Channel Folding—An Algorithm to Improve Efficiency of Multicast Video-on-Demand Systems

Jack Y. B. Lee, *Senior Member, IEEE*

***Abstract*—Recently a number of researchers have proposed and investigated new video-on-demand architectures that make use of network multicast to achieve vastly improved efficiency. Techniques such as batching, patching, periodic broadcasting, chaining, and piggybacking have been explored both in isolation and in combinations. This study investigates a new tool in the arsenal—channel folding, where aggressive client-side caching is used to merge clients from one multicast channel into the other. In particular, this channel folding algorithm is applied to a previously proposed unified video-on-demand (UVoD) architecture to demonstrate and to quantify its impact on the performance and the tradeoff in a multicast video distribution architecture. This paper presents this channel folding algorithm in the context of UVoD and derives a performance model to obtain the system latency, the near-optimal channel partition policy, and the client buffer requirement. Numerical results show that channel folding can double the capacity of UVoD with a remarkably small overhead in the client buffer requirement.**

***Index Terms*—Caching, channel folding, merging, multicast, performance analysis, unified video-on-demand (UVoD), video-on-demand (VoD).**

I. Introduction

DESPITE over a decade of research in video-on-demand (VoD) systems, VoD service is still far from widespread and only a handful of cities around the world have commercial VoD services deployed. Apart from nontechnical issues such as intellectual property right concerns, the cost of infrastructure in provisioning large-scale VoD services are still prohibitively expensive.

This challenge has motivated a number of researchers to depart from the traditional unicast VoD model, where a server sends a video stream to each client individually, to new models and architectures that are based on multicast transmissions. Unlike unicast transmission, data transmitted using multicast can be received by more than one receiver. The network will duplicate these video data at network junctions for forwarding to receivers on separate network segments. In this way, the bandwidth requirement at the server and the network can be substantially reduced. The challenges are to achieve short startup latency and to support interactive playback controls such as pause–resume and slow motion despite transmitting data using multicast.

Some of these novel approaches include batching [1]–[5], chaining [6], [7], periodic broadcasting [8]–[21], patching [22]–[26], and piggybacking [27]–[30]. These techniques are often complementary and hence can be combined into even more sophisticated architectures to further improve efficiency [31]–[35]. A more detailed review of these works will be presented in Section II.

In a previous study, Lee [35] presented and analyzed a unified video-on-demand (UVoD) architecture for building efficient large-scale VoD systems. The UVoD architecture integrates multicast with unicast to vastly reduce resource requirement at heavy loads. For example, under the same latency constraint, UVoD is able to support over 400% the capacity of an equivalent TVoD system.

This study extends this previous work by integrating a new *channel folding* algorithm into the UVoD architecture to further improve efficiency. In channel folding, a client aggressively caches video data from another multicast channel so that the current multicast channel can be released long before the video session ends. As a result, the multicast channel can be reused to shorten the startup latency of waiting clients. In this paper, we present this channel folding algorithm, quantify the resource reduction, and optimize the channel partitioning policy for the modified UVoD architecture. Numerical results show that channel folding can increase the system capacity by over 100% and, even more remarkably, with little to no tradeoff in buffer requirement.

The remainder of this paper is organized as follows. Section II reviews some related previous works. Section III presents an overview of the original UVoD architecture. Section IV presents the channel folding algorithm. Section V presents a performance model for the system. Sections VI and VII present two further improvements to the channel folding algorithm. Section VIII evaluates the performance of channel folding, and Section IX concludes the paper.

II. Related Works

In this section, we briefly review the related works and compare them with this study. Our literature survey reveals that there are five common approaches to improve VoD system efficiency, namely batching, chaining, periodic broadcasting, patching, and

Manuscript received February 15, 2002; revised February 28, 2003. This work was supported in part by a Direct Grant, and Earmarked Grant (CUHK 4328/02E, CUHK 4211/03E) from the HKSAR Research Grant Council and in part by the Area of Excellence Scheme, established under the University Grants Council of the Hong Kong Special Administrative Region, China (Project AoE/E–01/99). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Edward W. Knightly.

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

Digital Object Identifier 10.1109/TMM.2005.843356

piggybacking. These approaches can be used individually or combined to form even more sophisticated architectures.

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

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

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

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

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

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

The previous five approaches are complementary and hence can be combined to form even more sophisticated architectures. For example, Liao *et al.* [22], Hua *et al.* [23], and Cao *et al.* [24] have investigated integrating batching with patching to avoid the long startup delay due to batching. Oh *et al.* [31] have proposed an adaptive hybrid technique which integrated batching with Skyscraper Broadcasting. They proposed a new batching policy called Largest Aggregated Waiting Time First (LAW), which is a refinement of the MFQ policy [4]. Unlike most studies that assume stationary video popularity, their approach estimates video popularity using an online algorithm and revises the Skyscraper Broadcasting schedule from time to time to adapt to video popularity changes.

More recently, Gao *et al.* [32] proposed a *controlled multicast* technique that integrated patching with dynamically scheduled multicasting. This is further refined by Gao *et al.* [33] in their *catching* scheme, which employed the Greedy Disk-Conserving Broadcasting schedule for the periodic broadcasting channels. Their study found out that catching outperforms controlled multicast at high loads but is otherwise not as good as controlled multicast. This motivated them to further combine catching with controlled multicast to form a *selective catching* scheme that dynamically switches between catching and controlled multicast depending on the system load.

In another study, Ramesh *et al.* [34] proposed and analyzed the multicast with cache (*Mcache*) approach that integrated batching, patching, and prefix caching. They proposed placing regional cache servers close to the users to serve the initial portion (prefix) of the videos. In this way, a client can start video playback immediately by receiving prefix data streamed from a regional cache server. The server will then dynamically schedule a patching channel for the client to continue the patching process beyond the prefix and identify an existing multicast channel for the client to cache and eventually merge into.

In the previous work leading to this study, Lee [35] proposed and analyzed the unified video-on-demand (UVoD) architecture that integrated periodic broadcasting with patching. The UVoD architecture includes the traditional TVoD and NVoD as special cases of the architecture but generally outperforms them if properly configured. The study presented a performance model for the architecture and derived the optimal policy to allocate channels for patching and periodic broadcasting purposes.

The channel folding algorithm presented in this paper is a new approach that is orthogonal to the five approaches previously discussed. Therefore, while we focus on integrating channel folding to UVoD in this study, it is plausible that channel folding can also be integrated into other architectures. More investigations will be needed to characterize and quantify the performance impact to these other architectures.

III. System Architecture

In this section, we briefly review the architecture of UVoD, with a focus on channel allocation, scheduling, and admission algorithm. Interested readers are referred to [35] for details on other aspects of UVoD.

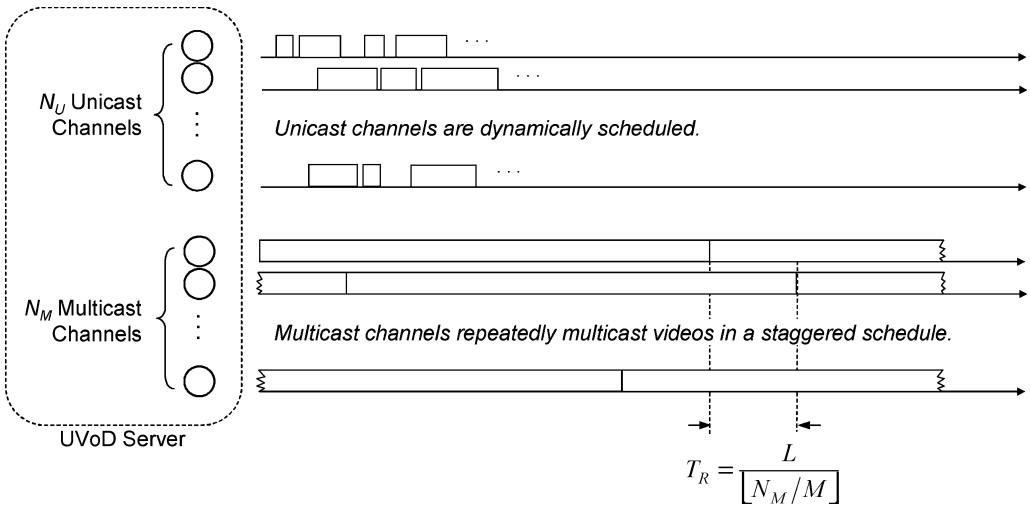


Fig. 1. Architecture of the UVoD server.

A. Channel Allocation and Scheduling

Fig. 1 depicts the architecture of the UVoD server. There are a total of N channels for streaming video data. A channel is the collection of resources, such as I/O capacity at the server and network bandwidth, for streaming one video stream. We assume all video titles are encoded with the same parameters to produce constant-bit-rate compressed bit streams at a rate of R_V bytes per second.

The N channels are then allocated for two purposes: N_M of them for periodic broadcasting and the remaining $N_U = N - N_M$ for patching. Each periodic broadcasting channel transmits the same video title over and over using multicast as in a NVoD system. Adjacent multicast channels broadcasting the same video title are offset by a time interval of

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

seconds, where M denotes the number of video titles in the system, each having the same length denoted by L.

These multicast channels will carry the bulk of the video data to the clients. However, since a client can start playback at any time, the data it needs may not be immediately available through the multicast channels. In the worst case, it may take up to T_R seconds to wait for the needed data to become available through the multicast channels. This is precisely the same limitation of NVoD systems. The UVoD architecture tackles this problem by dynamically allocating a patching channel to sustain playback while the client caches data from a multicast channel that it eventually merges into. The next section describes details of this admission process.

B. Admission Algorithms

A client can be admitted to a UVoD system in two ways, namely, *admit-via-unicast* and *admit-via-multicast*. When a user requests a new video session, say at time t_u, 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 (i.e., admit-via-multicast) if the waiting time is smaller than a predetermined admission threshold δ

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

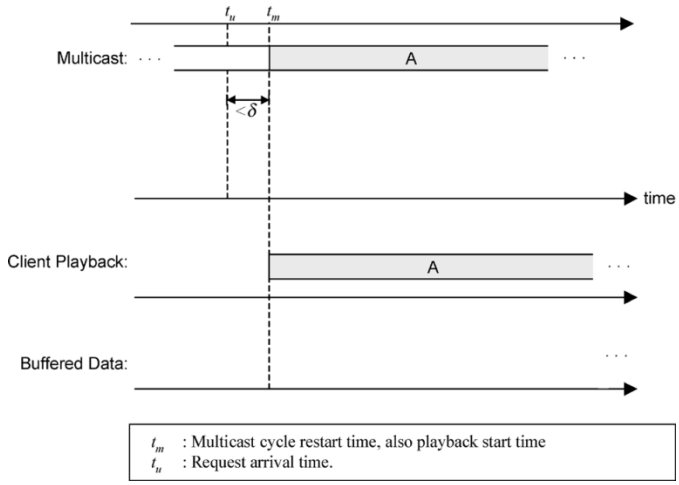


Fig. 2. Admission of a new user through the admit-via-multicast process.

The user then simply receives and plays back video data (segment "A" in Fig. 2) from the multicast channel for the rest of the session, as depicted in Fig. 2. The admission threshold is introduced to divert a portion of the traffic to the multicast channels. By setting the value of δ, one can control the latency experienced by the users admitted through this admit-via-multicast process.

If (2) is not satisfied, then the system will assign the user to wait for a free unicast channel to start playback using the admit-via-unicast process depicted in Fig. 3. Specifically, immediately upon arrival at time t_u, the client begins caching video data from the previous multicast of the requested video title (segment "B" in Fig. 3). At the same time, it waits for a free unicast channel to begin playback, i.e., receive and playback segment "A" from time t_v in Fig. 3. Let t_m and t_{m+1} be the nearest epoch times (i.e., time when the video is restarted from the beginning) of multicast channel m and channel $m+1$, for which $t_m < t_u < (t_{m+1} - \delta)$. Then the client will have cached video data from

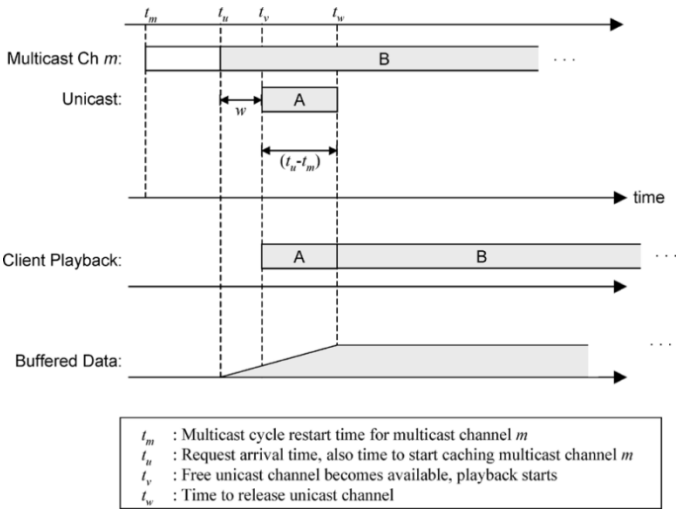


Fig. 3. Admission of a new user through the admit-via-unicast process.

multicast channel m for the video starting from video time $(t_u - t_m)$, where video time is the time offset relative to the beginning of the video. Therefore once video playback reaches time $t_w = t_v + (t_u - t_m)$, the unicast channel can be released and the client continues playback using data received from multicast channel m through the buffer. The local buffers effectively add time delay to the multicast video stream so that it matches the client's playback schedule.

This architecture achieves resource reduction in two ways. First, a portion of the users will be admitted to 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. Second, for admit-via-unicast users, since $0 < (t_u - t_m) < (T_R - \delta) \ll L$, we can see that the unicast channels are occupied for much shorter duration compared to the length of the video L. Therefore this substantially reduces the load at the unicast channels to allow far more requests to be served using the same number of channels.

IV. Channel Folding Algorithm

One of the keys to UVoD's improved performance is the use of caching to reduce the unicast channels' channel-holding time. In this study, we extend this principle into a channel folding algorithm to reduce the channel-holding time of some of the *multicast* channels. Channel folding takes advantage of additional client-side storage and aggressively caches data into the client so that the client can release the current multicast channel by merging into an adjacent multicast channel.

Specifically, the multicast channels are divided into even channels (i.e., channel $0, 2, 4, \ldots$) and odd channels (channel $1, 3, 5, \ldots$). Users admitted to an even channel will be served using the original admission algorithms while users admitted to an odd channel will be subject to channel folding, which eventually merges to an adjacent even channel. Therefore, on average, half of the admitted users will go through the channel folding process. In the following sections, we present the channel folding algorithm as applied to admit-via-multicast and admit-via-unicast users on those odd channels. Similar to the original UVoD architecture we assume a client can receive data from up to two channels simultaneously. Table I summarizes the notations used in this section.

TABLE I
Summary of Notations

Description	Symbol
Multicast cycle restart time for channel n+1	t_m
Multicast cycle restart time for channel n+2	t_{m+1}
Request arrival time	t_u
Playback start time	t_v
Time to start caching channel n	t_w
Time to release channel n+1	t_x
Video time at which the client begins caching from channel n	t_c
Waiting time for a unicast channel	w
Admission threshold	δ
Offset for multicast channels	T_R

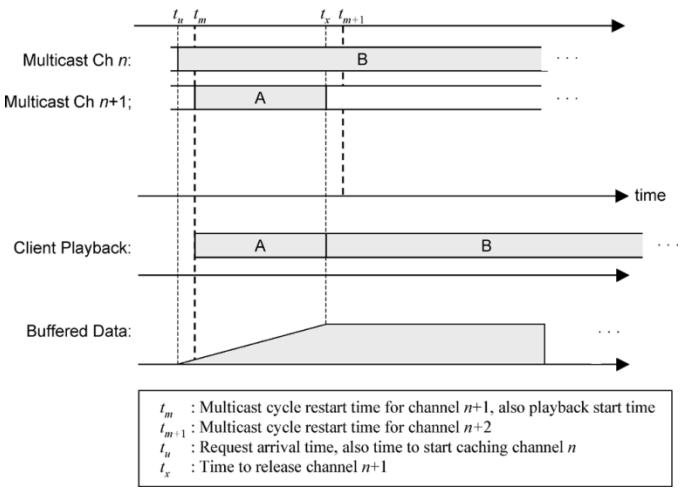


Fig. 4. Admission through admit-via-multicast and channel folding.

A. Algorithm for Admit-via-Multicast Clients

For an admit-via-multicast client, the channel folding process begins at the instant the client arrives at the system. Let the client arrives at the system at time t_u, waiting to begin playback at time t_m using multicast channel $n+1$, as shown in Fig. 4. Note that n is an even integer so channel $n+1$ is an odd channel.

In the original UVoD architecture and ignoring interactive playback control, this client simply receives all its data from multicast channel $n+1$ for the entire video session from time t_m to $t_m + L$. By contrast, with channel folding the client will begin caching data from multicast channel n immediately upon arriving at time t_u. As channel n begins its multicast cycle at a prior time $t_m - T_R$, the client can only cache data beginning from video time

$$t_c = t_u - (t_m - T_R). \tag{3}$$

The client continues to wait until t_m to begin playback via data received from channel $n+1$. As data beginning from video time t_c has been cached, it is easy to see that the client can leave channel $n+1$ at time

$$t_x = t_m + t_c \tag{4}$$

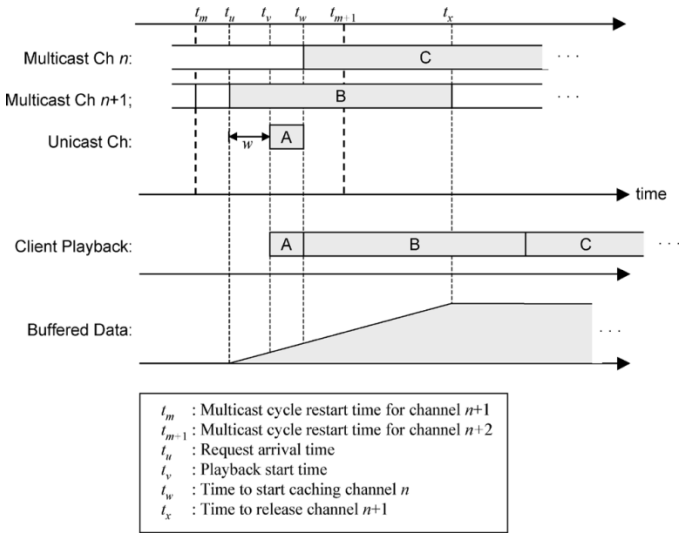


Fig. 5. Admission through admit-via-unicast and channel folding.

and then continue playback using data received from channel n through the cache. Note that more than one client can share channel $n+1$, and it is therefore necessary to wait for all clients on channel $n+1$ to complete channel folding before channel $n+1$ can be released.

As channel $n+1$ starts at time t_m and a client leaves at time t_x, the channel holding time is given by

$$s = t_x - t_m. \tag{5}$$

Substituting (3) and (4) into (5) gives

$$s = t_m + t_u - t_m + T_R - t_m. \tag{6}$$

Noting that $(t_m - \delta) \le t_u < t_m$, we can conclude that

$$T_R - \delta \le s < T_R, \quad \forall t_u, t_m. \tag{7}$$

In other words, all admit-via-multicast clients will have left channel $n+1$ by time

$$t_x = t_m + T_R. \tag{8}$$

B. Algorithm for Admit-via-Unicast Clients

For a client admitted through admit-via-unicast as depicted in Fig. 5, the channel folding process does not begin immediately upon the client's arrival. This is because during the startup phase, the client already has to receive two channels of data, one from a unicast channel for playback and the other from multicast channel $n+1$, thereby fully utilizing the client access bandwidth. This restriction can be relaxed if the client has additional access bandwidth (e.g., can receive three or more channels concurrently).

Let w be the time to wait for a free unicast channel. Then the startup phase will end at time

$$t_w = t_u + w + (t_u - t_m). \tag{9}$$

This is also the time when channel folding begins and the client starts caching data from channel n, which by then is transmitting data corresponding to video time

$$t_c = t_w - (t_m - T_R). \tag{10}$$

As data beginning from video time t_c has been cached from channel n, it is easy to see that the client can leave channel $n+1$ at time

$$t_x = t_m + t_c \tag{11}$$

and then continue playback using data received from channel n through the cache.

Substituting (9) and (10) into (11) gives

$$t_x = t_m + (t_u + w + (t_u - t_m)) - (t_m - T_R) \tag{12}$$

which simplifies to

$$t_x = 2t_u + w - t_m + T_R. \tag{13}$$

Again there can be more than one admit-via-unicast clients sharing channel $n+1$, and so it is necessary to wait for all clients on channel $n+1$ to complete the channel folding process before channel $n+1$ can be released. We note that the client will only wait until time t_{m+1} for a free unicast channel because otherwise the client can simply join channel $n+2$ to begin playback (i.e., revert to admit-via-multicast). Therefore, the waiting time is bounded by

$$0 \le w < t_{m+1} - t_u. \tag{14}$$

Together with the observation that $t_m < t_u < t_{m+1} - \delta$, we substitute the upper bound of w into (13) to obtain

$$\begin{aligned} t_x &< 2t_u + (t_{m+1} - t_u) - t_m + T_R \\ &= t_u + (t_m + T_R) - t_m + T_R \qquad \because t_{m+1} = (t_m + T_R) \\ &< (t_{m+1} - \delta) + 2T_R \\ &= t_m + 3T_R - \delta, \quad \forall t_u, t_m, t_{m+1}. \end{aligned} \tag{15}$$

In other words, all admit-via-unicast clients will have left channel $n+1$ latest by time t_x. As the multicast channel is shared by both admit-via-multicast and admit-via-unicast clients, the channel can only be released after both types of clients have all left the channel. Comparing (8) and (15), it is easy to see that, for small δ, admit-via-unicast clients always has longer channel holding time than admit-via-multicast clients. Thus, an odd multicast channel will be occupied for at most $3T_R$ seconds when $\delta = 0$.

C. Reusing Folded Channels

Channel folding enables all odd channels to be released $3T_R$ seconds after the start of a multicast cycle. This enables the system to share a channel among odd multicast channels. Specifically, an odd channel is utilized for only three out of the N_M segments of T_R seconds. As there are a total of $0.5N_M$

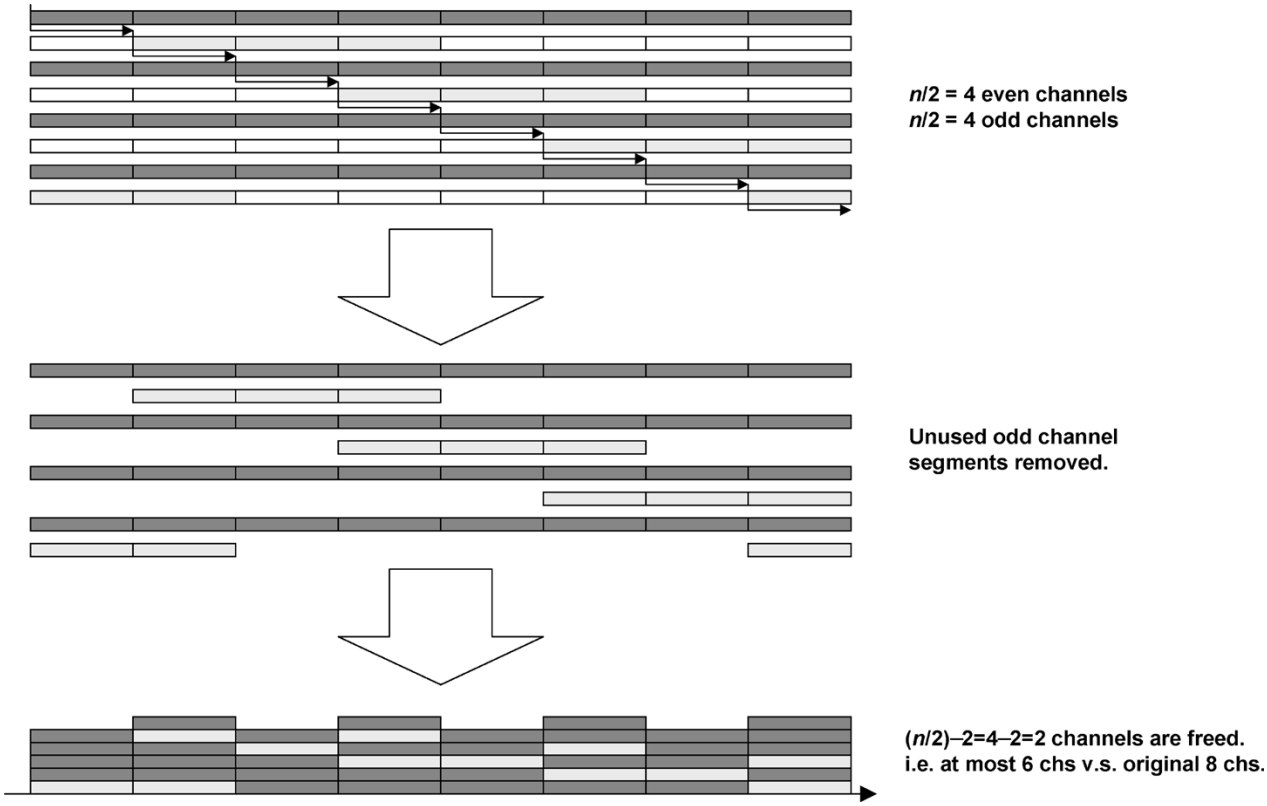


Fig. 6. Channel requirement after channel folding is applied (eight multicast channels).

odd channels, the average aggregate channel utilization is thus equal to $\lceil 0.5N_M(3/N_M)\rceil = 2$ channels. Moreover, we note that, at most, two odd channels will be active simultaneously because adjacent odd channels are offset by $2T_R$ seconds. Thus, channel folding reduces the number of channels used for odd channels from $0.5N_M$ to only two channels, i.e., a reduction of $(0.5N_M - 2)$ channels.

For example, consider the case in Fig. 6 with eight multicast channels. After removing the released channel segments and combining the active channel segments, the aggregate channel requirements will then be reduced to six channels. The two channels released can then be reused as unicast channels for admitting clients through the admit-via-unicast process, thereby reducing the time to wait for a free unicast channel. We formulate a performance model in the next section to quantify this performance gain.

V. Performance Modeling

Channel folding has three impacts on UVoD. First, the client buffer requirement is increased because more data need to be cached to perform channel folding. This is the price to pay for the performance gain. Second, the total number of channels required to achieve the same latency as the original UVoD is reduced. Third, the near-optimal channel allocation derived by Lee [35] no longer holds, and a new channel partition policy is needed. We address these impacts in the following three sections.

A. Latency

In [35], Lee has derived a performance model for UVoD. Due to space limitation, we do not repeat the derivations here and simply define a function $W(N_U, N_M)$ to return the latency achieved by UVoD using N_U unicast channels and N_M multicast channels. Interested readers are referred to [35] for details of the derivations.

From Section IV, we found that channel folding can release $(0.5N_M - 2)$ multicast channels. These free channels can be added to the pool of unicast channels to further reduce latency. Define $W_{CF}(N_U, N_M)$ as the latency function for UVoD with channel folding, and then it can be computed from

$$W_{CF}(N_U, N_M) = W(N_U + \max(\lfloor 0.5N_M \rfloor - 2, 0), N_M) \tag{16}$$

which accounts for the additional unicast channels made available by channel folding. We can also conclude that channel folding is effective only if $(\lfloor 0.5N_M \rfloor - 2) > 0$ or $N_M \geq 6$.

B. Channel Partition

Channel partition refers to the policy to divide available channels for use as unicast and multicast channels. Intuitively, too many multicast channels will leave too few channels for unicast, which may lead to overflow at the unicast channels. On the other hand, too few multicast channels will increase channel-holding time for requests entering the unicast channels, which again may lead to overflow.

In the original UVoD architecture, Lee [35] derived a near-optimal channel allocation policy for computing the number of channels allocated to multicast channels

$$n = \left\langle \frac{LN}{2LM - \delta N} \right\rangle \tag{17}$$

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

As additional channels are now made available by channel folding, the same channel allocation policy may no longer be valid. In the following, we derive the new near-optimal channel partition policy for UVoD incorporating the effect of channel folding.

Let λ be the rate at which new users arrive at the system. We assume that the videos have an arbitrary popularity profile given by $\{g_i,\ 1 \le i \le M\}$ where g_i is the probability that a new user requests video i. Without loss of generality, we assume that the video numbers are assigned according to popularity where $g_i \ge g_j\, \forall i < j$. Clearly we must have

$$\sum_{i=1}^{M} g_i = 1. \tag{18}$$

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

$$u_i = \frac{\lambda g_i}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right) \tag{19}$$

where n is the number of multicast channels assigned for each video, $(1 - (\delta n/L))$ is the proportion of requests routed to the unicast channels, and $(1/2)((L/n) - \delta)$ is the average service time.

In the original UVoD, the number of unicast channels is given by $N - Mn$. With channel folding, this number is increased to

$$N - Mn + M(\max\{\lfloor 0.5n \rfloor - 2, 0\}). \tag{20}$$

Therefore, the load at the unicast channels, denoted by ρ, becomes

$$\rho = \frac{\sum_{i=1}^{M} u_i}{N - Mn + M(\max\{\lfloor 0.5n \rfloor - 2, 0\})}. \tag{21}$$

Substituting (19) into (21) gives

$$\rho = \frac{\sum_{i=1}^{M} \frac{\lambda g_i}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn + M(\max\{\lfloor 0.5n \rfloor - 2, 0\})}. \tag{22}$$

Collecting terms gives

$$\rho = \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)\sum_{i=1}^{M} g_i}{N - Mn + M\left(\max\{\lfloor 0.5n \rfloor - 2, 0\}\right)}. \tag{23}$$

Noting (18), we can then drop g_i altogether as follows:

$$\rho = \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn + M\left(\max\{\lfloor 0.5n \rfloor - 2, 0\}\right)}. \tag{24}$$

Now, for $n < 6$, the last term in the denominator will become zero, and (24) reduces to

$$\rho = \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn}, \quad \because M(\max\{\lfloor 0.5n \rfloor - 2, 0\}) = 0, \forall n < 6 \tag{25}$$

which is the same as the original UVoD [35]. As $n < 6$ whenever $N < 6$, the optimal channel partition will be the same as the original UVoD, given by

$$n = \frac{LN}{2LM - \delta N}, \quad \forall N < 6. \tag{26}$$

For $N \ge 6$ and assuming it is divisible by 2, we can then rewrite (24) as

$$\rho = \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn + M(0.5n - 2)}. \tag{27}$$

It can be shown that (27) is a convex function between $n = 1$ and $n = N$. Thus, one way to find the n that minimizes ρ is to differentiate (27) with respect to n

$$\frac{\partial \rho}{\partial n} = \left(\frac{(N - 2M)\delta^2}{L} - \delta M\right) n^2 + LMn - L(N - 2M) \tag{28}$$

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

$$n = \frac{L(N - 2M)}{LM - \delta(N - 2M)}. \tag{29}$$

However, we note that (27) is defined only for $1 \le n \le N$, and thus we need to verify the optimal n computed from (29) to ensure that it is within the valid range. Moreover, in practice, n is discrete instead of continuous and as static multicast channels are grouped into pairs, n should be multiples of two as well. Adding these constraints and the constraint that the total number of channels allocated to multicast cannot be larger than N, we can obtain the near-optimal number of static multicast channels per video title from

$$n_{CF} = \max\left(\left\langle \frac{L(N - 2M)}{LM - \delta(N - 2M)} \cdot \frac{1}{2} \right\rangle 2, \frac{N}{M}\right). \tag{30}$$

To integrate this channel-partition policy into the derivation in Section V-A, we can simply replace the variable N_U and N_M in (16) by $(N - Mn_{CF})$ and Mn_{CF}, respectively, to obtain the new latency accordingly.

C. Client Buffer Requirement

To support channel folding, the client will need to have additional buffer space to cache video data for later playback. For the

patching operation, it is easy to see that the added buffer requirement is at most equal to T_R seconds' worth of video data as the patching phase can sustain for at most T_R seconds. Assuming video is encoded with constant-bit-rate (CBR) algorithm at an average rate R_V, the buffer requirement will be equal to $R_V T_R$ bytes.

For the folding operation, additional video data may need to be buffered and intuitively we need an additional buffer large enough to store $2T_R$ seconds' worth of video data. This gives a total client buffer requirement of $3R_V T_R$ bytes. However, we show in the following that this intuitive argument is too loose by deriving the buffer requirement from first principles.

We first consider the case of admit-via-multicast clients as depicted in Fig. 4. The client arrives at time t_u, immediately begins caching data from channel n while waiting to begin playback using channel $n+1$ at time t_m. At a later time t_x, the client will have reached the point where the data cached from channel n begins, and at this instant the client can release channel $n+1$ and continue playback via data from the cache.

To determine the buffer requirement in this scenario, we first note that the amount of data accumulated in the client buffer will never decrease during the startup phase. This is because the client will always be receiving data from at least one channel (at a rate R_V) after arrival at time t_u. Now given the data consumption rate during playback is also R_V, it is easy to see that the amount of accumulated data cannot decrease.

A consequence of the previous nondecreasing property is that the amount of data buffered at time t_x, i.e., time for which the startup phase completes, will determine the buffer requirement. This amount is simply equal to the total amount of data received by the time t_x, denoted by b_r, minus the total amount of data consumed by the time t_x, denoted by b_c as

$$B = b_r - b_c. \tag{31}$$

To compute b_r, we note that the total amount of data received from channel n is equal to

$$R_V(t_x - t_u) \tag{32}$$

and the total amount of data received from channel $n+1$ is equal to

$$R_V(t_x - t_m). \tag{33}$$

Therefore, b_r is simply equal to the sum of (32) and (33).

To compute b_c, we note that video playback starts from time t_m and, hence, by the time t_x, the amount of data consumed will be equal to

$$b_c = R_V(t_x - t_m) \tag{34}$$

and hence the buffer requirement can be computed from

$$\begin{aligned} B &= R_V(t_x - t_u) + R_V(t_x - t_m) - R_V(t_x - t_m) \\ &= R_V(t_x - t_u). \end{aligned} \tag{35}$$

We observe that t_x can be expressed in terms of t_u as

$$\begin{aligned} t_x &= t_{m+1} - (t_m - t_u) \\ &= t_m + T_R - (t_m - t_u). \end{aligned} \tag{36}$$

Rearranging (36), we have

$$t_x - t_u = T_R \tag{37}$$

and hence the buffer requirement is a constant and is equal to

$$B_{\max} = R_V T_R. \tag{38}$$

Next, we consider the case of admit-via-unicast clients, as depicted in Fig. 5. The client arrives at time t_u and immediately begins caching data from channel $n+1$ while waiting to begin playback using a unicast channel at time t_v. At a later time t_w, the client will have reached the point where the data cached from channel $n+1$ begins, and at this instant the client can release the unicast channel and continue playback via data from the cache. At the same time, the client starts caching data from channel n until a later time t_x when playback reaches the point where cached data from channel n begins. At this time, channel $n+1$ can be released and playback continues with data cached from channel n.

Using similar derivations, we can compute the total amount of data received by time t_x from

$$b_r = R_V(t_w - t_v) + R_V(t_x - t_u) + R_V(t_x - t_w) \tag{39}$$

where the first, second, and third terms are the amount of data received from the unicast channel, channel $n+1$, and channel n, respectively.

Now, as playback starts from time t_v, the amount of data consumed by time t_x is given by

$$b_c = R_V(t_x - t_v) \tag{40}$$

and the total buffer requirement becomes

$$\begin{aligned} B &= b_r - b_c \\ &= R_V(t_w - t_v) + R_V(t_x - t_u) \\ &\quad + R_V(t_x - t_w) - R_V(t_x - t_v) \\ &= R_V(t_x - t_u). \end{aligned} \tag{41}$$

Again, we express t_x in terms of t_u as follows:

$$\begin{aligned} t_x &= t_{m+1} + (t_u - t_m) + w + (t_u - t_m) \\ &= (t_m + T_R) + 2(t_u - t_m) + w \\ &= T_R + 2t_u - t_m + w. \end{aligned} \tag{42}$$

Rearranging, we have

$$t_x - t_u = T_R + t_u - t_m + w. \tag{43}$$

Consider w: it is bounded from above by

$$w \le T_R - (t_u - t_m). \tag{44}$$

Rearranging, we have

$$(t_u - t_m) + w \leq T_R. \tag{45}$$

Substituting into (43) gives

$$t_x - t_u \leq 2T_R \tag{46}$$

and hence the maximum buffer requirement is equal to

$$B_{\max} = 2R_V T_R. \tag{47}$$

As admit-via-multicast does not need any extra buffer for caching purpose, (47) then determines the client buffer requirement.

VI. Over Allocation of Multicast Channels

In deriving the near-optimal channel partition policy in (30), we have included a constraint to ensure that the number of multicast channels allocated will not exceed the total number of available channels, i.e., $nM \leq N$. While intuitively sound, this constraint can in fact be relaxed because odd multicast channels are not occupied for the entire video duration. In particular, the channel is occupied only during the first three segments of each multicast cycle.

This observation motivates us to relax the constraint: $nM \leq N$ to allow the allocation of more multicast channels than the total N – *over allocation*. To determine the new constraint, we begin with n multicast channels, of which $n/2$ channels are fixed and used as even channels. The channel folding process will require no more than two multicast channels at any one time, regardless of n (cf. Section IV-C). Therefore, the channel requirement after channel folding is equal to

$$\left(\frac{n}{2} + 2\right) M \tag{48}$$

and this must be no larger than the total number of channels available

$$\left(\frac{n}{2} + 2\right) M \leq N. \tag{49}$$

Rearranging, we can obtain the new constraint from

$$n \leq 2\left(\frac{N}{M} - 2\right) \tag{50}$$

and (30) can be revised to include this new constraint

$$n_{OA} = \max\left(\left\langle \frac{L(N-2M)}{LM - \delta(N-2M)} \cdot \frac{1}{2} \right\rangle 2, 2\left(\frac{N}{M} - 2\right)\right). \tag{51}$$

For example, given $N = 200$ channels and $M = 10$ videos, the channel partition policy in (30) allows allocation of up to 20 channels per video—ten for unicast and ten for multicast. Now, as channel folding reduces the channel requirement from ten to two, eight of the multicast channels can be reused as unicast channels, adding up to $10 + 8 = 18$ unicast channels. This configuration is equivalent to a 36-channel UVoD system without channel folding.

VII. Complementary Channel Scheduling

Another observation in Fig. 6 is that the aggregate number of multicast channels needed is not a constant. Instead, the number alternates between $(n/(2M)+2)$ and $(n/(2M)+1)$, where n is the number of multicast channels allocated. This oscillation is due to the fact that an odd channel is occupied for three repeating intervals while adjacent odd channels are time-staggered by two repeating intervals, as shown in Fig. 6. As a result, half a channel on average would be unused. For a system with M videos, the aggregate idling capacity would add up to $M/2$ channels, not an insignificant number in large systems with tens to hundreds of videos. To tackle this problem, we present below a complementary channel scheduling technique to combine the idle capacity for reuse.

The key idea of complementary channel scheduling is to schedule multicast channels for a pair of videos at a time so that their multicast schedules complements each other, i.e., when one of the video requires $(n/(2M) + 2)$ channels, the other video will require $(n/(2M) + 1)$ channels. Fig. 7 depicts this technique with $n =$ eight channels and $M =$ two videos. Let the repeating intervals be numbered from 0, 1, and so on, then the first video is scheduled with multicast channel i $(i = 0, 1, \ldots, (n/2) - 1)$ restarting at intervals

$$j\left(\frac{n}{2}\right) + i, \quad j = 0, 1, \ldots. \tag{52}$$

For the second video, the scheduled is time-shifted by one repeating interval with multicast channel i, restarting at intervals

$$j\left(\frac{n}{2}\right) + \left((i+3) \bmod \left(\frac{n}{2}\right)\right), \quad j = 0, 1, \ldots. \tag{53}$$

With this new complementary channel scheduling, it is easy to see that the number of channels occupied by the first video is $(n/(2M) + 1)$ in even numbered repeating intervals and $(n/(2M) + 2)$ in odd numbered repeating intervals. The reverse case applies to video two, occupying $(n/(2M) + 2)$ and $(n/(2M) + 1)$ channels during even- and odd-numbered repeating intervals, respectively. Hence, the combined channel requirement will simply be equal to

$$\left(\frac{n}{M} + 3\right) \tag{54}$$

for these two videos. Compared to the original channel requirement of $(n/M + 4)$ channels, we can free one more multicast channel for every pair of videos scheduled using this complementary channel scheduling technique.

To simplify the derivations, we assume that M is divisible by 2 (i.e., no orphan video that cannot be paired) and N ($N \geq 6$) is divisible by 4 (i.e., equal allocation of unicast and multicast channels). With these two assumptions, we can rewrite (24) as

$$\rho = \frac{\frac{\lambda}{2}\left(1 - \frac{\delta n}{L}\right)\left(\frac{L}{n} - \delta\right)}{N - Mn + M(0.5n - 1.5)}. \tag{55}$$

The near-optimal channel partition policy with over allocation will then become

$$n_{CCS} = \max\left(\left\langle \frac{L(N-1.5M)}{LM - \delta(N-1.5M)} \cdot \frac{1}{2} \right\rangle 2, 2\left(\frac{N}{M} - 1.5\right)\right). \tag{56}$$

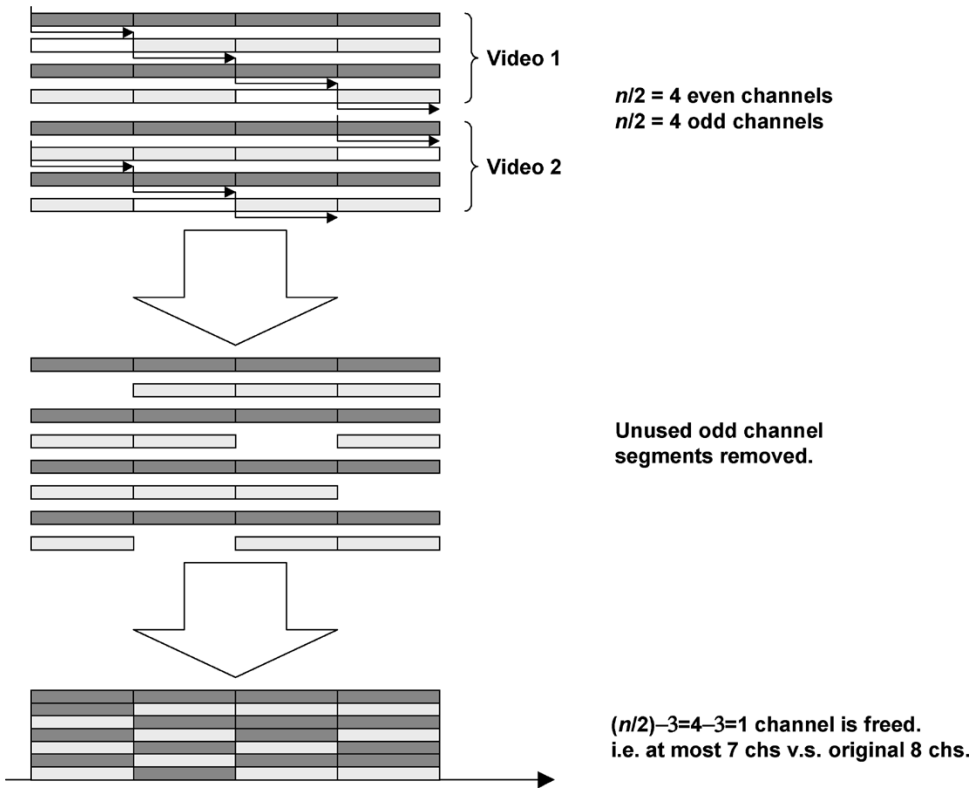


Fig. 7. Complementary channel scheduling (eight multicast channels, two videos).

TABLE II
LIST OF SYSTEM PARAMETERS

Parameters	Symbol	Value
Total number of available channels	N	200
Number of multicast channels	N_M	Computed
Number of unicast channels	N_U	Computed
Number of videos	M	10
Length of each video	L	120 minutes
Skewness for Zipf-distributed video popularity profile	θ	0.271

Similarly, we need to modify the latency function to account for the reduced channel requirement. Let $W_{CCS}(N_U, N_M)$ be the latency function for UVoD with channel folding and complementary channel scheduling. Assuming there are an even number of videos in the system, then it can be computed from

$$W_{CCS}(N_U, N_M) = W(N_U + \max(\lfloor 0.5N_M \rfloor - 1.5, 0), N_M) \tag{57}$$

where N_U and N_M are determined according to (56).

VIII. PERFORMANCE EVALUATION

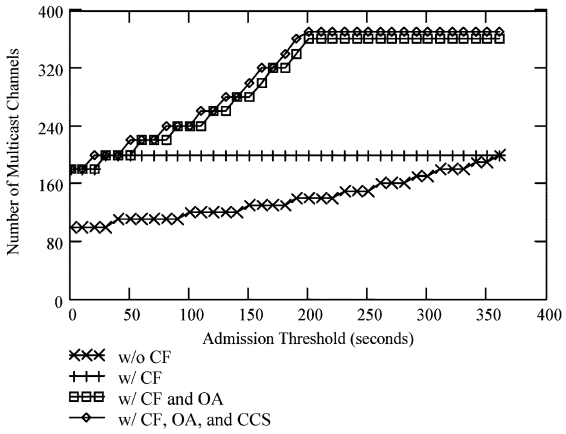
In this section, we evaluate the performance gain and tradeoff of applying channel folding to the UVoD architecture. In particular, we investigate the impact of channel folding on the channel partition policy, the client buffer requirement, latency, and scalability using the performance model derived in Section V. The system parameters employed are summarized in Table II.

A. Impact on Channel Partition

Fig. 8 plots the near-optimal number of multicast channels versus the admission threshold for a system with 200 channels and 10 videos. The admission threshold represents the load of the system as the average latency is equal to half of the admission threshold. The curves plot the optimal number of multicast channels that result in the lowest latency. The key observation is that the near-optimal channel partition policy differs significantly when channel folding is applied. In particular, channel folding leads to a much larger portion of available channels to be allocated for multicast transmissions. Second, with over allocation of multicast channels, the near-optimal point shifted to well over 200 channels. These results have a significant impact to the client buffer requirement, which we will discuss next.

B. Client Buffer Requirement

The primary tradeoff of channel folding is increased client buffer requirement. In particular, according to (47), a client needs a buffer large enough to store up to $2T_R$ seconds' worth



(CF: Channel Folding; OA: Over Allocation; CCS: Complementary Channel Scheduling)

Fig. 8. Near-optimal channel partition versus admission threshold for 200 channels and 10 videos.

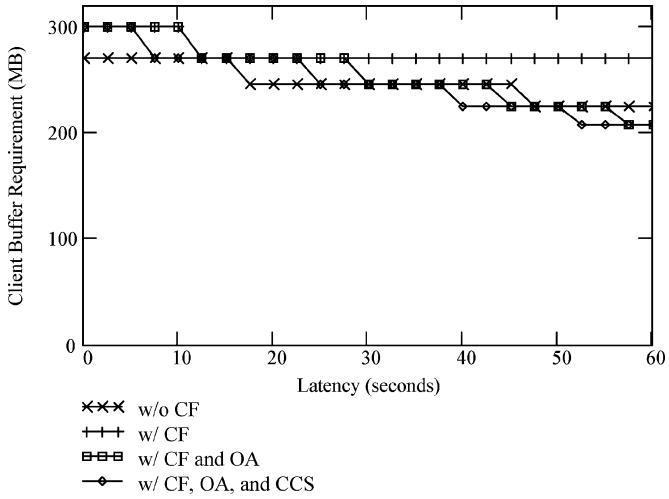


Fig. 9. Client buffer requirement versus latency ($R_v = 3$ Mb/s, 10 videos, 200 channels).

of video data, while the same for the case without channel folding is only T_R seconds [35]. This suggests that channel folding requires double the amount of client buffer to achieve its performance gain.

Remarkably, this is not the case, as shown in Fig. 9, which plots the client buffer requirements versus latency. While channel folding does require more buffers at small latencies (e.g., 10 s and below), the increase is far less than 100%. For example, at a latency of 10 s, the client buffer requirement without channel folding is 270 MB, while the same for the case with channel folding is only 300 MB. The buffer requirement is increased by only 11% instead of 100% as the equations suggested.

This counterintuitive result is a consequence of channel folding's impact on the channel partition policy. As discussed in Section VIII-A, the use of channel folding leads to far more channels being allocated for multicast transmissions (N_M multicast channels), and thus significantly reducing the multicast cycle length T_R [see (1)]. For example, T_R is equal to 720 s without channel folding but is reduced to 400 s with channel folding. Coupled with the fact that the client buffer requirement

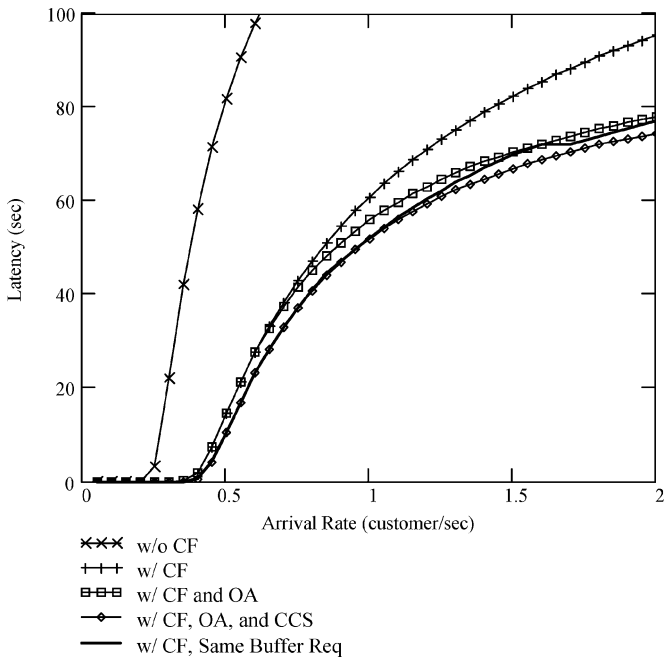


Fig. 10. Latency comparisons ($R_v = 3$ Mb/s, 10 videos, 200 channels).

is proportional to T_R, this reduction in T_R offset the increase in client buffer requirement.

C. Latency Comparisons

For the end users, latency is one of the primary measures in evaluating the quality of a VoD service. We plot the latency versus arrival rate in Fig. 10. There are two key observations from this result. First, at low arrival rate, e.g., less than 0.2 customer/s, both cases achieve zero latency and hence channel folding offers no advantage latency-wise. When the arrival rate is increased the performance gains due to channel folding is very substantial. The additional performance gains of over allocation and complementary channel scheduling are also more significant at high arrival rates (e.g., 1 customer/s or higher).

Second, note that the curve labeled "w/CF, Same Buffer Req" is obtained by forcing the buffer requirement of channel folding to be the same as the case without channel folding. This is achieved by changing the channel partition policy to assign exactly double the number computed from the original UVoD channel partition policy in (17). The results show that the impact on the latency is relatively small, implying that one can achieve significant performance gain even without any tradeoff in increased buffer requirement.

D. System Capacity and Scalability

In this section, we evaluate the capacity of a UVoD system with and without channel folding under a given latency constraint. Specifically, we determine the system capacity, denoted by C_{CF} and C for the case with and without channel folding, respectively, by increasing the arrival rate until the given latency constraint is reached:

$$C_{CF} = \max\{\lambda | W_{CF}(\lambda) \leq \mu, \forall \lambda \geq 0\} \quad (58)$$

$$C = \max\{\lambda | W(\lambda) \leq \mu, \forall \lambda \geq 0\} \quad (59)$$

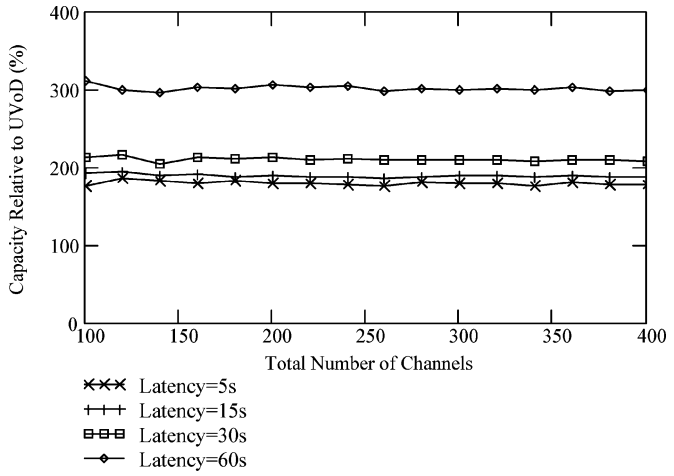


Fig. 11. Capacity gain under various system scale and latency constraints (each video is allocated 20 channels).

where λ is the arrival rate, μ is the latency constraint, and $W_{CF}(\lambda)$ and $W(\lambda)$ are the latency functions to compute the latency at the arrival rate of λ.

To facilitate comparison, we compute the capacity relative to UVoD without channel folding

$$\frac{C_{CF}}{C} \times 100\% \tag{60}$$

and plot the results versus system scale (i.e., total number of available channels) in Fig. 11. The primary observation is that the gain in capacity due to channel folding is consistent across system scales. For small latency values (< 30 s), channel folding can increase UVoD's capacity by around 100%. For larger latency value (e.g., 60 s), the improvement jumps to 200%. These results suggest that channel folding can be applied to systems of all scales with consistent performance improvement.

IX. Conclusion

In this study, we investigated a channel folding algorithm to improve the efficiency of a multicast video distribution architecture—the UVoD architecture. We derived a performance model for the improved architecture and obtained the latency, near-optimal channel partition policy, and client buffer requirement. Numerical results showed that channel folding can double the capacity of a UVoD system with only a small client-side buffering overhead. This channel folding algorithm is not limited to UVoD, and thus could be applied to other multicast video distribution architectures. Further investigations will be needed to explore the potential of this new tool in the arsenal.

References

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

[2] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel Allocation under Batching and VCR Control in Movie-on-Demand Servers,", Yorktown Heights, NY, IBM Res. Rep. RC19588, 1994.

[3] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *ACM Multimedia Syst.*, no. 4, pp. 112–121, 1996.

[4] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proc. Int. Conf. Multimedia Syst.*, Hiroshima, Japan, Jun. 1996, pp. 253–258.

[5] H. Shachnai and P. S. Yu, "Exploring wait tolerance in effective batching for video-on-demand scheduling," in *Proc. 8th Israeli Conf. Computer Syst. Software Eng.*, Jun. 1997, pp. 67–76.

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

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

[8] H. C. De-Bey, "Program Transmission Optimization," U.S. Patent 5 421 031, Mar. 30, 1995.

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

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

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

[12] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE J. Select. Areas Commun.*, vol. 14, no. 6, pp. 1110–1122, Aug. 1996.

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

[14] L. S. Juhn and L. M. Tseng, "Harmonic broadcasting for video-on-demand service," *IEEE Trans. Broadcast.*, vol. 43, no. 9, pp. 268–271, Sep. 1997.

[15] ——, "Staircase data broadcasting and receiving scheme for hot video service," *IEEE Trans. Consum. Electron.*, vol. 43, no. 11, pp. 1110–1117, Nov. 1997.

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

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

[18] J. F. Pâris, S. W. Carter, and D. D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proc. 6th Int. Symp. Modeling, Anal. Simulation Computer Telecommun. Syst.*, Jul. 1998, pp. 127–132.

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

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

[21] Y. Birk and R. Mondri, "Tailored transmissions for efficient near-video-on-demand service," in *Proc. IEEE Int. Conf. Multimedia Computing Syst.*, Florence, Italy, Jun. 1999, pp. 226–231.

[22] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, no. 4, pp. 51–62, Apr. 1997.

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

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

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

[26] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective video-on-demand," in *Proc. IS&T/SPIE Conf. Multimedia Computing Netw.*, San Jose, CA, Jan. 2000, pp. 206–215.

[27] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "Reducing I/O demand in video-on-demand storage servers," in *Proc. ACM SIGMETRICS Joint Int. Conf. Measurement Modeling of Computer Syst.*, Ottawa, ON, Canada, May 1995, pp. 25–36.

[28] ——, "Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers," *ACM Multimedia Syst.*, vol. 4, no. 30, pp. 14–55, 1996.

[29] S. W. Lau, J. C. S. Lui, and L. Golubchik, "Merging video streams in a multimedia storage server: Complexity and heuristics," *ACM Multimedia Syst.*, vol. 6, no. 1, pp. 29–42, 1998.
[30] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal piggyback merging policies for video-on-demand systems," in *Proc. Int. Conf. Multimedia Syst.*, Jun. 1996, pp. 253–8.
[31] J. H. Oh, K. A. Hua, and K. Vu, "An adaptive hybrid technique for video multicast," in *Proc. Int. Conf. Computer Commun. Netw.*, Lafayette, LA, Oct. 1998, pp. 227–234.
[32] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE Int. Conf. Multimedia Computing Syst.*, vol. 2, Florence, Italy, Jun. 1999, pp. 117–121.
[33] L. Gao, Z. L. Zhang, and D. Towsley, "Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams," in *Proc. 7th ACM Int. Multimedia Conf.*, Orlando, FL, Nov. 1999, pp. 203–206.
[34] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (Mcache): An adaptive zero-delay video-on-demand service," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 440–56, Mar. 2001.
[35] J. Y. B. Lee, "On a unified architecture for video-on-demand services," *IEEE Trans. Multimedia*, vol. 4, no. 1, pp. 38–47, Mar. 2002.

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

He participated in the research and development of commercial video streaming systems from 1997 to 1998 and later joined the Department of Computer Science at the Hong Kong University of Science and Technology from 1998 to 1999. In 1999, he joined the Department of Information Engineering, Chinese University of Hong Kong, and established the Multimedia Communications Laboratory to conduct research in distributed multimedia systems, fault-tolerant systems, peer-to-peer systems, multicast communications, and Internet computing.