Optimizing Channel Allocation in a Unified Video-on-Demand System

Jack Y. B. Lee

***Abstract*—Unified video-on-demand (UVoD) is a recently proposed architecture that integrates multicast transmission with unicast transmission to improve system efficiency. Streaming channels in a UVoD system are divided into unicast and multicast channels, with the multicast channels further divided equally among all videos. This uniform channel-allocation scheme is simple to design and implement, but the performance may not be optimal due to differences in video popularity. This paper investigates this channel-allocation problem with the goal of optimizing system efficiency. First, the uniform allocation assumption is removed and the channel-allocation problem formulated as a nonlinear integer optimization problem. This optimization model results in nonuniform channel allocations that can save up to 10% of channels. Second, to reduce the computational complexity in solving the nonlinear optimization model, an approximate model is derived and solved under small-latency conditions to obtain a closed-form solution. Third, a much simpler class-based popularity model is proposed and shown to achieve good efficiency, even if the precise popularity of each video is not known. Lastly, a zero-multicast channel-optimization algorithm is introduced that can further reduce channel requirement for systems with a large number of video selections. Numerical results show that optimized nonuniform channel-allocation policies can achieve channel reduction over uniform channel allocation by as much as 50% for a 1000-video system.**

***Index Terms*—Channel allocation, NVoD, performance analysis, TVoD, unified architecture, UVoD, 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 model of true-video-on-demand (TVoD) 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 of subscribers, the required infrastructure investment would be immense.

To tackle this problem, a number of researchers have started to investigate various innovative architectures in an attempt to improve the scalability and efficiency of large-scale VoD systems [1]–[13]. Examples include the periodic broadcasting approach by Chiueh *et al.* [1], the batching approach by Dan *et al.* [2] and Shachnai *et al.* [3], the split and merge protocol by Liao *et al.* [4], the stream tapping scheme by Carter *et al.* [5], the pyramid broadcasting approach by Viswanathan *et al.* [6] and Aggarwal *et al.* [7], the piggybacking approach by Golubchik *et al.* [8] and Aggarwal *et al.* [9], and so on. It is beyond the scope of this study to compare these difference approaches and the interested readers are referred to [5], [13] for some comparative discussions.

This study focuses on one of these approaches: the unified video-on-demand (UVoD) architecture [13], which combines the efficiency of near-video-on-demand (NVoD) with the short latency of TVoD by integrating multicast with unicast transmissions.

Briefly speaking, UVoD divides available channels into unicast and multicast channels. The multicast channels are then allocated equally to all videos. Each video is multicast repeatedly over the allocated multicast channels similar to a NVoD system. In NVoD, the startup latency is substantially longer than TVoD because an arriving user must wait until the next multicast cycle starts. UVoD solves this problem by allocating a transitory unicast channel to the user to start playback immediately while the client concurrently caches video data from a multicast channel. When the unicast stream catches up with the start of the cached multicast stream, the client can then be switched back to playback video data through the cache and releases the unicast channel.

The study by Lee [13] employs a uniform channel-allocation policy to divide multicast channels equally among all videos. This policy simplifies system design and implementation but can be suboptimal. Specifically, video popularity is highly skewed in practice [14], i.e., a small fraction of videos account for a large proportion of the traffic. Hence, allocating the same number of multicast channels to both popular and unpopular videos is intuitively suboptimal. For example, if a video is so unpopular that no one ever requests it, then the allocated multicast channels will be wasted.

In this study, we investigate this channel-allocation problem with the goal of optimizing system efficiency. The contributions of this study are as follows. First, we remove the uniform allocation assumption in Lee [13] and show that the channel-allocation problem can be formulated as a nonlinear integer optimization problem. This optimization model results in nonuniform channel allocations that can save up to 10% channels. Second, to reduce the computational complexity in solving the optimization model, we derive and solve an approximation model for small-latency conditions to obtain a close-form solution. Third, using a class-based popularity model, we show that good efficiency can still be obtained even if the precise popularity of each video is not known. Last but not least, we introduce a zero-multicast-channel optimization algorithm to further reduce channel requirement for systems with a large number of video selections. With these nonuniform channel-allocation techniques, the modified UVoD architecture investigated in this study can achieve

Manuscript received September 8, 2000; revised April 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. Watanabe.

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).

Digital Object Identifier 10.1109/TCSVT.2002.804890

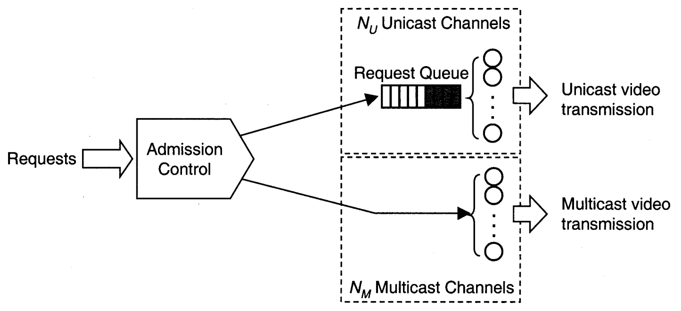

Fig. 1. Architecture of the UVoD system.

up to 50% resource reduction compared to the original UVoD architecture in [13].

The rest of paper is organized as follows. Section II presents an overview of the UVoD architecture; Section III presents the formulation of the channel-allocation problem as a nonlinear integer optimization problem; Section IV presents the small-latency approximation; Section V presents the class-based popularity model; Section VI presents the zero-multicast-channel optimization; Section VII evaluates and compares various channel-allocation policies using numerical results; and Section VIII concludes the paper.

II. UVoD Architecture

In this section, we review the UVoD architecture and present its basic properties. The UVoD architecture as proposed by Lee [13] is depicted in Fig. 1. There are a total of N available channels, of which N_U of them are unicast channels and $N_M = N - N_U$ of them are multicast channels. A channel is defined as the unit for resource allocation and includes network bandwidth as well as server bandwidth. Let there be M videos of length L seconds each. Under the uniform channel-allocation policy, the N_M multicast channels will be divided equally among those M videos so that each video is multicast over N_M/M multicast channels, assuming N_M is divisible by M. For each multicast channel, the assigned video is multicast repeatedly. Multicast channels streaming the same video are offset by (in seconds)

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

as in a NVoD system.

The N_U unicast channels share a common 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. Finally, the video clients are capable of receiving two video channels simultaneously and have local storage to cache up to T_R seconds of video data.

A. Admission Control

When a user requests a new video session, say 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. 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

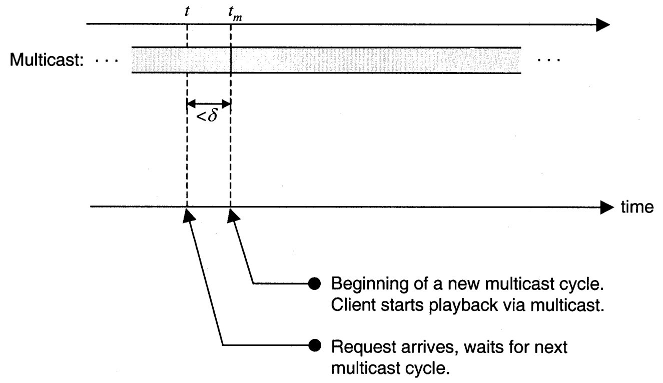
t
t_m
Multicast: · · ·
· · ·
$<\delta$
time
Beginning of a new multicast cycle. Client starts playback via multicast.
Request arrives, waits for next multicast cycle.

Fig. 2. Admission procedure for an admit-via-multicast client.

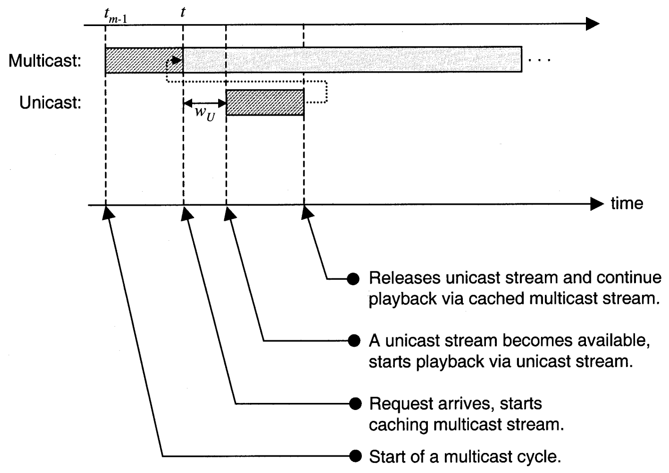
t_{m-1}
t
Multicast:
· · ·
Unicast:
w_U
time
Releases unicast stream and continue playback via cached multicast stream.
A unicast stream becomes available, starts playback via unicast stream.
Request arrives, starts caching multicast stream.
Start of a multicast cycle.

Fig. 3. Admission procedure for an admit-via-unicast client.

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

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 a uniform latency experienced by both admit-via-multicast and admit-via-unicast users.

For admit-via-multicast users, the operation is essentially the same as in a 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, as shown in Fig. 2.

For admit-via-unicast users, the client first starts caching video data from the previous multicast of the requested video, as shown in Fig. 3. 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, 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. 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 complete as the client still occupies one unicast channel. Since the client concurrently caches multicasted video data for the video starting from video time[1] $(t - t_{m-1})$, the unicast channel can be released after a time $(t - t_{m-1})$ and the client can continue video playback using the local cache. Since $0 < (t - t_{m-1}) < (T_R - \delta) \ll L$, we can see that the unicast channels are occupied for much shorter duration when compared to TVoD. This reduction in service time allows more requests to be served by the unicast channels.

B. Optimizing Uniform Channel Allocation

The uniform channel-allocation policy has one controllable parameter, namely N_M—the number of channels allocated for multicast. Allocating too few multicast channels, and the unicast channels will become overloaded due to the long service time T_R [cf. (1)]. Allocating too many multicast channels, and there will be too few unicast channels left for serving admit-via-unicast users. Therefore one needs to find a balancing point so that the system performance (i.e., latency) is optimized.

Lee [13] suggested that since the latency depends on the load at the unicast channels, one can minimize latency by simply minimizing the load at the unicast channels. Specifically, the probability for an incoming user to be admitted via a unicast channel is given by

$$P_u = 1 - \frac{\delta}{T_R}. \tag{3}$$

Given an arrival rate of λ users per second, users will arrive at the unicast channels with a reduced rate equal to

$$\lambda_u = \lambda P_u. \tag{4}$$

The service times of these users depend on the arrival time t and the time t_{m-1} for the previous multicast of the requested video. Since $0 < (t - t_{m-1}) < (T_R - \delta)$, the service time s for requests entering the unicast-channel queue is uniformly distributed between

$$0 < s < T_R - \delta. \tag{5}$$

Hence, the traffic intensity at the unicast channels can be computed from

$$u = \lambda P_u \frac{(T_R - \delta)}{2} \tag{6}$$

where $(1/2)\,((L/n) - \delta)$ is the average service time. Given there are N_M multicast channels, the load at the unicast channels, denoted by ρ_u, is then given by

$$\rho_u = \frac{u}{N - N_M}. \tag{7}$$

[1]Video time is the time offset relative to the beginning of the video.

By differentiating (7) with respect to N_M, it can be shown [13] that the optimal number of multicast channels that minimizes the unicast channel load is given by

$$N_M = \left\langle \frac{LN}{2LM - \delta N} \right\rangle M \tag{8}$$

where $\langle\cdot\rangle$ rounds the argument to the nearest integer.

III. Nonuniform Channel Allocation

In this section, we relax the uniform allocation assumption and investigate nonuniform channel-allocation policies that can further improve system efficiency. Specifically, given the request arrival rate λ, the desired mean waiting time w, we want to find the minimum number of channels required, and the corresponding allocation vector (defined below) for the video titles.

First, we assume that videos have an arbitrary popularity profile specified by $\{g_i | i = 1, 2, \ldots, M\}$ where g_i is the probability that a user requests video i. Without loss of generality, we can assume that the videos are numbered according to decreasing popularity, i.e., $g_i \geq g_j, \forall i < j$. Clearly, we must have

$$\sum_{i=1}^{M} g_i = 1. \tag{9}$$

Let n_0 be the number of unicast channels, and n_i, $i = 1, 2, \ldots, M$ be the number of multicast channels allocated to video i. Then the set $\{n_i | i = 0, 1, \ldots, M\}$ forms the channel-allocation vector. Consider video i, the corresponding traffic intensity going into the unicast channels is given by

$$u_i = \begin{cases} \dfrac{\lambda g_i}{2}\left(1 - \dfrac{\delta n_i}{L}\right)\left(\dfrac{L}{n_i} - \delta\right), & n_i \geq 1 \\ \lambda g_i L, & n_i = 0 \end{cases} \tag{10}$$

where $(1 - (\delta n_i / L))$ is the proportion of requests routed to the unicast channels, and $(1/2)\,((L/n_i) - \delta)$ is the average service time. We can obtain the utilization of the unicast channels, denoted by ρ, from

$$\rho = \frac{1}{n_0} \sum_{i=1}^{M} u_i. \tag{11}$$

Since lower utilization results in shorter queuing delay at the unicast channels, our goal is to find a channel-allocation vector such that ρ is minimized. This can be formulated as

$$\begin{aligned} &\text{minimize } \rho = \frac{1}{n_0} \sum_{i=1}^{M} u_i \\ &\text{subject to } \begin{cases} \displaystyle\sum_{i=0}^{M} n_i \leq N \\ n_i \geq 0, \quad \forall i = 0, 1, \ldots, M \end{cases} \end{aligned} \tag{12}$$

which is a nonlinear integer optimization problem. This optimization model does not have simple closed-form solutions and, therefore, numerical methods are needed to obtain solutions.

The previous optimization model still does not provide a direct answer to our question posted at the beginning of the section. In particular, N and δ must be given in order to find the channel-allocation vector. Given a desired latency of w, it is easy to see that $\delta = 2w$, considering that the latency is half of the admission threshold for admit-via-multicast users. However, N is not known *a priori*, and so we still need to perform an additional step: iteratively find the minimum N that can meet the service specifications, namely arrival rate λ and latency w.

To find the latency given N, we model the unicast channels as a $G/G/m$ queuing system and apply the Allen–Cunneen approximation [15] to compute the average wait (i.e., latency)

$$w_U(\delta) \approx \frac{E_C(N_U, u)}{N_U(1-\rho)} \left(\frac{C_A^2 + C_S^2}{2}\right) T_S \tag{13}$$

where C_A and C_S are the coefficient of variation (CoV) for inter-arrival time and service time, respectively, T_S is the average service time, u is the traffic intensity, ρ is the server utilization as given in (11), and $E_C(N_U, u)$ is the Erlang-C function, as given by

$$E_C(m, u) = \frac{u^m/m!}{u^m/m! + (1-\rho)\sum_{k=0}^{m-1} \frac{u^k}{k!}}. \tag{14}$$

Now we need to derive the input parameters for (13). Given a channel-allocation vector, we can compute the traffic parameters for each of the video. Specifically, the service time for an admit-via-unicast user requesting video i is uniformly distributed between 0 and $((L/n_i) - 2w)$ for $n_i > 0$. For $n_i = 0$, the service time is simply the video length. Hence, the mean service time can be computed from

$$\bar{s}_i = \begin{cases} \frac{1}{2}\left(\frac{L}{n_i} - 2w\right), & n_i > 0 \\ L, & n_i = 0. \end{cases} \tag{15}$$

The arrival rate of admit-via-unicast users requesting video i is given by

$$\lambda_i = \lambda g_i \left(1 - \frac{2wn_i}{L}\right). \tag{16}$$

As there are M different videos, the combined traffic entering the unicast channels will have the following parameters:

$$\text{arrival rate:} \qquad \lambda_u = \sum_{i=1}^{M} \lambda_i \tag{17}$$

$$\text{mean service time:} \quad \bar{s}_u = \frac{1}{\lambda_u} \sum_{i=1}^{M} \lambda_i \bar{s}_i. \tag{18}$$

For simplicity, we assume that the combined traffic have a CoV of $C_A = 1$ (i.e., same as Poisson). The CoV for the service time can be computed from

$$C_S^2 = \frac{E[s^2] - (E[s])^2}{(E[s])^2} \tag{19}$$

where $E[s] = \bar{s}_u$ and

$$E[s^2] = \frac{1}{\lambda} \sum_{i=1}^{M} \lambda_i \eta_i \tag{20}$$

where

$$\eta_i = \begin{cases} \frac{\lambda_i n_i^2}{3}, & n_i > 0 \\ L, & n_i = 0. \end{cases} \tag{21}$$

Now all input parameters for the Allen–Cunneen formula is known, we can proceed to compute the minimum N required to satisfy the latency constraint from

$$N = \min\left\{ n \left| \frac{E_C(n_0, \lambda_u \bar{s}_u)}{n_0(1 - \lambda_u \bar{s}_u / n_0)} \left(\frac{1 + C_S^2}{2}\right) \bar{s}_u \le w \right. \right\} \tag{22}$$

using conventional numerical methods. Once N is known, the complete channel-allocation vector can then be computed from the optimization model in (12).

IV. Small-Latency Approximation

Using the previous optimization model, a system designer can perform system dimensioning and determine the channel-allocation vector simultaneously. However, this optimization approach is not without limitation. In particular, the optimization problem in (12) must be solved using numerical methods and the computational complexity is relatively high. Worst, each iteration in finding the minimum N in (22) requires solving (12), thereby further multiplying the computation time.

As an illustration, using the numerical solver in MathCAD Professional 2001[2] on a Compaq Professional Workstation AP550 with Dual Pentium III 728-MHz processors, it takes 13, 81, and 235 s to solve (12) with M equal to 50, 100, and 150, respectively. Moreover, the MathCAD solver failed to obtain solution for $M = 200$. We were able to overcome this by installing the optional Solving and Optimization Extension Pack[3] for MathCAD and obtained a solution for $M = 200$ in 660 s. Nonetheless, even the advanced solver failed to obtain solution for larger values of M tested (e.g., $M = 300, 400$, etc.).

While one can still obtain solutions for large M using other optimization tools or methods, the nature of the model (nonlinear with integer solutions) suggests that the results obtained may only be a local optimum rather than a global optimum. Moreover, the computational complexity and unpredictability of the result will limit the optimization process to be manually conducted offline.

[2]Information on MathCAD Professional 2001 can be found at http://www.mathcad.com.

[3]Information on the Solving and Optimization Extension Pack can be found at http://www.mathcad.com/addons/soe_pack_ben.asp.

To tackle this limitation, we present in this section an approximate model for (12), where the closed-form solution for the channel-allocation vector can be obtained. This approximate model can be used in place of the nonlinear model in (12) to reduce the computation time, or in combination with the nonlinear model in (12) to ensure that a solution is available and it will not be poorer than the approximated solution. Apart from these practical uses, the small-latency approximation also provides important insight into the performance model. For example, the approximate model reveals that the optimal allocation always reserves half of the channels for multicast and the other half for unicast, regardless of other system parameters.

Formally, the approximate model is based on the method of Lagrange multiplier [16] under three assumptions: 1) the integer solution can be obtained from a continuous approximation; 2) each video is allocated with at least one multicast channel; and 3) the latency under consideration is small. The last assumption is motivated by the observation that VoD services in practice require short response time in order to provide good quality of service, thereby making the small-latency approximation applicable.

To derive the approximation model, we first form an auxiliary function according to the method of Lagrange from the objective function and the constraint function

$$F(\{n_i\}, \alpha) = \rho + \alpha\left(\sum_{i=0}^{M} n_i - N\right) \tag{23}$$

where ρ is given in (11) and α is the Lagrange multiplier.

With the first and second assumptions, this function becomes differentiable with respect to the function arguments $\{\alpha, n_i\colon i = 0, 1, \ldots, M\}$. As the admission threshold δ is equal to double the latency, the admission threshold will be small given the third assumption. In particular, we assume that δ is sufficiently small compared to L/n_i, such that

$$\left(1 - \frac{\delta n_i}{L}\right) \to 1 \tag{24}$$

and

$$\left(\frac{L}{n_i} + \delta\right) \to \frac{L}{n_i}. \tag{25}$$

This enables us to simplify (23) to

$$F(\{n_i\}, \alpha) = \frac{1}{n_0}\left(\sum_{i=1}^{M} \frac{\lambda g_i L}{2n_i}\right) + \alpha\left(\sum_{i=0}^{M} n_i - N\right) \tag{26}$$

and obtain the partial derivatives

$$\begin{cases} \dfrac{\partial F(\{n_i\}, \alpha)}{\partial n_i} = \dfrac{-\lambda g_i L}{2n_0 n_i^2} + \alpha, \quad \text{for } i = 1 \cdots M & (27) \\ \dfrac{\partial F(\{n_i\}, \alpha)}{\partial n_0} = \dfrac{-1}{n_0^2}\displaystyle\sum_{i=1}^{M} \dfrac{\lambda g_i L}{2n_i} + \alpha & (28) \\ \dfrac{\partial F(\{n_i\}, \alpha)}{\partial \alpha} = \displaystyle\sum_{i=0}^{M} n_i - N. & (29) \end{cases}$$

Equating (27)–(29) to zero gives the following set of $(M+2)$ equations in $(M+2)$ unknowns

$$\begin{cases} n_i^2 = \dfrac{\lambda g_i L}{2n_0\alpha}, \quad \text{for } i = 1 \cdots M & (30) \\ n_0^2 = \displaystyle\sum_{i=1}^{M} \dfrac{\lambda g_i L}{2\alpha n_i} & (31) \\ \displaystyle\sum_{i=0}^{M} n_i = N. & (32) \end{cases}$$

To solve for the channel-allocation vector n_i, we first use (30) and (31) to solve for n_0

$$n_0 = \left(\frac{2\lambda L}{\alpha}\right)^{1/3}\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^{2/3}. \tag{33}$$

Now consider the total number of multicast channels

$$\sum_{i=1}^{M} n_i = \sum_{i=1}^{M}\left(\frac{\lambda g_i L}{2n_0\alpha}\right)^{1/2}. \tag{34}$$

Substituting (33) in place of n_0 in (34) gives a surprising result

$$\begin{aligned} \sum_{i=1}^{M} n_i &= \left(\frac{2\lambda L}{\alpha}\right)^{-1/3}\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^{-2/3}\sum_{i=1}^{M}\left(\frac{\lambda g_i L}{2\alpha}\right)^{1/2} \\ &= \left(\frac{2\lambda L}{\alpha}\right)^{1/3}\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^{2/3} \equiv n_0. \end{aligned} \tag{35}$$

In other words, under optimal allocation, the number of channels assigned to multicast and unicast is always the *same*. As we have not assumed any particular values for $\{g_i\}$, λ, M, L, and N, this result is true for all systems as long as δ is small.

Using this property, we can immediately obtain n_0 from

$$n_0 = \frac{N}{2}. \tag{36}$$

Equating (36) with (33), we have

$$\left(\frac{2\lambda L}{\alpha}\right)^{1/3}\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^{2/3} = \frac{N}{2} \tag{37}$$

which can be solved to obtain α

$$\alpha = \frac{4\lambda L\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^2}{N^3}. \tag{38}$$

Finally, substituting n_0 and α into (30) gives all n_i's

$$n_i^2 = \frac{g_i N^2}{4\left(\sum_{i=1}^{M} \sqrt{g_i}\right)^2} \quad \text{for } i = 1 \cdots M \tag{39}$$

for the channel-allocation vector.

V. Class-Based Popularity Model

In developing the optimization model in Section III and the approximate model in Section IV, we have assumed that individual video popularities $\{g_i | i = 1, 2, \ldots, M\}$ are known. In practice, a service provider can estimate the video popularities by collecting the on-going user access statistics over a period of time (e.g., several days) for computing and updating the channel-allocation vector. Interested readers are referred to the study by Griwodz *et al.* [17] for a more in-depth study of movie popularity models.

Nevertheless, when adding new video titles to an existing system or when setting up a new system, prior access statistics will not be available and it would be difficult to estimate the relative popularity of the new video titles.

To tackle this problem, we propose dividing videos into a smaller number of popularity classes. Each video is then classified into one of the popularity classes. All videos in a popularity class are allocated the same number of multicast channels. By decreasing the number of classes, we can simplify the classification process as well as the system implementation (e.g., disk and transmission scheduling). We will evaluate the performance tradeoff of this class-based popularity model in Section VII-C.

Let H be the number of popularity classes. Then $H = 1$ represents the uniform channel-allocation model as investigated by Lee [13] and $H = M$ represents the individual popularity model as investigated in Sections III and IV. Let c_i $(i = 1, 2, \ldots, H)$ be the aggregate popularity for class i, defined as

$$c_i = \sum_{j=(i-1)n_c}^{in_c - 1} g_j \tag{40}$$

and let a_i $(i = 1, 2, \ldots, H)$ be the aggregate arrival rate for class i, defined as

$$a_i = \sum_{j=(i-1)n_c}^{in_c - 1} \lambda_j \tag{41}$$

where $n_c = M/H$. As the number of classes is likely to be small to be practical, we assume that M is divisible by H to simplify notations. The model can be modified to cater for nondivisible cases by choosing explicit class boundaries.

To incorporate this class-based popularity model into the optimization model in Sections III and IV, we only need to replace individual video popularity with aggregate class popularity in (40), replace individual arrival rate with aggregate class arrival rate in (41), and round the resultant channel-allocation vector elements to integral multiples of class sizes. The rest of the derivations will be the same.

VI. Zero-Multicast-Channel Optimization

One of the assumptions in deriving the short-latency approximation in Section IV is that each video is allocated at least one multicast channel. For systems where the expected arrival rate is large, this assumption is valid (e.g., arrival rate of 0.5 customers/s). However, for systems designed for small arrival rate, and in particular, with a large number of videos, this assumption may lead to inefficient channel allocations. As an example, consider a system serving 100 videos with a latency constraint of one second and an arrival rate of 0.02 customers/s. The channel requirement for UVoD with uniform channel allocation and nonuniform channel allocation are 193 and 203 channels, respectively. However, TVoD under the same arrival rate requires only 174 channels. We present below a zero-multicast-channel optimization (ZMO) algorithm to tackle this deficiency under light traffic conditions.

```
Step  1: Given initial channel allocation vector {$n_i$|$i$=0…$H$};
         target latency = $w$.
Step  2: $c$ = $M$ / $H$ (i.e. number of videos per class)
Step  3: for $i$ = {$H$,$H$-1,…,1}
Step  4: {
Step  5:    if ($n_i$ == $c$) then
Step  6:    {
Step  7:       $old\_n_i$ = $n_i$
Step  8:       $n_i$ = 0
Step  9:       for $x$ = {1,2,…,c}
Step 10:       {
Step 11:          $n_0$ = $n_0$ + 1
Step 12:          compute latency $w_u${$n_j$|$j$=0…$H$}) using Eq.(13)
Step 13:          if ($w_u \le w$) then break
Step 14:       }
Step 15:       if ($w_u > w$) then
Step 16:       {
Step 17:          $n_i$ = $old\_n_i$
Step 18:          $n_0$ = $n_0$ - $c$
Step 19:       }
Step 20:    }
Step 21: }
```

Fig. 4. Pseudocode for the zero-multicast-channel optimization algorithm.

ZMO is a post-processing procedure that attempts to adjust the computed channel-allocation vector to further reduce the total channel requirement. The ZMO algorithm is shown in Fig. 4 in the form of pseudocode. The algorithm has two nested loops. The outer loop (Step 3) iterates through elements in the channel-allocation vector in reverse popularity order. For each video class where exactly one multicast channel is allocated to each video, the allocated multicast channels are first removed (Step 8). This renders videos in this class to be served solely by the unicast channels. Next, the inner loop (Step 9) computes the new latency of the system, and increases the number of unicast channels until the latency constraint is satisfied (Step 13). If the latency constraint cannot be satisfied, even if all saved channels are returned to the unicast pool, then the original multicast channels will be restored (Steps 15–19).

This ZMO algorithm can be generalized to an n-multicast-channel optimization (nMC) algorithm, which either allocates n or more channels for each video or none is allocated. This is useful in cases where the video client has limited storage for caching the multicast stream. In particular, one can adjust the channel-allocation vector to conform to the client storage specification by setting n, such that

$$T_R = \frac{L}{n} \tag{42}$$

is within the client's storage capacity.

TABLE I
LIST OF SYSTEM PARAMETERS

Parameters	Symbol	Value
Total number of available channels	N	Computed
Number of multicast channels	N_M	Computed
Number of unicast channels	N_U	Computed
Number of videos	M	10, 100,or 1000
Length of each video	L	120 minutes
Skewness for Zipf-distributed video popularity profile	θ	0.271 [14]

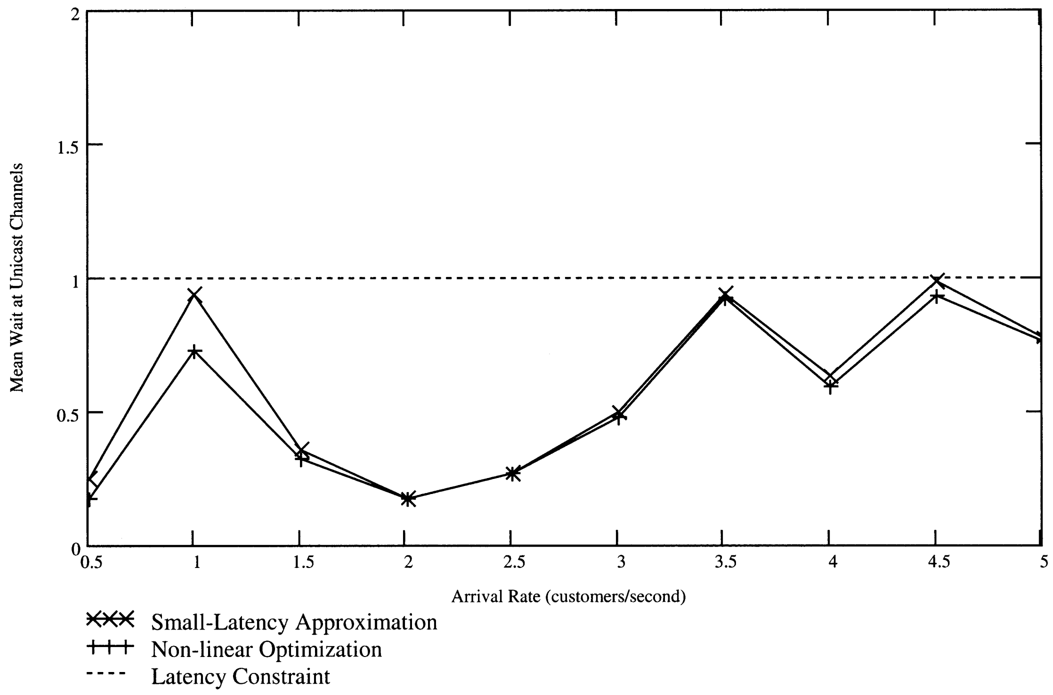

Fig. 5. Verification of the small-latency approximation model (latency constraint = 1 s).

VII. PERFORMANCE EVALUATION

In this section, we present numerical results to evaluate the channel-allocation algorithms studied in this paper. The system parameters are summarized in Table I.

A. Verification of the Small-Latency Approximation

First, we compare results computed from the nonlinear optimization model in Section III with the small-latency approximation model in Section IV. We set a target latency constraint of 1 s and then compute the channel-allocation vector from the small-latency approximation model for arrival rates ranging from 0.5 to 5 customers/s. Next, we use the total channel requirement N obtained from

$$N = \sum_{i=0}^{H} n_i \tag{43}$$

where H is the total number of popularity classes and $\{n_i\}$ is the computed channel-allocation vector, and repeat the channel-allocation process using the nonlinear optimization model.

To solve the nonlinear optimization model, we make use of MathCAD's solver with initial guess values set according to

$$\begin{cases} n_0 = N/2 \\ n_i = N/(2H), \quad i = 1, 2, \ldots, H \end{cases} \tag{44}$$

which is motivated by the observation that at small latency half of the channels are allocated for unicast (cf. Section IV) and the other half for multicast.

To compare the channel-allocation policies, we compute the latency using the channel-allocation vectors and plot the result in Fig. 5. Two sets of channel-allocation vectors at $\lambda = 1$ and $\lambda = 5$, respectively, are also listed in Table II for comparison. Clearly, both models produce very close results, verifying the small-latency approximation model.

To check how the approximation performs at larger latency, we repeat the same procedure with a latency constraint of 60 s. The resultant latency is plotted in Fig. 6. We can see that in this case the approximation model produces channel-allocation vectors with higher latency than the nonlinear optimization model. Consequently, the minimum number of channels required to satisfy the latency constraint of 60 s are also higher (see Table III).

TABLE II
COMPARISON OF CHANNEL ALLOCATION VECTORS

		n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
λ=1	NOM*	580	120	70	60	50	50	50	40	40	40	40
	SLA#	600	120	70	60	50	50	40	40	40	40	30
λ=5	NOM	1237	270	160	140	120	110	100	100	90	90	80
	SLA	1267	270	160	130	120	110	100	90	90	80	80

* NOM: Non-linear optimization mode; # SLA: Small-latency approximation model.

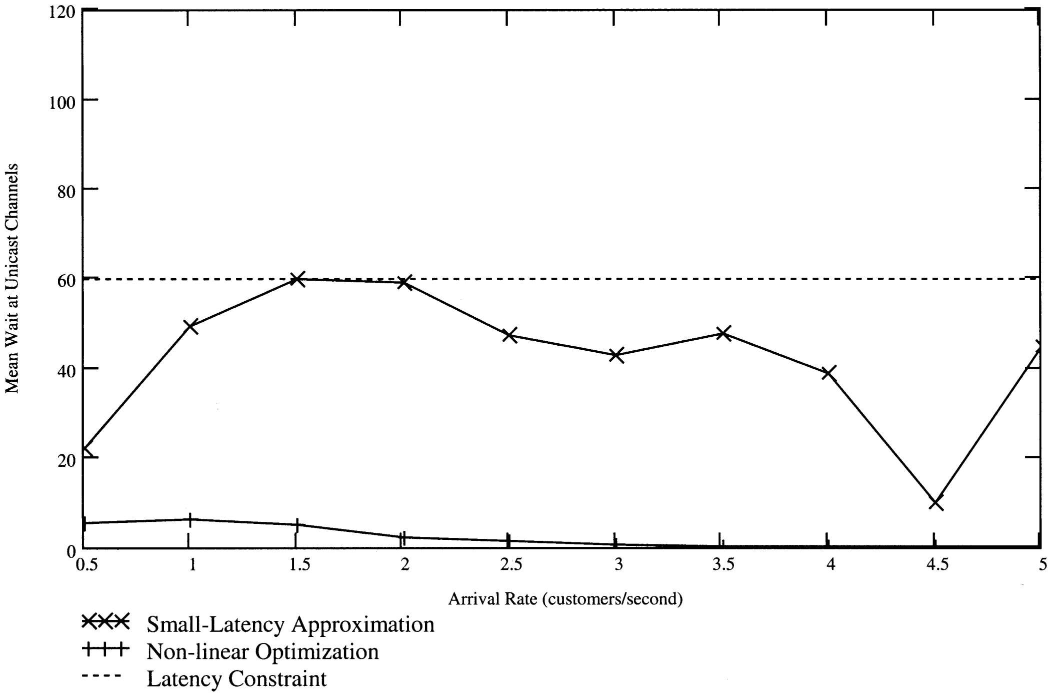

Fig. 6. Deviation of the small-latency approximation model under latency constraint of 60 s.

TABLE III
COMPARISON OF CHANNEL REQUIREMENT UNDER LARGE-LATENCY CONDITION (60 S)

Arrival Rate:	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
NOM*	732	998	1091	1347	1479	1594	1697	1791	1879	1955
SLA#	745	1010	1206	1368	1504	1627	1736	1830	1931	2009
Differences	13	12	15	21	25	33	39	39	55	54

* NOM: Non-linear optimization mode; # SLA: Small-latency approximation model.

B. Sensitivity to Arrival Rate

To facilitate comparison of nonuniform and uniform channel-allocation algorithms, we define a normalized channel reduction factor

$$R = \frac{N_{\mathrm{UCA}} - N_{\mathrm{NCA}}}{N_{\mathrm{UCA}}} \tag{45}$$

where N_{UCA} and N_{NCA} are the channel requirements under the uniform and nonuniform channel-allocation policies, respectively. The value $R(R < 1)$ can be interpreted as the proportion of channels saved by the use of nonuniform channel allocation.

The results for light traffic range and heavy traffic range are plotted in Figs. 7 and 8, respectively. We observe that channel reduction generally increases with more video selections, except at very small arrival rates. For example, the 1000-video curve drops significantly for arrival rates smaller than 0.28. In some cases (e.g., $\lambda < 0.08$), the reduction is in fact negative, i.e., more channels are required by using nonlinear channel allocation.

This poor performance at small arrival rate is a result of the requirement that each video is allocated at least one channel. Applying the zero-multicast-channel optimization increases the channel reduction dramatically as evident in Fig. 9. In particular, channel reduction for the 1000-video case increases to 55% at $\lambda = 0.08$. Moreover, the reduction never drops below zero, even at extremely small arrival rates. This is because at extremely small arrival rates, multicasting video offers no performance advantage and all the multicast channels are removed by the ZMO algorithm. The system in this case degenerates into a TVoD system and serves users using only unicast channels.

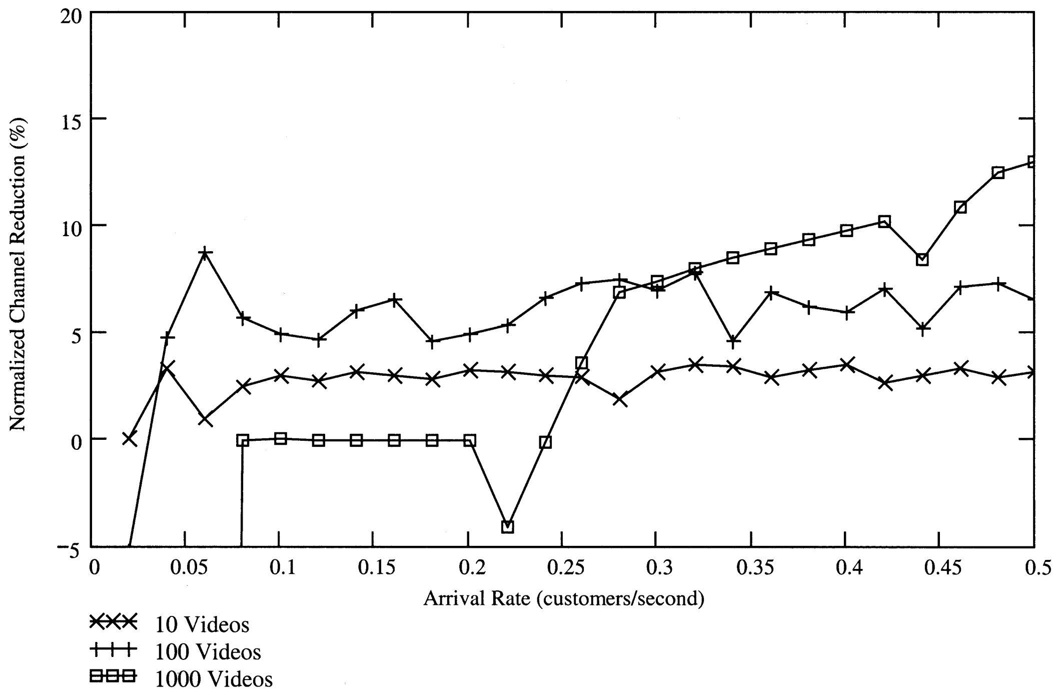

Fig. 7. Normalized channel reduction versus arrival under light traffic conditions.

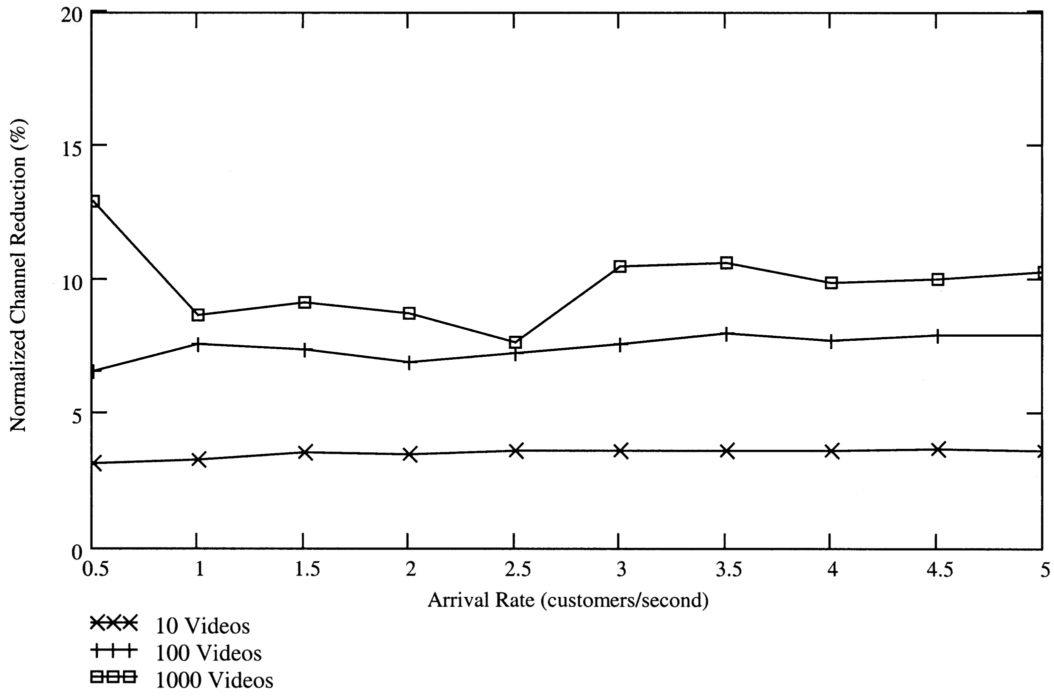

Fig. 8. Normalized channel reduction versus arrival under heavy traffic conditions.

C. Sensitivity to Video Popularity Model

Fig. 10 plots the normalized channel reduction versus video popularity skewness ranging from 0.02 to 0.5. The result shows that nonuniform channel allocation achieves better channel reduction for increased popularity skewness (note that skewness increases with decreasing value of θ). At an arrival rate of $\lambda = 1$, as shown in Fig. 10, the performance gain of ZMO is negligible, except for slight improvement in the 1000-videos case. By contrast, ZMO dramatically raises channel reduction at a lower arrival rate of $\lambda = 0.1$, as shown in Fig. 11. These results demonstrate that the ZMO algorithm is most effective at medium traffic range with high popularity skewness.

The previous results were computed by dividing the video selections into ten popularity classes, which reflects the practical difficulty of knowing the exact video popularity. To investigate the

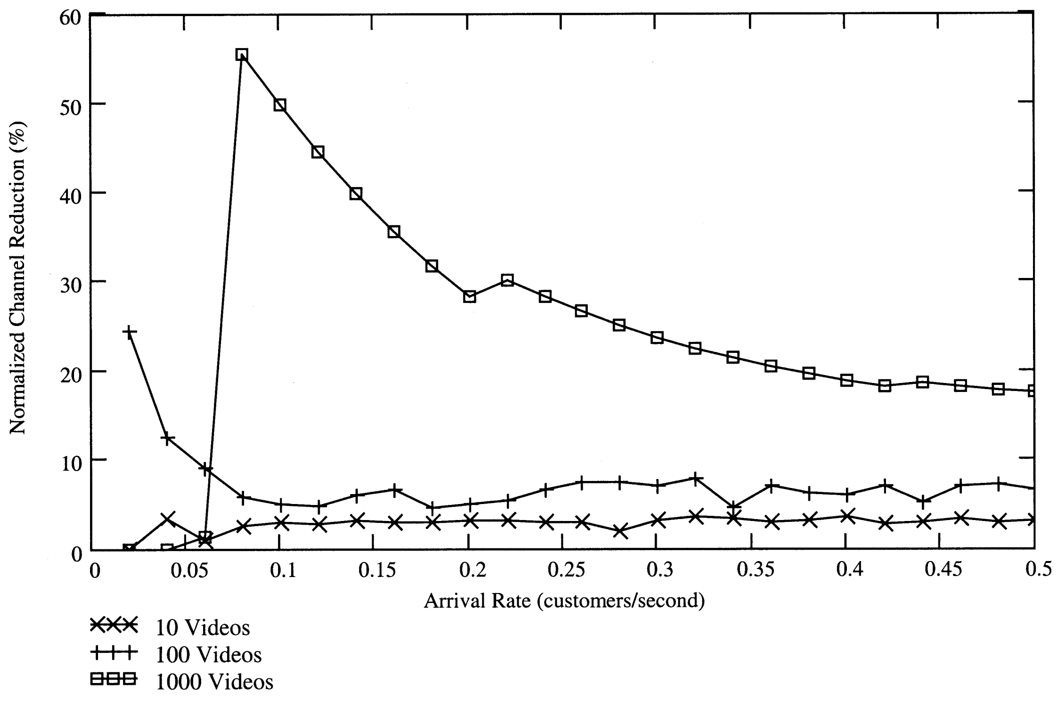

Fig. 9. Normalized channel reduction versus arrival rate with zero-multicast-channel optimization.

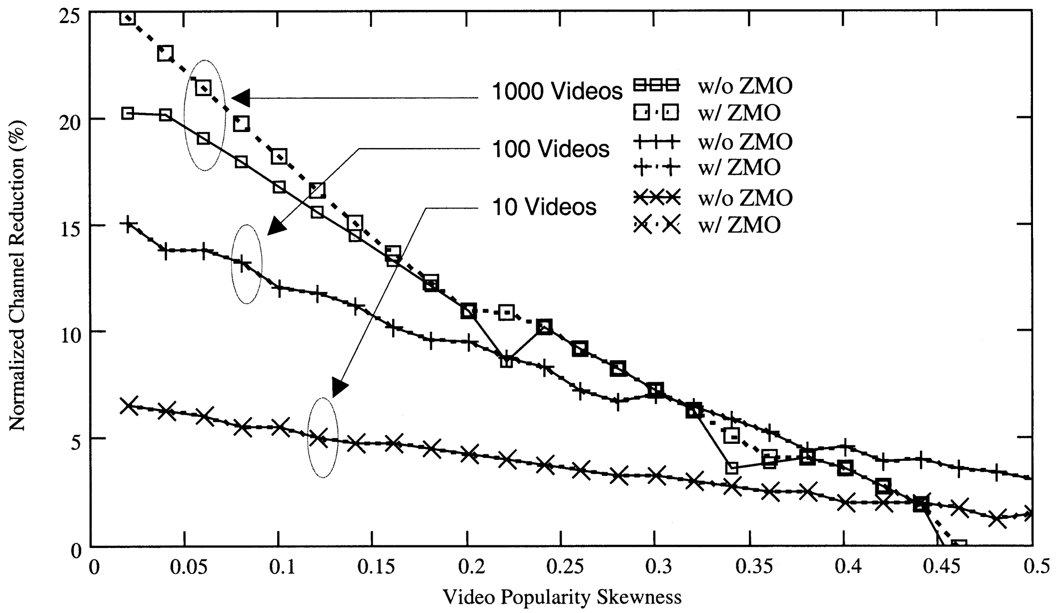

Fig. 10. Normalized channel reduction versus video popularity skewness at arrival rate $\lambda = 1$.

performance impact of this simplification, we compute and plot in Fig. 12 the normalized channel reduction for 1, 2, 5, 10, 25, 50, and 100 popularity classes, respectively. Surprisingly, channel-reduction levels off for ten or more popularity classes. This renders exact knowledge of the video popularity unnecessary and enhance the practicality of nonuniform channel allocation.

D. Channel Reduction Over TVoD

Figs. 13 and 14 plot the channel reduction over TVoD versus arrival rate, comparing the three channel-allocation algorithms: uniform channel allocation, nonuniform channel allocation, and nonuniform channel allocation with ZMO. There are two observations. First, the nonuniform channel-allocation algorithms outperform uniform channel allocation except for very small arrival rates, where they perform equally. Second, the ZMO algorithm offers substantial improvement over a range of medium arrival rates. The exact range depends on the total number of videos (e.g., $0.007 \le \lambda \le 0.04$ for 100 videos, $0.07 \le \lambda \le 0.4$ for 1000 videos), but the improvements are consistent as evident in Figs. 13 and 14.

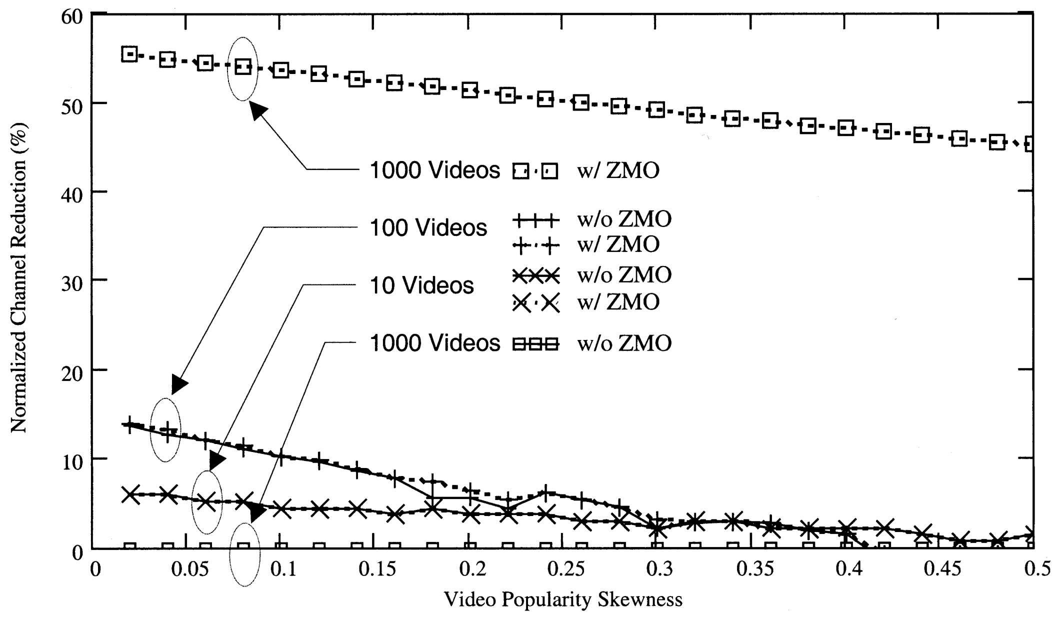

Fig. 11. Normalized channel reduction versus video popularity skewness at arrival rate $\lambda = 0.1$.

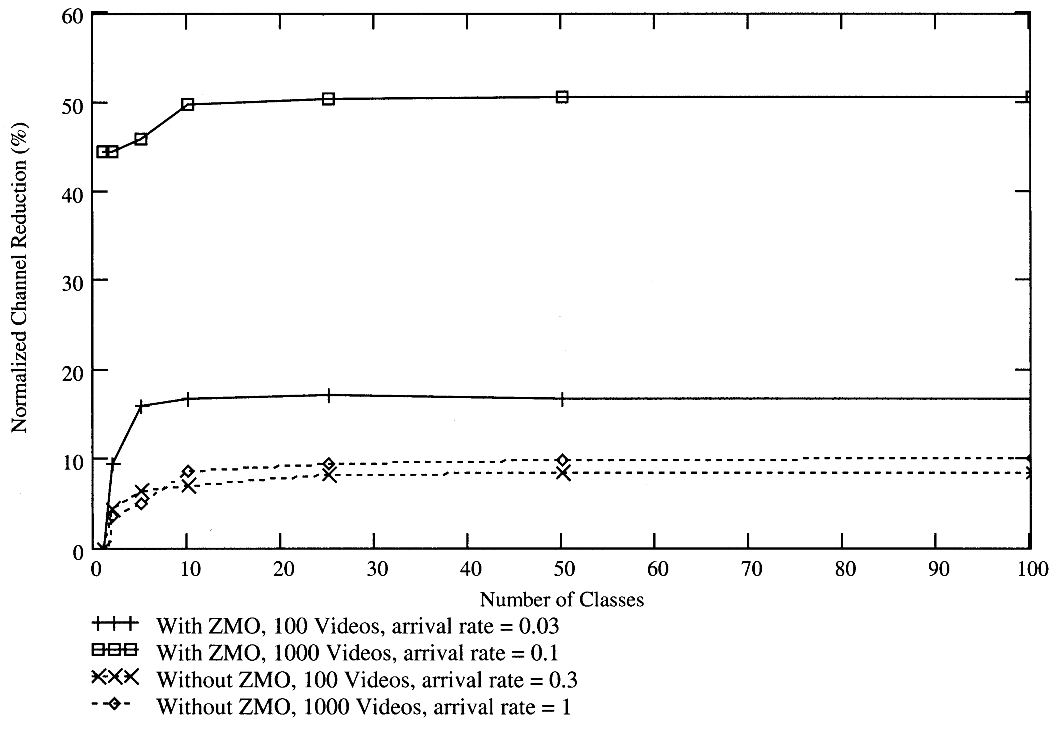

Fig. 12. Normalized channel reduction versus number of popularity classes.

VIII. Conclusion

In this study, we investigated the channel-allocation problem in a UVoD system with the goal of minimizing the channel requirement. While uniform channel allocation is simple to implement, we show that the resultant resource requirement will not be minimal as video popularity is highly skewed in practice. Nonuniform channel allocation provides a solution to this popularity skewness problem by allocating available channels according to the video popularity. In practice, a system designer can start with the small-latency approximation model to perform initial dimensioning, and then use the nonlinear optimization model for more accurate results. Provided that the video selection can be separated into around ten popularity classes, nonuniform channel allocation can offer channel reduction by as much as 50% compared to uniform channel allocation.

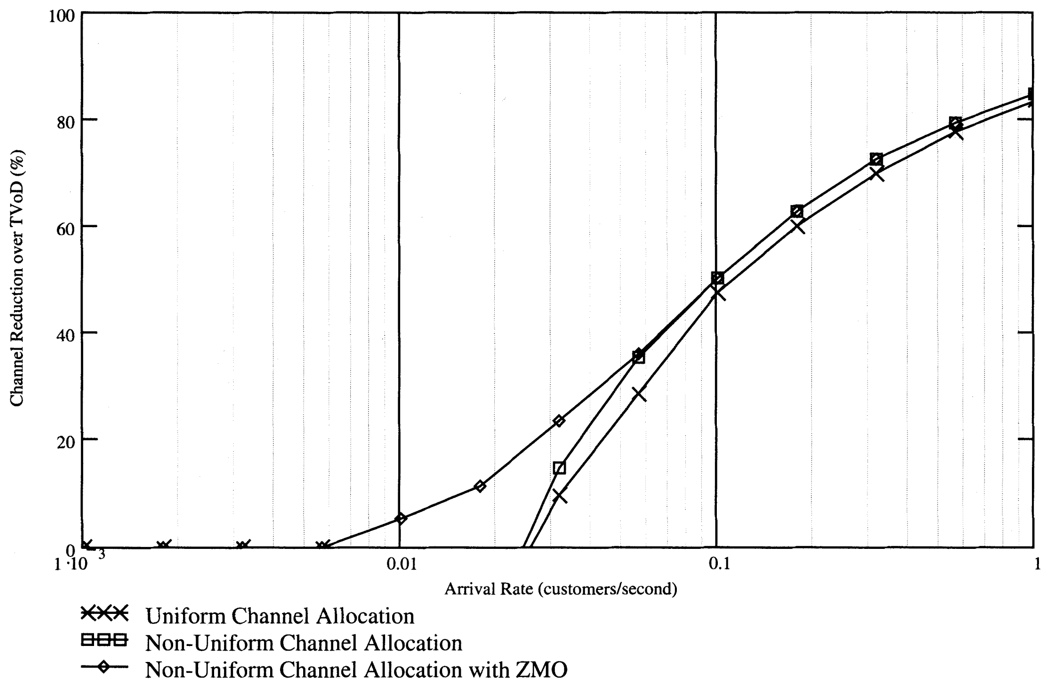

Fig. 13. Comparison of channel reductions over TVoD ($M = 100$ videos).

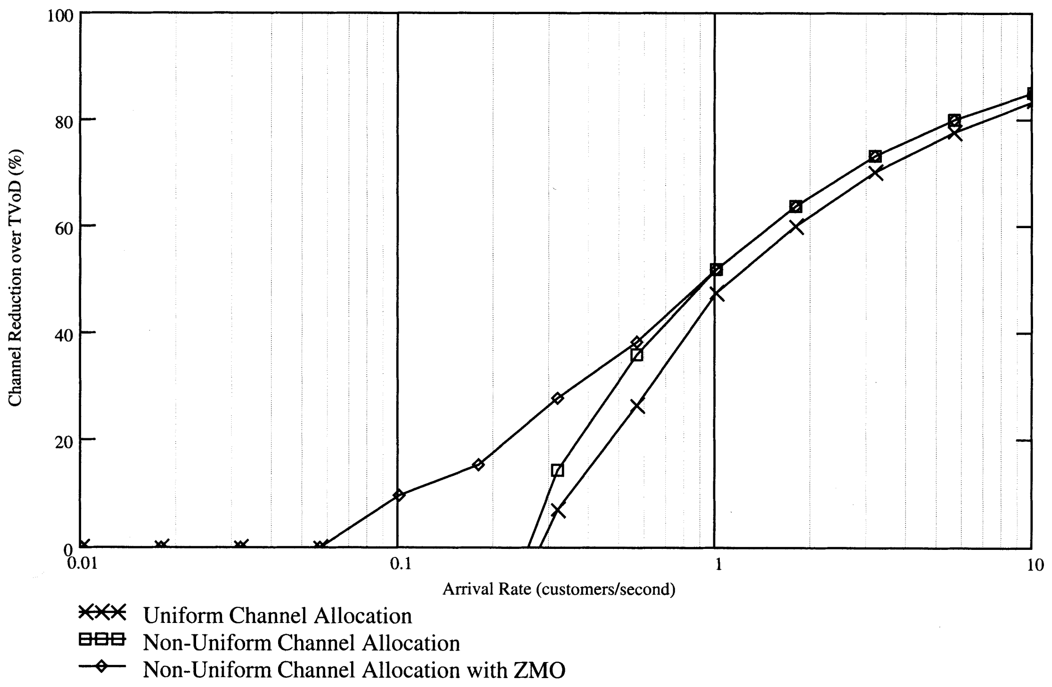

Fig. 14. Comparison of channel reductions over TVoD ($M = 1000$ videos).

Acknowledgment

The author would like to express his gratitude to the anonymous reviewers for their constructive comments and suggestions in improving this paper.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] J. Y. B. Lee, “UVoD—A unified architecture for video-on-demand services,” *IEEE Commun. Lett.*, vol. 3, no. 9, pp. 277–279, Sept. 1999.

[14] G. Zipf, *Human Behavior and the Principle of Least Effort*: Addison-Wesley, 1994.

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

[16] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

[17] C. Griwodz, M. Bar, and L. C. Wolf, “Long-term movie popularity models in video-on-demand systems,” in *Proc. 1997 ACM SIGMM*, Seattle, WA, Nov. 1997, pp. 349–357.