A Recursive Frequency-Splitting Scheme for Broadcasting Hot Videos in VOD Service

Yu-Chee Tseng, *Member, IEEE*, Ming-Hour Yang, and Chi-He Chang

***Abstract*—One way to broadcast a popular/hot video is to let multiple users share a few channels. The stress on the scarce channels can be alleviated without sacrificing viewer waiting time. One common approach is to partition the video into fixed-length segments, which are broadcast on several channels periodically. Two representative approaches are the Fast Broadcasting scheme and the PAGODA scheme, which can broadcast a video using k channels by having new viewers wait no longer than $\Theta(D/2^k)$ and $\Theta(D/5^{k/2})$ time, respectively, where D is the length of the video. In this paper, we propose a new scheme, called *Recursive Frequency Splitting (RFS)*, that significantly improves on existing schemes in terms of viewer waiting time. Some lower bounds on the viewers' waiting time are also developed.**

***Index Terms*—Broadband networks, broadcasting, cable TV, digital video broadcasting, scheduling, video-on-demand (VOD).**

I. Introduction

With the advancement of broadband networking technology and the growth of processor speed and disk capacity, video-on-demand (VOD) services have become possible [20], [22]. Offering such services is likely to be popular in local residential areas, and viable in metropolitan areas in the near future.

A VOD system is typically implemented by a client-server architecture supported by certain transport networks such as cable TV or satellite networks [5], [13], [27]. The simplest scheme is to dedicate a channel to each client [9], [21]. Many VCR-like functions may be provided (e.g., forward, rewind, pause, search, etc.). Since video is an isochronous medium, the video server has to reserve a sufficient amount of network bandwidth and input–output (I/O) bandwidth for each video stream before committing to a client's request [11]. Such systems may easily run out of channels because the growth in the number of channels can never keep up with the growth in the number of clients. This results in tremendous demand for computing power and communication bandwidths on the system.

Paper approved by M. R. Civanlar, the Editor for Image Processing of the IEEE Communications Society. Manuscript received February 12, 2001; revised December 31, 2001. The work of Y.-C. Tseng was supported by the Lee and MTI Center for Networking Research at National Chiao Tung University, Hisn-Chu, Taiwan, R.O.C., by the Ministry of Education under Contract 89-H-FA07-1-4 and Contract 89-E-FA04-1-4, and by the National Science Council, Taiwan, R.O.C. under Contract NSC90-2213-E009-154.

Y.-C. Tseng is with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsin-Chu, 30050, Taiwan, R.O.C. (e-mail: yctseng@csie.nctu.edu.tw).

M.-H. Yang and C.-H. Chang are with Department of Computer Science and Information Engineering, National Central University, Chung-Li 32054, Taiwan, R.O.C.

Publisher Item Identifier 10.1109/TCOMM.2002.801466.

To relieve the stress on the bandwidth and I/O demands, many alternatives have been proposed by sacrificing some VCR functions, or known as *near-VOD services*. The *batching* approach collects a group of requests that arrive close in time, and serves them all together when a channel is available [1], [7], [8]. A scheduling policy based on the arrival of requests is required to best utilize the channels. Two *patching* schemes [10], [14] are proposed on top of the batching approach to allow late-coming clients to join the service with some buffering space and server channel constraints. A stream-tapping technique is proposed in [3]. Adaptive batching schemes are proposed in [25] and [26]. A survey on different scheduling techniques in a near-VOD system can be found in [12].

In this paper, we consider the *broadcasting* approach, where the server uses multiple dedicated channels to broadcast a video cooperatively. Each channel is responsible for broadcasting some portion of the video. Each client follows some reception rule to grab data from appropriate channels to play the whole video continuously. The server's broadcasting activity is independent of the arrivals of requests. Such an approach is more appropriate for popular or hot videos that may interest many viewers at a certain period of time. According to [7] and [8], 80% of demands are on a few (10 or 20) very popular videos.

One important issue in the broadcasting approach is the *segment-scheduling problem*, which refers to how a video server partitions a video into segments and schedules these segments on the communication channels. To reduce the new viewers' waiting time, one naive approach is to periodically broadcast the video in several channels differentiated by time [6]; this can decrease the maximum waiting time linearly with respect to the number of channels used. To further reduce viewers' waiting time, many approaches that are based on partitioning the video into segments have been proposed. Here we categorize segmentations into two types, *vertical* and *horizontal*. Let b be the bandwidth required to transmit a video to a client sequentially through unicast. In vertical segmentation, the video is allowed to be partitioned into a number of segments along the time axis, such that each segment still needs a bandwidth b to transmit. In horizontal segmentation, the video is allowed to be partitioned along the bandwidth axis, such that each segment only needs a fraction of b to transmit. In the literature, solutions may be solely based on vertical segmentation, or based on a combination of vertical and horizontal segmentations. This classification is illustrated in Fig. 1.

Schemes based on vertical segmentation include [2], [15], [19], [23], [24], [28], and [29]. In [2] and [29], a *pyramid scheme* is proposed, which can reduce the maximum waiting time in an exponential ratio with respect to the number of channels used. In [15] and [19], a *Fast Broadcasting (FB) scheme* is proposed,

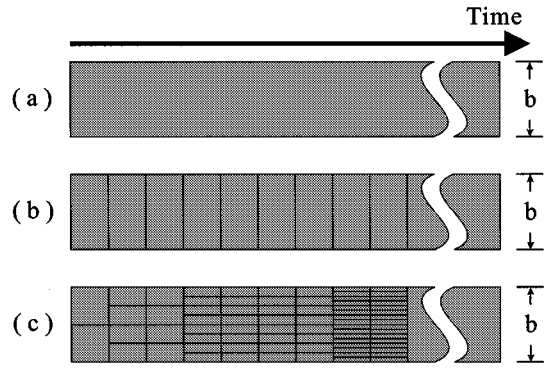


Fig. 1. Classification of video partitioning. (a) The original video. (b) Vertical segmentation. (c) Combination of vertical and horizontal segmentations.

which incurs $D/(2^k-1)$ waiting time when using k channels to broadcast a video of length D. A *PAGODA Broadcasting (PB) scheme* is proposed in [23] and [24], which further reduces the new viewers' waiting time to $D/(4(5^{(k/2)-1})-1)$ if k is even, and $D/(2(5^{\lfloor k/2 \rfloor})-1)$ if k is odd.

Schemes based on combinations of vertical and horizontal segmentations include [16]–[18]. In the *harmonic scheme* [16], [18], a video will be partitioned horizontally according to the harmonic series into segments of bandwidths $b/2, b/3, b/4, \ldots$. In the *staircase scheme* [17], a video will be partitioned horizontally into segments of bandwidths $b/2, b/2^2, b/2^3, \ldots$. (Fig. 1(c) is drawn based on the staircase scheme.) The staircase scheme can reduce clients' buffering requirement, while keeping viewers' waiting time the same as the FB scheme. The harmonic scheme can significantly reduce viewers' waiting time. In fact, this scheme is recently proved to be optimal with respect to viewers' waiting time given a particular transmission bandwidth [31]. However, horizontal segmentation is less practical than vertical segmentation due to the partitioning itself. An implementation of the FB scheme has been reported recently in [30].

In this paper, we propose a new broadcasting scheme based on the vertical segmentation model. Called a *Recursive Frequency-Splitting (RFS) scheme*, our approach is very systematic and simple in concept. Our scheme is based on a simple observation (see *Lemma 1*) on how frequently a video segment should be broadcast. The result significantly improves over the existing schemes [2], [15], [19], [24], [23], [29] that also use the same model. To our knowledge, this is the best vertical segmentation scheme in terms of viewer waiting time. To understand how close our result is to the optimal solution, we also develop some lower bounds on the viewer waiting time under the vertical segmentation model.

The rest of this paper is organized as follows. In Section II, we review the FB scheme and the PB scheme. Some important observations are made based on these schemes, and a more efficient scheme is proposed in Section III. Some analysis and simulation results on the performance of our scheme are presented in Section IV. Conclusions are drawn in Section V.

II. Reviews

To help understand the essence of the segment-scheduling problem, we review two representative vertical segmentation schemes in the literature.

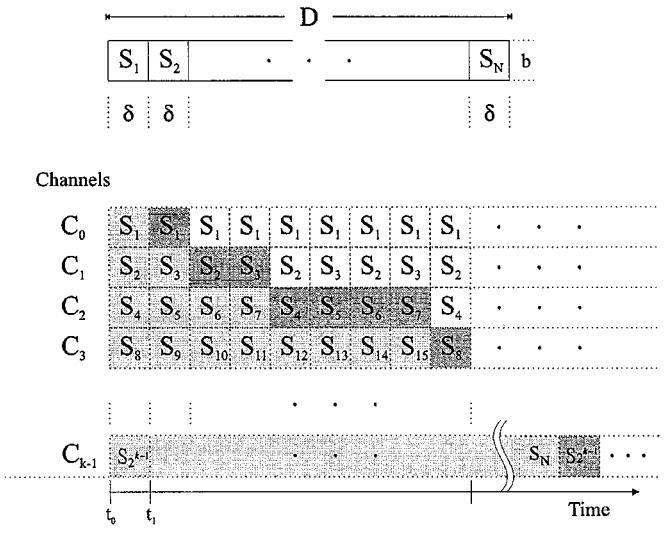


Fig. 2. FB scheme.

A. Fast Broadcasting (FB) Scheme

In the FB scheme [15], [19], we are given a video V of length D of bandwidth b. Since it is assumed that V is a popular video, providing each client a dedicated channel to view V is infeasible. To relieve the demand on channels, the FB scheme assigns a fixed number of k channels, $C_0, C_1, \ldots, C_{k-1}$, each of bandwidth b, to the video. These channels will work together to broadcast V with some special arrangement. The main goal is to reduce the waiting time incurred by new viewers to start watching the video.

The video server broadcasts V as follows.

1) Partition V vertically and evenly into n segments, $S_1, S_2, \ldots, S_n$, where $n = 2^k - 1$. That is, the concatenation $S_1 \circ S_2 \circ \cdots \circ S_n = V$ (we denote $\circ$ as the concatenation operator). The length of each segment is $\delta = D/n = D/(2^k - 1)$.
2) Divide each channel $C_i, i = 0, \ldots, k-1$ into time slots of length δ. On C_i, broadcast data segments $S_{2^i}, S_{2^i+1}, \ldots, S_{2^{i+1}-1}$ periodically and in that order. Note that the first segment S_{2^i} of each $C_i, i = 0, \ldots, k-1$ should be aligned in the same time slot.

An example is in Fig. 2. Channel C_0 broadcasts the first segment periodically, C_1 broadcasts the next two segments periodically, C_2 broadcasts the next four segments periodically, etc.

To view V, a client should monitor and receive data from all k channels according to the following rules.

1) To start the service, wait until the beginning of *any* new time slot.
2) Concurrently from each channel $C_i, i = 0, \ldots, k-1$, download 2^i consecutive segments starting from the first time slot.
3) Right at the moment when step 2 begins, start to consume the video $S_1 \circ S_2 \circ \cdots \circ S_n$.

Let us use an example to show how the FB scheme works. In Fig. 2, suppose the video server allocates $k = 4$ channels to V. So V will be partitioned evenly into $n = 2^4 - 1 = 15$ segments. For a client starting at time t_0 in Fig. 2, in the first time slot, it will receive segments S_1, S_2, S_4, S_8 from C_0, C_1, C_2, C_3, re-

spectively. During the first time slot, segment S_1 will be consumed, and the other premature segments S_2, S_4, S_8 will be buffered at the client's local storage for future use. In the second slot, the client will consume segment S_2 from its local storage. At the same time, segments S_3, S_5, S_9 from C_1, C_2, C_3, respectively, will be buffered. In the third time slot, the client will consume the S_3 from its local storage, and simultaneously buffer S_6 and S_{10} from C_2 and C_3, respectively. This will be repeated until the client has received $2^3 = 8$ data segments from C_3. At last, the client will finish watching the video at time $t_0 + n\delta = t_0 + 15\delta$. The reader should be able to derive similar results easily for viewers starting at other time slots.

In some special time slots, it is possible for a client to play the video without buffering. For instance, if a client starts at time t_1 of Fig. 2, it can continuously receive every required segment (the darker segments in the figure) just in time from one of the channels. However, this happens only once every 2^{k-1} time slots.

In summary, the FB scheme allows a client to start at the beginning of any time slot by ensuring that whenever a segment is needed to be consumed, either it has been buffered previously, or it is being broadcast just in time on one of the channels. We briefly outline the proof as follows. Suppose that a client begins to download S_1 at time t. Consider the 2^i segments $S_{2^i}, S_{2^{i+1}}, \ldots, S_{2^{i+1}-1}$, which are periodically broadcast on $C_i, i = 0, \ldots, k-1$. These segments will be downloaded by the client from C_i in the time interval $[t, t + 2^i\delta]$. However, these segments will be viewed by the client in the time interval $[t + (2^i - 1)\delta, t + (2^{i+1} - 1)\delta]$. There is only one slot of overlapping, i.e., $[t + (2^i - 1)\delta, t + 2^i\delta]$ between the above two time intervals. In this time slot, S_{2^i} is the segment to be played. It can be easily observed that S_{2^i} either has appeared on C_i previously, or is currently being broadcast on C_i in time. This concludes the proof.

What the FB scheme achieves is to shorten viewers' maximum waiting time with only a few channels. A client has to wait no longer than δ time to start viewing the video. The average waiting time is $\delta/2$. Since $\delta = D/(2^k - 1)$, a small increase in k can reduce the waiting time significantly. For instance, given a 120-minute video, with 5 channels, a viewer has to wait no more than $120/(2^5 - 1) < 4$ minutes to start the service, and with 6 channels, the maximum waiting time further reduces to $120/(2^6 - 1) < 2$ minutes.

One problem with the FB scheme is that the number of channels used on a video can not be changed according to the level of "hotness" of the video. In [28], an interesting scheme is proposed to enhance the FB scheme so that the number of channels used by a video can be dynamically adjusted on the fly without causing any interruption.

B. PAGODA Broadcasting (PB) Scheme

The PB scheme [24] can further reduce the users' waiting time. Still, we are given a video V of length D and k channels, $C_0, C_1, \ldots, C_{k-1}$. The video server uses the following rules to broadcast V.

1) Partition V evenly into n segments, $S_1, S_2, \ldots, S_n$, where $n = 4(5^{(k/2)-1}) - 1$ if k is even, and $n = 2(5^{\lfloor k/2 \rfloor}) - 1$ if k is odd. Also, divide each channel into time slots of fixed length $\delta = D/n$.
2) On C_0, broadcast segment S_1 periodically. For $j = 1, \ldots, \lfloor (k-1)/2 \rfloor$, periodically broadcast on channel C_{2j-1} the segments

$$S_z, S_{2z}, S_{z+1}, S_{2z+2}, \ldots, S_{\frac{3z}{2}-1}, S_{3z-2}, S_z, S_{2z+1}, S_{z+1}, S_{2z+3}, \ldots, S_{\frac{3z}{2}-1}, S_{3z-1}$$

and on channel C_{2j} the segments

$$S_{\frac{3z}{2}}, S_{3z}, S_{4z}, \ldots, S_{2z-1}, S_{4z-2}, S_{5z-2}, S_{\frac{3z}{2}}, S_{3z+1}, S_{4z+1}, \ldots, S_{2z-1}, S_{4z-1}, S_{5z-1}$$

where $z = 2(5^{j-1})$.
3) If k is even, then periodically broadcast on the last channel C_{k-1} the segments

$$S_z, S_{z+1}, \ldots S_{2z+1}$$

where $z = 2(5^{j-1})$.

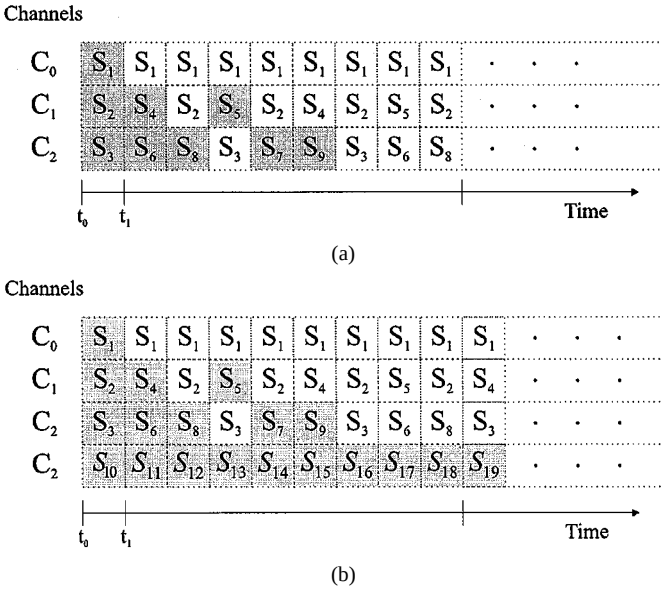


Fig. 3. The PB scheme. (a) $k = 3$ channels. (b) $k = 4$ channels.

Fig. 3 shows the PB scheme's scheduling for $k = 3$ and 4. The video will be partitioned into $2(5^{\lfloor 3/2 \rfloor}) - 1 = 9$ and $4(5^{(4/2)-1}) - 1 = 19$ segments, respectively. In the figure, we mark by gray when and where to grab the necessary segments for a client starting at the first time slot.

In summary, the PB scheme places segments in a more efficient and compact way than the FB scheme. The number of segments (n) will grow much faster as k increases than in the FB scheme. So the waiting time (i.e., D/n) can be significantly reduced. For instance, with 8 channels, a video will be partitioned into 499 segments by the PB scheme, and 254 segments by the FB scheme.

The New PAGODA Broadcasting (NPB) scheme [23] further improves on the PB scheme on the waiting time. However, a less regular arrangement pattern is used.

III. Our RFS Scheme

From the above reviews, we can see that both the FB and the PAGODA schemes partition the video into a number of fixed-length segments, each of which is scheduled to broadcast on one of the channels periodically. The maximum waiting time incurred by viewers is thus reflected by the length of a segment, or equivalently, the inverse of the number of segments. We thus formally define our problem as follows.

Definition 1: Given a video V of length D and a set of k channels $C_0, C_1, \ldots, C_{k-1}$, the *Fixed-Length Segment-Scheduling (FLSS) Problem* is to find a partition of V into n fixed-length segments $S_1, S_2, \ldots, S_n$, and to find a placement of these n segments on the k channels. Channels will be synchronously divided into time slots of length $\delta = D/n$. The placement should guarantee that for any viewer starting to play the video at the beginning of any time slot, each video segment will be received or has been received at the time slot when the viewer needs to consume the segment. The goal is to maximize n, or equivalently to minimize D/n.

The following lemma gives a necessary and sufficient condition to solve this problem.

Lemma 1: For any solution to the FLSS problem, each segment $S_i, 1 \le i \le n$, must be broadcast at least once on one of the k channels in every continuous i time slot.

Proof: Suppose that a viewer starts to play the video at time slot j. Then the viewer will consume segment S_i at time slot $j+i-1$. This implies that S_i must be broadcast on one of the channels at time slot $j+i-1$, or has been broadcast on one of the channels during slots $j, j+1, \ldots, j+i-2$. This proves that the condition given in the lemma is a sufficient condition. To see that this is also a necessary condition, simply observe that if S_i has not been sent in the aforementioned time slots, the viewer will experience an interruption at time slot $j+i-1$. □

The basic idea of our scheme is as follows. To satisfy the above lemma, we will schedule S_i to be broadcast on one of the channels periodically with a frequency no less than $1/i$. Suppose our scheme can accommodate n segments. Then, overall, our scheme needs to determine n sequences of periodical time slots, each of which appears in one of the k channels, and each of which can accommodate one of the n segments. The challenge is that when putting all these n sequences together, we must guarantee that in no time slot, more than one segment is scheduled to be broadcast on any of the channels simultaneously.

The above discussion introduces the concept of "periodical time slots," which can be regarded as an infinite sequence of time slots, each spaced by a fixed amount of time. The following definition shows how we represent such a concept.

Definition 2: A slot sequence $\mathrm{SS}(C_i, \eta, p)$ is an infinite sequence of time slots $[\eta, \eta+p, \eta+2p, \ldots]$ belonging to channel C_i, beginning at slot η, and repeating infinitely with a period of p slots, where C_i is one of the k channels, $\eta \ge 0$ is an integer, and $p \ge 1$ is an integer, $0 \le \eta \le p-1$.

Note that for ease of presentation, throughout the paper, we will count the time slots of a channel starting from 0, instead of 1. Several examples of slot sequences are shown in Fig. 4. Also, note that when $p = 1$, the time slots will be continuous, and it represents a complete channel (e.g., $C_i = \mathrm{SS}(C_i, 0, 1)$).

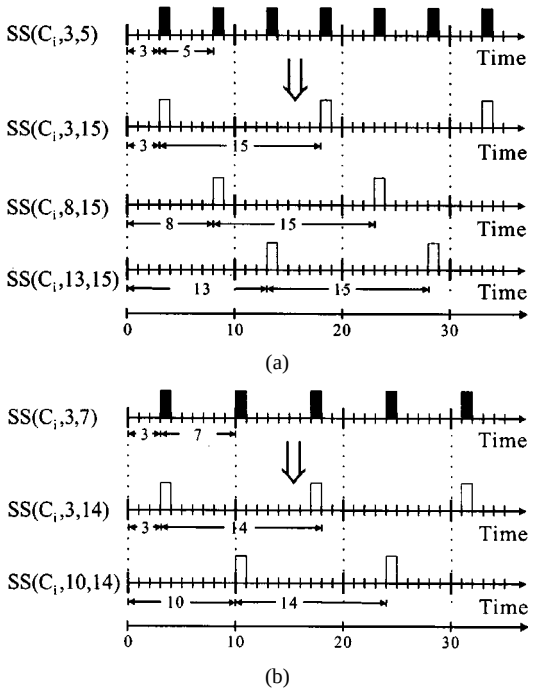


Fig. 4. (a) Splitting result after applying *Lemma 2* on $\mathrm{SS}(C_i, 3, 5)$ and S_{15}. (b) Splitting result after applying *Lemma 2* on $\mathrm{SS}(C_i, 3, 7)$ and S_{15}.

Our scheme is based on a concept called "frequency splitting." According to *Lemma 1*, we should allocate a slot sequence $\mathrm{SS}(C_i, \eta, p)$ for segment S_j, such that $p \le j$. It is desirable that the value of p be as close to j as possible, since a larger p means less waste in communication bandwidth. The best case is $p = j$. However, in case that $p < j$, the following lemma shows how to split the slot sequence into a number of subsequences to save bandwidth.

Lemma 2: Given a free slot sequence $\mathrm{SS}(C_i, \eta, p)$ and a video segment S_j, such that $p \le j$, one possible placement for S_j is to partition $\mathrm{SS}(C_i, \eta, p)$ into the following $\alpha = \lfloor j/p \rfloor$ slot sequences:

$$\mathrm{SS}(C_i, \eta, \alpha \cdot p), \mathrm{SS}(C_i, \eta+p, \alpha \cdot p), \mathrm{SS}(C_i, \eta+2p, \alpha \cdot p), \ldots, \mathrm{SS}(C_i, \eta+(\alpha-1)p, \alpha \cdot p)$$

assign one of the slot sequences to S_j, and keep the remaining $\alpha - 1$ slot sequences free.

Proof (Sketched): The period αp is the maximal period that is a multiple of p and can satisfy the condition in *Lemma 1*. □

Fig. 4(a) shows an example of applying this lemma to $\mathrm{SS}(C_i, 3, 5)$ and S_{15}. We partition $\mathrm{SS}(C_i, 3, 5)$ into $\alpha = \lfloor 15/5 \rfloor = 3$ subsequences $\mathrm{SS}(C_i, 3, 15)$, $\mathrm{SS}(C_i, 8, 15)$, and $\mathrm{SS}(C_i, 13, 15)$. We can assign one subsequence to S_{15}, and keep the remaining two free for future use. Fig. 4(b) shows another example when applying this lemma to $\mathrm{SS}(C_i, 3, 7)$ and S_{15}. $\mathrm{SS}(C_i, 3, 7)$ is partitioned into $\alpha = \lfloor 15/7 \rfloor = 2$ subsequences $\mathrm{SS}(C_i, 3, 14)$ and $\mathrm{SS}(C_i, 10, 14)$. In this case, assigning S_{15} to any of these sequences will waste $(15-14)/(15 \cdot 14)$ bandwidth of channel C_i.

A. RFS Scheme

Next, we present our RFS scheme. We are initially given a set of k channels $C_0, C_1, \ldots, C_{k-1}$, which is considered a pool of slot sequences. We will start the placement from S_1. For each segment to be placed, we will take one slot sequence from the pool and apply *Lemma 2* on it. After applying *Lemma 2*, some new subsequences may be generated and returned to the pool after the splitting. The is recursively repeated until the pool becomes empty. The outputs of the scheme are a value of n, and the assignment of one slot sequence for each segment $S_i, i = 1 \ldots n$.

1) Let POOL be a set of slot sequences:

$$\text{POOL} = \{\text{SS}(C_0, 0, 1), \text{SS}(C_1, 0, 1), \ldots, \text{SS}(C_{k-1}, 0, 1)\}.$$

   Intuitively, this is the set of free channels $C_0, C_1, \ldots, C_{k-1}$ that we are given initially.
2) Initialize $n = 1$.
3) Pick a slot sequence $\text{SS}(C_i, \eta, p) \in \text{POOL}$, such that $p \leq n$. If more than one sequence in POOL satisfies this condition, apply the following rules to pick one.
   a) The sequence(s) with the smallest value of $n \bmod p$ should be picked first.
   b) If step a) still renders more than one sequence, pick any one with the largest p.

   Then do the subtraction $\text{POOL} = \text{POOL} - \{\text{SS}(C_i, \eta, p)\}$.
4) Apply *Lemma 2* on sequence $\text{SS}(C_i, \eta, p)$ and segment S_n to split $\text{SS}(C_i, \eta, p)$ into $\alpha = \lfloor n/p \rfloor$ slot sequences.
   a) Assign $\text{SS}(C_i, \eta, \alpha \cdot p)$ to S_n.
   b) Do the union $\text{POOL} = \text{POOL} \cup \{\text{SS}(C_i, \eta + jp, \alpha \cdot p), j = 1 \ldots \alpha - 1\}$.
5) If there exists a slot sequence $\text{SS}(C_i, \eta, p) \in \text{POOL}$ such that $p \leq n + 1$, then increase n by 1 and go to step 3 to schedule the next segment; otherwise, terminate this procedure and output the value of n.

In the above procedure, we try to increase the value of n repeatedly. In step 3, we try to pick one slot sequence in POOL that can be used for S_n. Step 3 a) is a heuristic for reducing the waste of bandwidth when performing the assignment in step 4. Step 3 b) is a heuristic for leaving more flexibility in subsequent assignments. Step 4 performs the splitting. Finally, step 5 checks whether we can proceed to accommodate the next segment. If so, n is increased by 1, and we loop back to step 3. Also note that step 5 is written in a way for ease of understanding. The condition, "there exists a slot sequence $\text{SS}(C_i, \eta, p) \in \text{POOL}$ such that $p \leq n+1$" in step 5 can be reduced to, "if $\text{POOL} \neq \emptyset$." The reason is that when doing splitting, we never generate a subsequence which has a period larger than the current value of n.

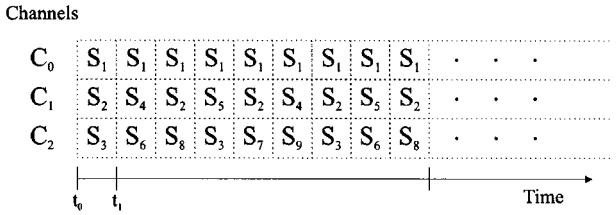
For example, let us consider $k = 3$. A total of nine iterations will be executed, as shown in Fig. 5(a). The final arrangement is shown in Fig. 5(b).

B. m-RFS

In the above RFS, we try to arrange one segment in each iteration. In the following, we propose a modification of RFS called m-RFS, which tries to arrange m segments in each iteration,

Iteration (n)	POOL
n = 1	~~SS(C_1,0,1)~~, SS(C_2,0,1)*, SS(C_3,0,1)
n = 2	~~SS(C_2,0,2)~~, SS(C_2,1,2), SS(C_3,0,1)*
n = 3	SS(C_2,1,2)*, ~~SS(C_3,0,3)~~, SS(C_3,1,3), SS(C_3,2,3)
n = 4	~~SS(C_2,1,4)~~, SS(C_2,3,4), SS(C_3,1,3), SS(C_3,2,3)
n = 5	~~SS(C_2,3,4)~~, SS(C_3,1,3)*, SS(C_3,2,3)
n = 6	~~SS(C_3,1,6)~~, SS(C_3,4,6), SS(C_3,2,3)
n = 7	~~SS(C_3,4,6)~~, SS(C_3,2,3)*
n = 8	~~SS(C_3,2,6)~~, SS(C_3,5,6)
n = 9	~~SS(C_3,5,6)~~

(a)



(b)

Fig. 5. Running our RFS scheme given $k = 3$ channels. (a) Result of POOL after each application of *Lemma 2*. (b) Final placement. In (a), sequences that are stricken out are assigned to segments, and sequences with a superscript * are picked for splitting by *Lemma 2*.

where m is any positive integer. Specifically, in each iteration, m segments $S_{mi+1}, S_{mi+2}, \ldots, S_{m(i+1)}, i \geq 0$ will be considered. We then schedule these segments one by one using *Lemma 2* in different orders (there are $m!$ such permutations). Among all possible choices, we then choose the one that has the minimal waste of bandwidth. This is repeated until it is impossible to schedule m segments at a time, and then we go back to RFS and schedule one segment at a time, until no more segments can be scheduled. Intuitively, by different permutations, we hope to find a better solution than RFS. When $m = 1$, this degenerates to RFS, and when $m = \infty$, this becomes a brute force exhausted search.

For example, when there are $k = 4$ channels, 4-RFS can schedule as many as 26 segments, as opposed to 25 segments by RFS. Figs. 6(a) and (b) show the broadcasting sequences found by 4-RFS when $k = 4$ and $k = 5$, respectively.

IV. Analysis and Comparison

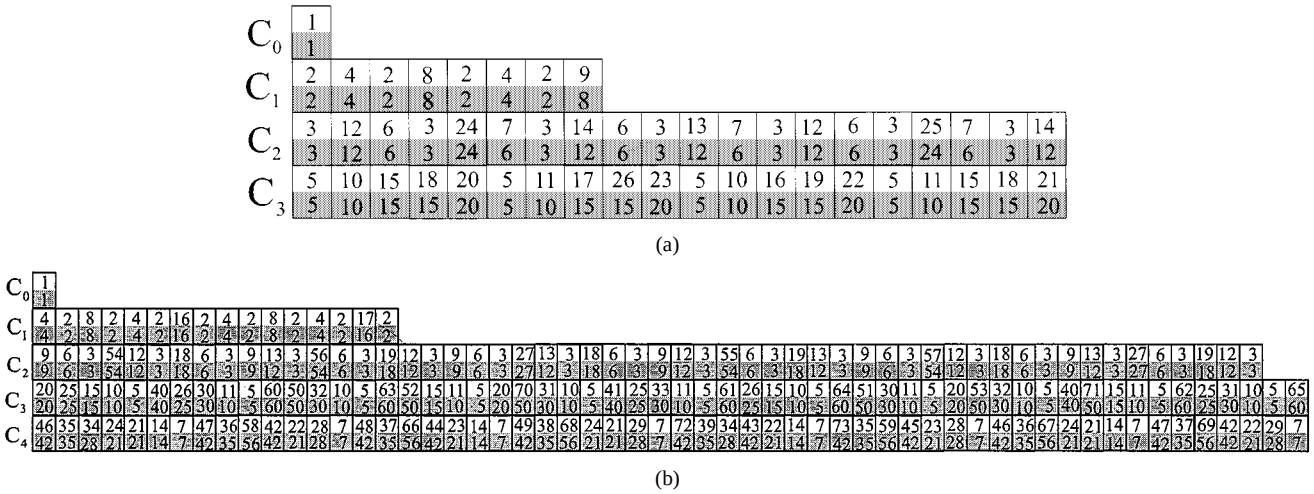
In this section, we propose some upper bounds on the value of n for the FLSS problem, and then compare the value of n found by our RFS scheme against the upper bounds and those found by existing schemes.

Theorem 1: For any solution to the FLSS problem, given k channels, an upper bound on the value of n must satisfy

$$\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n} \leq k < \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n} + \frac{1}{n+1}.$$

Proof: According to *Lemma 1*, segment S_i must be broadcast at least once in every continuous i time slot. Thus, S_i will consume at least $1/i$ bandwidth of a channel. Summing this over all segments gives this upper bound. □

The above upper bound is applicable to *any* solution to the FLSS problem. However, it does not impose that a segment al-



(a)

(b)

Fig. 6. Placement of 26 and 73 segments found by 4-RFS given (a) $k = 4$ and (b) $k = 5$ channels, respectively. (On each channel, for each time slot, two elements are shown. The first element on the top is the segment number, while the second element on the bottom with dark background is the period in which the corresponding segment appears.)

ways be broadcast on a fixed channel. That is, it is possible that the broadcast scheduling of a segment switches from one channel to another channel when collision occurs. If we do not allow such flexibility, the following derivation gives a tighter bound that applies to any solution that *broadcasts each S_i periodically in a fixed channel*. Note that this condition is applicable to all existing schemes [15], [19], [23], [24] and our scheme. This assumption also makes implementation easier.

Lemma 3: It is impossible to find two disjoint slot sequences $\mathrm{SS}(C_i, \delta_1, p_1)$ and $\mathrm{SS}(C_i, \delta_2, p_2)$, such that $\gcd(p_1, p_2) = 1$.

Proof: It is obvious that this lemma is true if $p_1 = 1$ or $p_2 = 1$. So we assume that $p_1, p_2 \geq 2$. We need to prove that for any integers a and b, $\delta_1 + a \times p_1 \neq \delta_2 + b \times q_2$, i.e., $\mathrm{SS}(C_i, \delta_1, p_1)$ and $\mathrm{SS}(C_i, \delta_2, p_2)$ share no common time slot. Since p_1 and p_2 are relatively prime, according to the Little Fermat's Theorem, there exist two integers c and d, such that $c \times p_1 + d \times p_2 = 1$. Multiplying both sides by $\delta_2 - \delta_1$, we have

$$c \times p_1 \times (\delta_2 - \delta_1) + d \times p_2 \times (\delta_2 - \delta_1) = \delta_2 - \delta_1.$$

Letting $a = c \times (\delta_2 - \delta_1)$ and $b = d \times (\delta_1 - \delta_2)$, we reach a contradiction

$$\begin{aligned} \delta_1 + a \times p_1 &= \delta_1 + c \times p_1 \times (\delta_2 - \delta_1) \\ &= \delta_1 + \delta_2 - \delta_1 - d \times p_2 \times (\delta_2 - \delta_1) \\ &= \delta_2 + b \times p_2. \end{aligned}$$

□

Based on the above lemma, we can derive a tighter upper bound on n. Intuitively, we consider the waste of bandwidth when broadcasting segment S_i by slot sequence $\mathrm{SS}(C_i, \eta, p)$, such that $p < i$. A fraction $(i - p)/(i \times p)$ of the bandwidth of a channel are wasted in this case. The following theorem will apply this concept to prime numbers. For ease of presentation, let $p(i)$ be the ith prime number (i.e., $p(1) = 2, p(2) = 3, p(3) = 4$, etc.).

Theorem 2: For any solution to the FLSS problem that uses k channels and that satisfies the condition that each segment is periodically broadcast in a fixed channel, an upper bound on the value of n must satisfy

$$\sum_{i=1}^{n} \frac{1}{f(i)} \leq k < \sum_{i=1}^{n+1} \frac{1}{f(i)}$$

where

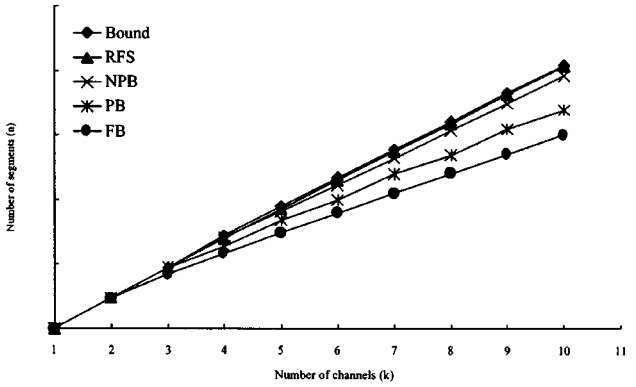
$$f(i) = \begin{cases} i - 1, & \text{if } i \text{ is a prime number and } i \geq p(k) \\ i, & \text{otherwise.} \end{cases}$$

Proof: Similar to the philosophy in the proof of *Theorem 1*, we will calculate an upper bound on n based on the bandwidth required in a scheduling. We assume that there is a perfect solution, such that for any S_i such that i is a composite number, S_i can be broadcast with a period i. So a fraction $1/i$ of the bandwidth of a channel is required for each of these segments.

Now consider the placement of segments S_i, such that i is a prime number. According to *Lemma 3*, no two segments S_i and S_j, such that i and j are primes, can be broadcast in the same channel with periods i and j, respectively, since $\gcd(i, j) = 1$. So one of S_i and S_j must be broadcast with a smaller period. Now consider the use of the k channels in accommodating segments S_i, such that i is a prime. One of the channels must be used to broadcast S_1. The other $k - 1$ channels can each accommodate one S_i, such that i is a prime. Based on the "waste of bandwidth" concept that we pointed out earlier, under the best situation we may place each of the $k - 1$ segments $S_{p(1)}, S_{p(2)}, \ldots, S_{p(k-1)}$ in a separate channel by broadcasting them with perfect periods $p(1), p(2), \ldots, p(k-1)$, respectively. The other segments $S_{p(k)}, S_{p(k+1)}, S_{p(k+2)}, \ldots$ will each be forced to broadcast with a smaller period less than its segment number no matter on which channels they are broadcast (otherwise, conflicts will occur). In the best case, these segments may be broadcast with periods $p(k)-1, p(k+1)-1, p(k+2)-1, \ldots$. This will lead to the least waste of bandwidth; if one tries to broadcast any S_i, such that i is a prime and $i \geq p(k)$ with a perfect period i, then one of the segments $S_{p(1)}, S_{p(2)}, \ldots, S_{p(k-1)}$ will be

	1	2	3	4	5	6	7	8	9	10
Bound	1	3	9	28	80	220	604	1650	4501	12260
4-RFS	1	3	9	26	73	201	565	1523	4284	11638
RFS	1	3	9	25	73	201	565	1522	4284	11637
NPB	1	3	9	26	66	172	442	1183	3092	8285
PB	1	3	9	19	49	99	249	499	1249	2499
FB	1	3	7	15	31	63	127	255	511	1023

(a)



(b)

Fig. 7. The maximal numbers of segments, n, offered by different schemes.

forced to be broadcast with a smaller period and the waste can only become larger.

Similar to the philosophy in *Theorem 1*, summing all bandwidths necessitated by all segments together gives this upper bound on n. □

To understand how well our RFS scheme performs, we have calculated the values of n offered by the FB, PB, NPB, RFS, and m-RFS schemes given different numbers of channels. The result is shown in Fig. 7. The upper bound is obtained from *Theorem 2*. From Fig. 7(a), we see that our RFS scheme outperforms all other schemes except $k = 4$, as compared to the NPB scheme. By using m-RFS with $m = 4$, this problem can be conquered, but there is only very slight improvement over RFS. As can be seen from our experiments, in the range of $m \leq 10$, only with $m = 4, 8$, and 10 can 4-RFS outperform RFS. So using RFS is quite efficient and simple. In fact, from Fig. 7(b), which is drawn in a logarithmic scale, we see that our RFS scheme performs asymptotically quite close to the upper bound.

The inverse of n offered by each scheme reflects the average waiting time for a new viewer to start his/her VOD service. Fig. 8 compares the average waiting time of different schemes.

The schemes discussed above all require buffering prematured segments at the client side. We do not have a close formula for the required buffering space of our RFS and m-RFS schemes. So we wrote a simulator to broadcast a 120-minute movie. We exhaustively searched all possibilities to find the maximum buffering space required at different numbers of channels. The comparison of maximum buffering spaces required by FB, seamless FB (with seamless channel transition capability [28] by setting $\alpha = 2$), RFS, 4-RFS, and staircase broadcasting [17]) is in Table I. Surprisingly, in addition to smaller waiting time, our schemes have a slightly lower buffering space requirement than that of FB and seamless FB. The staircase scheme incurs significantly less buffering space, however, this scheme does not fit into the category

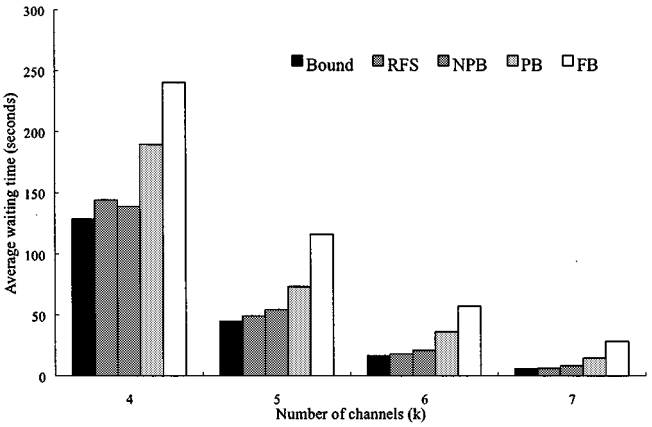


Fig. 8. Comparison of average waiting time incurred on new viewers at different numbers of channels ($D = 120$ min).

TABLE I
COMPARISON OF MAXIMUM BUFFERING SPACES REQUIRED BY FB, SEAMLESS-FB, RFS, 4-RFS, AND STAIRCASE (SB) IN THE NUMBER OF MINUTES OF A 120-MINUTE VIDEO USING k CHANNELS

k	2	3	4	5	6	7	8	9
FB	40.00	51.42	56.00	58.06	59.04	59.52	59.76	59.88
seamless FB	40.00	60.00	70.00	75.00	77.50	78.75	79.37	79.68
RFS	40.00	53.33	52.80	52.60	51.90	49.49	49.01	49.00
4-RFS	40.00	53.33	50.77	52.60	51.90	49.49	49.00	49.0
SB	20.00	25.71	28.00	29.03	29.52	29.76	29.88	29.94

of fixed-length segmentation approach since segments in the staircase scheme are partitioned in both vertical and horizontal directions and have different sizes and lengths.

V. CONCLUSION

The video broadcasting service is already popular in cable TV systems. Asynchronous video service is likely to grow quickly when the network infrastructure is ready. In this paper, we have proposed a new scheme for scheduling video segments on multiple channels to reduce viewers' waiting time. The result is shown to be more efficient than the best known schemes. Future research could be directed toward finding a more efficient scheme that can further reduce the viewers' waiting time, or toward deriving a tighter bound than the ones derived in this paper. The requirement of buffering space is obtained experimentally, but not analytically, and thus deserves further investigation. While most existing results adopt fixed scheduling, a recent work [4] that may deserve our attention proposes to use a reactive scheduling that can adapt to the popularity of a video.

ACKNOWLEDGMENT

The authors would like to thank Dr. J.-F. Paris for providing some experimental data on the New PAGODA scheme.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, June 1996, pp. 253–258.

[2] ——, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, June 1996, pp. 118–126.
[3] S. W. Carter and D. D. E. Long, "Improving video-on-demand server efficiency through stream tapping," in *Proc. Int. Conf. Computer Communications and Networks*, 1997, pp. 200–207.
[4] S. W. Carter, J. F. Paris, and D. D. E. Long, "A dynamic heuristic broadcasting protocol for video-on-demand," in *Proc. Int. Conf. Distributed Computing Systems*, 2001, pp. 657–664.
[5] Y. H. Chang, D. Coggins, D. Pitt, and D. Skellern, "An open-system approach to video on demand," *IEEE Commun. Mag.*, vol. 32, pp. 68–80, May 1994.
[6] T. Chiueh and C. Lu, "A periodic broadcasting approach to video-on-demand service," *Int. Soc. Optical Eng.*, vol. 2615, pp. 162–169, Oct. 1995.
[7] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. ACM Multimedia*, Florence, Italy, 1994, pp. 15–23.
[8] ——, "Dynamic batching policies for an on-demand video server," *Multimedia Syst.*, vol. 4, no. 3, pp. 112–121, June 1996.
[9] D. Deloddere, W. Verbiest, and H. Verhille, "Interactive video on demand," *IEEE Commun. Mag.*, vol. 32, pp. 82–88, May 1994.
[10] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," *IEEE Multimedia*, pp. 117–121, June 1999.
[11] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popolar videos," in *Proc. Int. Workshop Network and Operaing Syst. Support for Digital Audio and Video*, Aug. 1998, pp. 317–329.
[12] D. Ghose and H. J. Kim, "Scheduling video streams in video-on-demand systems: A Survey," *Multimedia Tools and Applicat.*, vol. 11, pp. 167–195, June 2000.
[13] W. Hodges, S. Mabon, and J. T. Powers Jr., "Video on demand: Architecture, systems, and applications," *SMPTE J., Building an Infrastructure for Managing Compressed Video Systems*, pp. 791–803, Sept. 1993.
[14] K. A. Hua, Y. Cai, and S. sheu, "Patching: A multicast technique for true video-on-demand services," *ACM Multimedia*, pp. 191–200, Sept. 1998.
[15] L.-S. Juhn and L.-M. Tseng, "Fast broadcasting for hot video access," *Real-Time Comput. Syst. and Applicat.*, pp. 237–243, Oct. 1997.
[16] ——, "Harmonic broadcasting for video-on-demand service," *IEEE Trans. Broadcast.*, vol. 43, pp. 268–271, Sept. 1997.
[17] ——, "Staircase data broadcasting and receiving scheme for hot video service," *IEEE Trans. Consumer Electron.*, vol. 43, pp. 1110–1117, Nov. 1997.
[18] ——, "Enhanced harmonic data broadcasting and receiving scheme for popular video service," *IEEE Trans. Consumer Electron.*, vol. 44, pp. 343–346, May 1998.
[19] ——, "Fast data broadcasting and receiving scheme for popular video service," *IEEE Trans. Broadcast.*, vol. 44, pp. 100–105, Mar. 1998.
[20] T. L. Kunii *et al.*, "Issues in storage and retrieval of multimedia data," *Multimedia Syst.*, vol. 3, no. 5, pp. 298–304, 1995.
[21] T. D. C. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia*, vol. 1, pp. 14–24, Mar. 1994.
[22] B. Ozden, R. Rastogi, and A. Silberschatz, "On the design of a low cost video-on-demand storage system," *Multimedia Syst.*, vol. 4, no. 1, pp. 40–54, 1996.
[23] J.-F. Paris, "A simple low-bandwidth broadcasting protocol for video-on-demand," in *Proc. Int. Conf. Computer Communication and Network*, 1999, pp. 118–123.
[24] J.-F. Paris, S.-W. Carter, and D.-D. Long, "A hybrid broadcasting protocol for video on demand," in *Proc. Multimedia Computing and Networking Conf.*, 1999, pp. 317–326.
[25] W. F. Poon and K. T. Lo, "New batching policy for providing true video-on-demand (T-VoD) in multicast system," in *Proc. IEEE ICC*, vol. 2, 1999, pp. 983–987.
[26] W. F. Poon, K. T. Lo, and J. Feng, "Adaptive batching scheme for multicast video-on-demand systems," *IEEE Trans. Broadcast.*, vol. 47, pp. 66–70, Mar. 2001.
[27] W. D. Sincoskie, "System architecture for a large scale video on demand service," *Computer Networks and ISDN Syst.*, vol. 22, pp. 565–570, 1991.
[28] Y.-C. Tseng, C.-M. Hsieh, M.-H. Yang, W.-H. Liao, and J.-P. Sheu, "Data broadcasting and seamless channel transition for highly-demanded videos," in *Proc. INFOCOM*, Mar. 2000, pp. 727–736.
[29] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *IEEE Multimedia*, pp. 197–208, Aug. 1996.
[30] Z.-Y. Yang, "The telepresentation system over Internet with latecomers support," Ph.D. Dissertation, National Central Univ., Chung-Li, Taiwan, R.O.C., July 2000.
[31] Z.-Y. Yang, L.-S. Juhn, and L.-M. Tseng, "On optimal broadcasting scheme for popular video service," *IEEE Trans. Broadcast.*, vol. 45, pp. 318–322, Sept. 1999.

Yu-Chee Tseng (S'91–M'95) received the B.S. degree from the National Taiwan University in 1985, and the M.S. degree from the National Tsing-Hua University in 1987, Hsin-Chu, Taiwan, both in computer science. He received the Ph.D. degree in computer and information science from Ohio State University, Columbus, OH, in 1994.

He was an engineer for D-LINK Inc., Hsin-Chu, Taiwan, in 1990. From 1994 to 1996, he was an Associate Professor in the Department of Computer Science, Chung-Hua University, Hsin-Chu, Taiwan. He joined the Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan, in 1996, and has been a Full Professor since 1999. In 2000, he joined the Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsin-Chu, Taiwan, as a Full Professor. His research interests include wireless communication, network security, parallel and distributed computing, and computer architecture.

Dr. Tseng serves as a Guest Editor in the special issue of "Advances in Mobile and Wireless Systems" in *ACM Wireless Networks*, and in the special issue of "Wireless Internet" of the IEEE TRANSACTIONS ON COMPUTERS. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Ming-Hour Yang received the M.S. degree in electronic engineering from the Chung-Hua University, Hsin-Chu, Taiwan, in 1996, and the Ph.D. degree in computer science and information engineering from the National Central University, Chung-Li, Taiwan, in 2001. He is currently a Research Fellow of National Strategic Studies Institute at National Defense University, Taipei, Taiwan.

His research interests include video on demand, parallel and distributed computing, fault tolerance, mobile computing, and wireless networks.

Chi-He Chang received the B.S. degree in computer science from the Chung Yuan Christian University, Chung-Li, Taiwan, in 1998, and the M.S. degree in computer science from National Central University, Chung-Li, Taiwan, in 2000.

His interests include video on demand and interactive multimedia.