because UVoD can be configured to provide service quality similar to TVoD, while still be able to gracefully cope with any amount of additional loads (with tradeoffs in latency) in case the demand exceeds the anticipated rate. Second, the most essential interactive control, pause-resume, can be supported in UVoD without any additional overhead. This not only reduces system resource requirements, but also allows for a simpler pricing policy (e.g., no additional cost for pause-resume and charges only other forms of interactions) for the service provider.

Finally, while UVoD does require additional buffering capability at the client set-top boxes, the extra buffer can be implemented using a low-cost local hard disk. It is highly likely that future set-top boxes will become full-feature multimedia entertainment devices that can provide not only VoD, but also web browsing, gaming, etc. Hence, the additional storage will be needed anyway (e.g., for caching web pages and downloading games) and the cost can be amortized over multiple applications.

Appendix

Proof of Theorem 1: We assume that the movies have an arbitrary popularity profile given by $\{g_i, 1 \leq i \leq M\}$ where g_i is the probability that a new user requests movie i. Without loss of generality, we assume that the movie numbers are assigned according to popularity where $g_i \geq g_j \forall i < j$. Clearly, we must have

$$\sum_{i=1}^{M} g_i = 1. \tag{16}$$

Hence, the traffic intensity due to movie i at the unicast channels is given by

$$u_i = \frac{\lambda g_i}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right) \tag{17}$$

where n is the number of multicast channels assigned for each movie; $(1 - (\delta n/L))$ is the proportion of requests routed to the unicast channels; and $(1/2)((L/n) - \delta)$ is the average service time. We can then obtain the load at the unicast channels, denoted by ρ, from

$$\rho = \frac{\sum_{i=1}^{M} u_i}{N - Mn}. \tag{18}$$

Substituting (17) into (18) gives

$$\begin{aligned}\rho &= \frac{\sum_{i=1}^{M} \frac{\lambda g_i}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn} \\ &= \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)\sum_{i=1}^{M} g_i}{N - Mn} \\ &= \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn}.\end{aligned} \tag{19}$$

To minimize ρ with respect to n, we can differentiate (19)

$$\frac{\partial \rho}{\partial n} = \left(\frac{N\delta^2}{L} - 2\delta M\right)n^2 + 2LMn - LN \tag{20}$$

and then solve for n by setting $\partial\rho/\partial n = 0$ to obtain

$$n = \frac{LN}{2LM - \delta N} \tag{21}$$

which minimizes ρ. We also need to round up n to the nearest integer, as a video channel cannot be divided between multiple movies. This will be the optimal number of multicast channels for one movie. Note that n is nonnegative as long as $\delta < T_R$. To see why, we can assume that $n < 0$ and rearrange (21) as

$$\delta = \frac{\alpha LN + 2LM}{N} \quad \text{where } \alpha \geq 0. \tag{22}$$

Now, since $\delta < T_R$, we can express T_R in terms of n

$$\delta < T_R = \frac{L}{n} = \frac{2LM - \delta N}{N}. \tag{23}$$

Rearranging, we can then obtain δ from

$$\delta < \frac{2LM}{N} \tag{24}$$

which contradicts with (22). Hence, n must be nonnegative.

Since we have M movies and the allocation is uniform, the optimal proportion of channels for multicast is simply given by

$$\frac{nM}{N} = \left\langle \frac{LN}{2LM - \delta N} \right\rangle \frac{M}{N} \tag{25}$$

and the result follows. ■

ACKNOWLEDGMENT

The author would like to thank the anonymous reviewers for their insightful comments and suggestions in improving this paper.

REFERENCES

[1] S. L. Tsao and Y. M. Huang, "An efficient storage server in near video-on-demand systems," *IEEE Trans. Consumer Electron.*, vol. 44, no. 1, pp. 27–32, 1998.

[2] T. C. Chiueh and C. H. Lu, "A periodic broadcasting approach to video-on-demand service," in *Proc. SPIE*, vol. 2615, 1996, pp. 162–169.

[3] 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.

[4] H. Shachnai and P. S. Yu, "Exploring waiting tolerance in effective batching for video-on-demand scheduling," in *Proc. 8th Israeli Conf. Computer Systems and Software Engineering*, June 1997, pp. 67–76.

[5] V. O. K. Li, W. Liao, X. Qiu, and E. W. M. Wong, "Performance model of interactive video-on-demand systems," *IEEE J. Select. Areas Commun.*, vol. 14, no. 6, pp. 1099–1109, 1996.

[6] 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, no. 6, pp. 1110–1122, 1996.

[7] 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, 1997.

[8] H. K. Park and H. B. Ryou, "Multicast delivery for interactive video-on-demand service," in *Proc. 12th Int. Conf. Information Networking*, Jan. 1998, pp. 46–50.

[9] E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, July 1998, pp. 66–75.

[10] 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.

[11] 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.

[12] 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. 118–126.

[13] 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.

[14] 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. 253–258.

[15] H. Woo and C. K. Kim, "Multicast scheduling for VoD services," *Multimedia Tools Applicat.*, vol. 2, no. 2, pp. 157–171, 1996.

[16] H. Kalva and B. Furht, "Techniques for improving the capacity of video-on-demand systems," in *Proc. 29th Annu. Hawaii Int. Conf. Systems Sciences*, Jan. 1996, pp. 308–315.

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

Jack Y. B. Lee received the Ph.D. in information engineering from The Chinese University of Hong Kong (CUHK) , Shatin, N.T., Hong Kong, in 1997.

He was with the Department of Computer Science at the Hong Kong University of Science and Technology from 1998 to 1999. Currently, he is Assistant Professor at the Department of Information Engineering, CUHK, where he directs the Multimedia Communications Laboratory (http://www.mcl.ie.cuhk.edu.hk) and conducts research in distributed multimedia systems, fault-tolerant systems, multicast communications, and Internet computing.