A Fast Video-on-Demand Broadcasting Scheme for Popular Videos

Jang-Ping Sheu, Han-Lih Wang, Chi-He Chang, and Yu-Chee Tseng

***Abstract*—Broadcasting is a prospective approach to support near video-on-demand services with light communication overhead. By letting clients share channels, such approaches involve partitioning a video into segments and repeatedly broadcasting these segments in multiple channels. An early paper proposed a broadcasting scheme called RFS (*recursive frequency splitting*), which can significantly reduce clients' waiting time. While efficient, RFS suffers from a high computational complexity of $O(n \log n)$, where n is the number of segments of the video, which is typically very large. This paper proposes an efficient segmentation scheme, which can significantly reduce the computational overhead by slightly sacrificing the number of segments that can be arranged as compared to RFS.**

***Index Terms*—Broadcasting, broadband networks, cable TV, video-on-demand.**

I. Introduction

ADVANCES in broadband networking and multimedia technologies offer easy, daily access to multimedia services through high-speed communication networks. One example is the Video-on-demand (VoD) service, which allows customers to connect to an on-line video server and watch videos asynchronously. A VoD system is typically implemented by a client-server architecture. However, so far VoD has not been commercially successful. One of the reasons is that it may require a number of channels (and thus communication bandwidth) proportional to the number of requests, which is infeasible when the demand is high. For example, a video in MPEG-2 compressed format requires a bandwidth of 0.5 MB/sec [17]. Times the number of requests, extremely high disk I/O and communication bandwidth are needed.

To reduce the cost, many broadcast-based approaches are proposed to provide near-VoD services. The *batching* approach collects a group of requests that arrive close in time, and serves them altogether when a channel is available [1], [3], [4]. Two *patching* schemes [5], [7] are proposed on top of the batching approach to allow late-coming clients to join the service. Adaptive batching schemes are proposed in [12], [13]. One prospective direction is to partition a video into multiple segments and broadcast them cooperatively by multiple channels [2], [6], [8]–[11], [14]–[16]. Since the server's broadcasting activity is independent of the arrivals of requests, such solutions are more appropriate for popular or hot videos that may attract many viewers at a certain period of time. The *pyramid scheme* [2], [16] can reduce clients' maximum waiting time in an exponential ratio with respect to the number of channels used. The *Fast Broadcasting (FB) scheme* [8], [9] only incurs $D/(2^k - 1)$ waiting time when using k channels, where D is the length of the video. The *PAGODA scheme* [10], [11] can further reduce the 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. In [6], Hu gives a comprehensive study of existing broadcasting protocols, and proposes a generalized analytical approach to evaluate such protocols. The recently proposed *Recursive Frequency Splitting (RFS) scheme* [14] can further reduce the waiting time significantly.

While efficient, the RFS scheme [14] suffers from a high computational complexity of $O(n \log n)$, where n is the number of segments that the video is partitioned into. Since n is typically very large, reducing the overhead is desirable. In this paper, we propose an efficient segmentation scheme, which can significantly reduce the computational overhead by slightly sacrificing the number of segments (i.e., n) that can be arranged.

The rest of this paper is organized as follows. Our new scheme is proposed in Section II. A further refinement is presented in Section III. Performance comparison is in Section IV.

II. The Proposed Broadcasting Scheme

Consider a video V of length D. We are given k channels, $C_0, C_1, \ldots, C_{k-1}$, each of a bandwidth capable of supporting broadcasting V. The problem is to find a largest possible n such that V is partitioned into n equal-length segments $S_1, S_2, \ldots, S_n$ and to obtain a placement of these n segments on the given k channels such that any client, after waiting no longer than D/n time, can start enjoying viewing V without any disruption.

Note that at this point, the value of n is not known yet. Its value will be an output of our algorithm. Once n is known, we can partition the k channels into time slots of length $\delta = D/n$. A client has to wait no longer than one time slot to start viewing V. Therefore, the maximum waiting time is D/n. A general guideline to guarantee a nondisrupted playback is given in [14].

Lemma 1: The i-th segment $S_i, 1 \leq i \leq n$, must be broadcast with a period p such that $p \leq i$.

Manuscript received December 31, 2003; revised January 5, 2004. The work of J. P. Sheu was sponsored by the NSC of the Republic of China under Grant Number 91-2213-E-008-025. The work of Y. C. Tseng was co-sponsored by the MOE Program for Promoting Academic Excellence of Universities under Grant Number 89-E-FA04-1-4, by NSC of Taiwan under Grant Numbers NSC92-2213-E009-076 and NSC92-2219-E009-013, by the Institute for Information Industry and MOEA, R.O.C., under the Handheld Device Embedded System Software Technology Development Project, and by the Lee and MTI Center of NCTU.

J.-P. Sheu and H.-L. Wang are with the Department of Computer Science and Information Engineering, National Central University, Chung-Li 32054, Taiwan (e-mail: sheujp@csie.ncu.edu.tw).

C.-H. Chang and Y.-C. Tseng are with the Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsin-Chu 30050, Taiwan.

Digital Object Identifier 10.1109/TBC.2004.828754

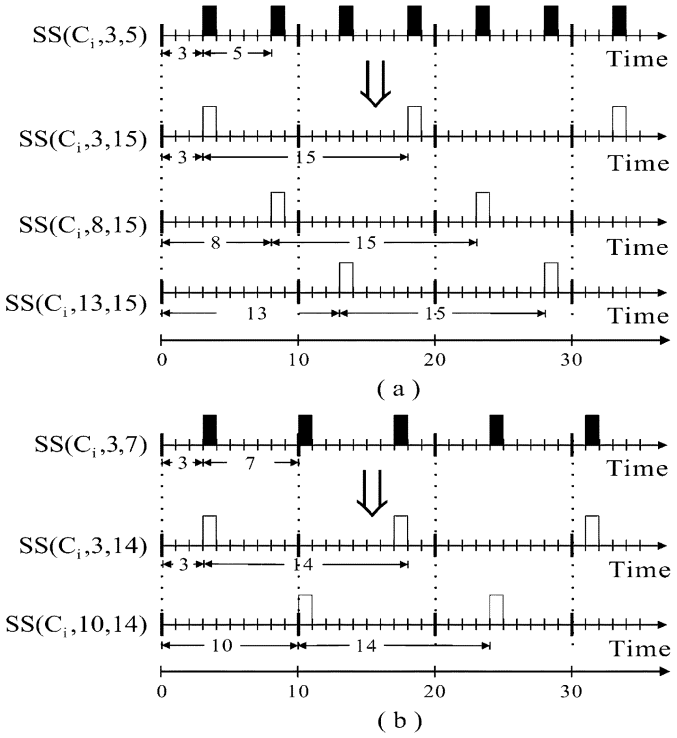


Fig. 1. (a) Splitting slot sequence $\mathrm{SS}(C_i, 3, 5)$ by a factor of 3, and (b) splitting $\mathrm{SS}(C_i, 3, 7)$ by a factor of 2.

Our goal is to assign for each segment a sequence of periodical time slots as defined below.

Definition 1: 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, $\eta \geq 0$ is an integer (i.e., offset), and $p \geq 1$ is an integer (i.e., period), $0 \leq \eta \leq p-1$.

In this work, we will count time slots of a channel starting from 0, instead of 1. Several examples of slot sequences are shown in Fig. 1.

The most bandwidth-efficient slot assignment, according to Lemma 1, is to assign a slot sequence of period i to S_i. However, this is not always feasible since two slot sequences on the same channels with periods that are mutually prime will eventually collide with each other. For example, given any two slot sequences $\mathrm{SS}(C_i, \eta, p)$ and $\mathrm{SS}(C_i, \eta', p')$ such p and p' are mutually prime, it is a simple result in number theory that we can find two integers i and j such that $\eta + ip = \eta' + jp'$. Thus, S_p and $S_{p'}$ cannot be placed in the same channel unless some bandwidth is sacrificed. When assigning a slot sequence $\mathrm{SS}(C_j, \eta, p)$ to S_i such that $p < i$, we say that the amount of waste is $(1/p) - (1/i)$. So we should make p as large as possible to reduce the waste.

Our scheme is based on a concept called "frequency-splitting" to generate slot sequences of different periods, as formulated below.

Lemma 2: A slot sequence $\mathrm{SS}(C_i, \eta, p)$ can be split by a factor of α into α slot sequences

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

Two splitting examples are in Fig. 1. Below we further generalize the result by grouping multiple slot sequences into a set, as defined below.

Definition 2: The union of q slot sequences with the same period p, namely $\mathrm{SS}(C_i, \eta_1, p)$, $\mathrm{SS}(C_i, \eta_2, p), \ldots, \mathrm{SS}(C_i, \eta_q, p)$, can be written as a *slot sequence set* $\mathrm{SSS}(C_i, \{\eta_1, \eta_2, \ldots, \eta_q\}, p)$.

Lemma 3: A slot sequence set $\mathrm{SSS}(C_i, \{\eta_1, \eta_2, \ldots, \eta_q\}, p)$ can be split by a factor of α into a slot sequence set $\mathrm{SSS}(C_i, \{\eta_a + bp \,|\, a = 1 \ldots q, b = 0 \ldots \alpha - 1\}, \alpha p)$.

For example, the $\mathrm{SSS}(C_i, \{3, 10\}, 14)$ in Fig. 1(b) can be split by a factor of 3 into $\mathrm{SSS}(C_i, \{3, 10, 17, 24, 31, 38\}, 42)$.

The basic idea of our scheme is to represent available slots as slot sequence sets and store them in a resource pool. We will use above lemma to split slot sequences, and multiple video segments will be assigned in groups, until the resource pool is exhausted. The scheme is formally derived below. The outputs are an n and the assignment of each segment $S_i, i = 1 \ldots n$, to one slot sequence.

1) Let the resource pool, denoted by a set *POOL*, be

$$\mathrm{POOL} = \{\mathrm{SSS}(C_i, \{0, 1, 2, \ldots, p(i) - 1\}, p(i)) \,|\, i = 0 \ldots k - 1\},$$

where $p(0) = 1$ and $p(i)$ is the i-th prime number when $i \geq 1$ (e.g., $p(1) = 2, p(2) = 3$, and $p(3) = 5$). Intuitively, this represents the set of all free slot sequences at the beginning.

2) Assign $\mathrm{SS}(C_i, 0, p(i))$ to $S_{p(i)}, i = 0 \ldots k-1$, and delete these slot sequences from *POOL*.

3) Let j be the smallest index such that segment S_j is not assigned to any slot sequence yet. Let $\mathrm{SSS}(C_{i1}, \Psi_1, p_1)$ be the slot sequence set in *POOL* such that its period, p_1, is the smallest (in case that *POOL* doesn't have any slot sequence set remaining, we can simply imagine that $p_1 = \infty$). Also, let $\mathrm{SSS}(C_{i2}, \Psi_2, p_2)$ be the slot sequence set in *POOL* such that its period, p_2, is the second smallest (in case that *POOL* has only one slot sequence set remaining, we simply let $p_2 = 2p_1$). We consider the relationship between j and p_1. There are three cases:

   a) If $j < p_1$, then no slot sequence in *POOL* can accommodate segment S_j. So we terminate this algorithm.

   b) If $p_1 \leq j < 2 \cdot p_1$, then we sequentially assign the slot sequences in $\mathrm{SSS}(C_{i1}, \Psi_1, p_1)$ to segments $S_j, S_{j+1}, \ldots, S_{2p_1 - 1}$ until either all segments are successfully assigned or the slot sequences in $\mathrm{SSS}(C_{i1}, \Psi_1, p_1)$ have been exhausted. Then go back to the beginning of step 3.

   c) If $2 \cdot p_1 \leq j$, then we split the slot sequence set $\mathrm{SSS}(C_{i1}, \Psi_1, p_1)$ by a factor of two into, say $\mathrm{SSS}(C_{i1}, \Psi_1', 2p_1)$, by applying Lemma 3. Then we sequentially assign the slot sequences in $\mathrm{SSS}(C_{i1}, \Psi_1', 2p_1)$ to segments $S_j, S_{j+1}, \ldots, S_{2p_2 - 1}$ until either all segments are successfully assigned or the slot sequences in $\mathrm{SSS}(C_{i1}, \Psi_1', 2p_1)$ have been exhausted. Then go back to the beginning of step 3.

TABLE I
THE ASSIGNMENT EXAMPLE WHEN $k = 4$ CHANNELS.

iteration	segments assigned	*POOL* (after the assignment)
0	S_1, S_2, S_3, S_5	$SSS(C_1, \{1\}, 2), SSS(C_2, \{1, 2\}, 3), SSS(C_3, \{1, 2, 3, 4\}, 5)$
1	S_4	$SSS(C_2, \{1, 2\}, 3), SSS(C_1, \{3\}, 4), SSS(C_3, \{1, 2, 3, 4\}, 5)$
2	S_6, S_7	$SSS(C_1, \{3\}, 4), SSS(C_3, \{1, 2, 3, 4\}, 5), SSS(C_2, \{4, 5\}, 6)$
3	S_8, S_9	$SSS(C_3, \{1, 2, 3, 4\}, 5), SSS(C_2, \{4, 5\}, 6)$
4	S_{10}, S_{11}	$SSS(C_2, \{4, 5\}, 6), SSS(C_3, \{3, 4, 6, 7, 8, 9\}, 10)$
5	$S_{12}, S_{13}, S_{14}, S_{15}$	$SSS(C_3, \{3, 4, 6, 7, 8, 9\}, 10)$
6	$S_{16}, S_{17}, S_{18}, S_{19}$	$SSS(C_3, \{8, 9\}, 10)$
7	$S_{20}, S_{21}, S_{22}, S_{23}$	$\emptyset$
8	$\emptyset$	$\emptyset$

Note that step 3a is the only exit of the algorithm. Steps 3b and 3c are where we assign multiple segments in one step. According to our experience, in most cases step 3c will be executed as opposed to step 3b. The following is an example for $k = 4$ channels.

1) In the beginning, we separate channels C_0, C_1, C_2, and C_3 into slot sequence sets $\mathrm{SSS}(C_0, \{0\}, 1), \mathrm{SSS}(C_1, \{0,1\}, 2), \mathrm{SSS}(C_2, \{0,1,2\}, 3)$, and $\mathrm{SSS}(C_3, \{0, 1, 2, 3, 4\}, 5)$.
2) Then we assign $\mathrm{SS}(C_0, 0, 1)$ to $S_1, \mathrm{SS}(C_1, 0, 2)$ to $S_2, \mathrm{SS}(C_2, 0, 3)$ to S_3, and $\mathrm{SS}(C_3, 0, 5)$ to S_5.
3) In the first iteration, $j = 4, p_1 = 2$, and $p_2 = 3$. We split $\mathrm{SSS}(C_1, \{1\}, 2)$ into $\mathrm{SSS}(C_1, \{1, 3\}, 4)$ and then assign $\mathrm{SS}(C_1, 1, 4)$ to S_4.
4) In the second iteration, $j = 6, p_1 = 3$, and $p_2 = 4$. We split $\mathrm{SSS}(C_2, \{1, 2\}, 3)$ into $\mathrm{SSS}(C_2, \{1, 2, 4, 5\}, 6)$ and then assign $\mathrm{SSS}(C_2, \{1, 2\}, 4)$ to $\{S_6, S_7\}$.
5) In the third iteration, $j = 8, p_1 = 4$, and $p_2 = 5$. We split $\mathrm{SSS}(C_1, \{3\}, 4)$ into $\mathrm{SSS}(C_1, \{3, 7\}, 8)$ and then assign $\mathrm{SSS}(C_1, \{3, 7\}, 8)$ to $\{S_8, S_9\}$.
6) In the fourth iteration, $j = 10, p_1 = 5$, and $p_2 = 6$. We split $\mathrm{SSS}(C_3, \{1, 2, 3, 4\}, 5)$ into $\mathrm{SSS}(C_3, \{1, 2, 3, 4, 6, 7, 8, 9\}, 10)$ and then assign $\mathrm{SSS}(C_1, \{1, 2\}, 10)$ to $\{S_{10}, S_{11}\}$.
7) In the fifth iteration, $j = 12, p_1 = 6$, and $p_2 = 10$. We split $\mathrm{SSS}(C_2, \{4, 5\}, 6)$ into $\mathrm{SSS}(C_2, \{4, 5, 10, 11\}, 12)$ and then assign $\mathrm{SSS}(C_2, \{4, 5, 10, 11\}, 12)$ to $\{S_{12}, S_{13}, S_{14}, S_{15}\}$. Note that in this case we have exhausted all slot sequences available in $\mathrm{SSS}(C_2, \{4, 5, 10, 11\}, 12)$.
8) In the sixth iteration, $j = 16, p_1 = 10$, and $p_2 = \infty$. In this case, we will enter step 3b and assign $\mathrm{SSS}(C_3, \{3, 4, 6, 7\}, 10)$ to $\{S_{16}, S_{17}, S_{18}, S_{19}\}$.
9) In the seventh iteration, $j = 20, p_1 = 10$, and $p_2 = 2p_1 = 20$. Again, we will enter step 3b, which will split $\mathrm{SSS}(C_3, \{8, 9\}, 10)$ into $\mathrm{SSS}(C_3, \{8, 9, 18, 19\}, 20)$ and then assign $\mathrm{SSS}(C_3, \{8, 9, 18, 19\}, 20)$ to $\{S_{20}, S_{21}, S_{22}, S_{23}\}$.
10) In the eighth iteration, since all slot sequences have been exhausted, $j = 24, p_1 = \infty$, and $p_2 = \infty$. We will enter step 3a, and the assignment will be terminated.

III. A FURTHER REFINEMENT

We make two observations on the above scheme. First, after the initial assignment in steps 1 and 2, *POOL* has slot sequences of periods $p(1), p(2), \ldots, p(k-1)$. Second, since slot sequence sets are always split by a factor of 2, the only periods that may appear in *POOL* is $2^i p(1), 2^i p(2), \ldots, 2^i p(k-1)$ for some i. For example, in Table I, the possible periods in *POOL* are 2, 3, 4, 5, 6, 8, 10, 12, 20, etc.

The basic idea in the refinement is to change the initial value of *POOL* so that there are more choices of periods in *POOL*. As noted earlier, assigning $\mathrm{SS}(C_i, \eta, p)$ to S_j causes a bandwidth waste of $(1/p) - (1/j)$. A more variety of periods may incur less waste. Given k channels, the refined scheme works as follows.

1) Still, let the resource pool, denoted by a set *POOL*, be

$$\mathrm{POOL} = \{\mathrm{SSS}(C_i, \{0, 1, 2, \ldots, p(i) - 1\}, p(i)) \,|\, i = 0 \ldots k-1\}.$$

Then, for each $\mathrm{SSS}(C_i, \{0, 1, 2, \ldots, p(i) - 1\}, p(i)), i = 2 \ldots k-1$, keep the first $\lceil (p(i)/2) \rceil$ slot sequences unchanged, and for each of the remaining $\lfloor (p(i)/2) \rfloor$ slot sequences, split it by a factor of $2j + 1$ such that $j = 1 \ldots \lfloor (p(i)/2) \rfloor$.
2) For each value of p that appears in *POOL* as a period of some slot sequence set, allocate one slot sequence from the corresponding SSS and assign it to segment S_p. Then delete the slot sequence from *POOL*.
3) Repeat steps 3 in the original scheme in Section II.

Table II illustrates the refined assignment example with 4 channels. Since the difference between the refined assignment and the original assignment only happens at the first two steps, we only discuss the first few iterations below:

1) In the beginning, we separate channels C_0, C_1, C_2, and C_3 into slot sequence sets $\mathrm{SSS}(C_0, \{0\}, 1), \mathrm{SSS}(C_1, \{0, 1\}, 2), \mathrm{SSS}(C_2, \{0, 1, 2\}, 3)$, and $\mathrm{SSS}(C_3, \{0, 1, 2, 3, 4\}, 5)$.
2) Then, we separate $\mathrm{SSS}(C_2, \{2\}, 3)$ into $\mathrm{SSS}(C_2, \{2, 5, 8\}, 9)$, separate $\mathrm{SSS}(C_3, \{3\}, 5)$ into $\mathrm{SSS}(C_3, \{3, 8, 13\}, 15)$, and separate $\mathrm{SSS}(C_3, \{4\}, 5)$ into $\mathrm{SSS}(C_3, \{4, 9, 14, 19, 24\}, 25)$.
3) Assign $\mathrm{SS}(C_0, 0, 1)$ to $S_1, \mathrm{SS}(C_1, 0, 2)$ to $S_2, \mathrm{SS}(C_2, 0, 3)$ to $S_3, \mathrm{SS}(C_3, 0, 5)$ to $S_5, \mathrm{SS}(C_2, 59)$ to $S_9, \mathrm{SS}(C_3, 3, 15)$ to S_{15}, and $\mathrm{SSS}(C_3, 4, 25)$ to S_{25}.
4) The following steps are quite similar to the original scheme. Note that in the ninth iteration, we have $j = 21, p_1 = 25$, and $p_2 = 50$. Although the slot sequences have not been exhausted yet, we have to enter step 3a and terminate the algorithm.

TABLE II
THE REFINED ASSIGNMENT EXAMPLE WHEN $k = 4$ CHANNELS.

iteration	segments assigned	$POOL$ (after the assignment)
0	$S_1, S_2, S_3, S_5, S_9, S_{15}, S_{25}$	$SSS(C_1,\{1\},2), SSS(C_2,\{1\},3), SSS(C_3,\{1,2\},5)$ $SSS(C_2,\{5,8\},9), SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
1	S_4	$SSS(C_2,\{1\},3), SSS(C_1,\{3\},4), SSS(C_3,\{1,2\},5)$ $SSS(C_2,\{5,8\},9), SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
2	S_6, S_7	$SSS(C_1,\{3\},4), SSS(C_3,\{1,2\},5)SSS(C_2,\{5,8\},9),$ $SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
3	S_8	$SSS(C_3,\{1,2\},5), SSS(C_1,\{7\},8)S(C_2,\{5,8\},9),$ $SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
4	$S_{10}, S_{11}, S_{12}, S_{13}$	$SSS(C_2,\{7\},8), SSS(C_2,\{5,8\},9), SSS(C_3,\{9\},10),$ $SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
5	S_{14}	$SSS(C_2,\{5,8\},9), SSS(C_3,\{9\},10),$ $SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
6	S_{16}, S_{17}	$SSS(C_3,\{9\},10), SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
7	S_{18}	$SSS(C_3,\{8,13\},15), SSS(C_3,\{9,14,19,24\},25)$
8	S_{19}, S_{20}	$SSS(C_3,\{9,14,19,24\},25)$
9	$\emptyset$	$SSS(C_3,\{9,14,19,24\},25)$

TABLE III
THE NUMBERS OF MAXIMUM SEGMENTS (n) OFFERED BY DIFFERENT SCHEMES.

channels (k)	3	4	5	6	7	8	9	10	11	12	13	14	15
RFS [14]	9	25	73	201	565	1522	4284	11637	31677	86428	237705	653958	1792699
our refined scheme	9	20	71	193	509	1395	3535	9087	27423	73215	192639	503295	1212415
our basic scheme	9	23	63	135	319	1055	2431	6399	17919	51199	131215	290815	655359
New PAGODA [10]	9	26	66	172	442	1183	3092	8285	N/A	N/A	N/A	N/A	N/A
PAGODA [11]	9	19	49	99	249	499	1249	2499	6249	12499	31249	62499	156249
FB [8, 9]	7	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767

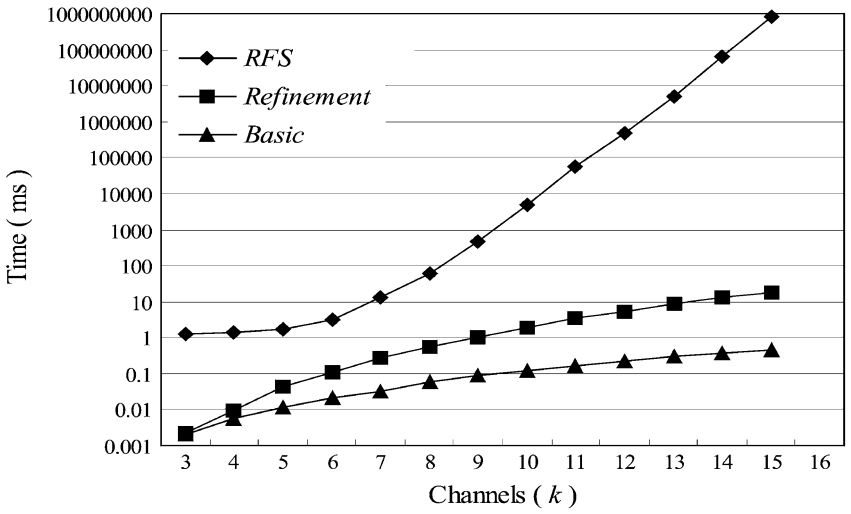


Fig. 2. Comparison of computational overhead of our and RFS schemes.

IV. PERFORMANCE ANALYSIS AND COMPARISON

Table III compares the number of segments that can be arranged (i.e., n) by different schemes. The inverse of n represents clients' waiting time. As can be seen, our schemes outperform the FB, PAGODA, and new PAGODA schemes as k enlarges, but slightly fall behind the RFS scheme. Also note that the new PAGODA scheme is built purely by heuristic and thus a general result for n is not available.

The buffer requirement at the client side is also an important issue for video broadcasting. Since the server side broadcasts k segments every time slot when using k channels, we have to buffer at most k more segments but consume only one segment from the buffer after every time slot. However, we do not need to buffer S_j at the i-th time slot such that $0 < j < i$. So, the maximum buffer requirement at the client side when using k channels must be bounded by:

$$B_k = \max_{0<i\le n} \left\{ k \times i - \sum_{j=1}^{i} \left\lfloor \frac{i}{p_{k,j}} \right\rfloor \right\},$$

where $p_{k,j}$ the time period of the slot sequence assigned to the segment S_j when using k channel.

Next, we analyze the time complexity of our schemes. In each assignment we only scan *POOL* for the smallest and second smallest periods. Also observe that the size of *POOL* is $O(k)$ for the basic scheme, and $O(k^2)$ for the refined scheme. If *POOL* is maintained by a heap tree, then both deletion and insertion operations to the heap will incur a cost of $O(\log k)$ for both schemes. So the time complexity for both the basic and refined schemes is $O(n \log k)$.

In comparison, the RFS scheme is also derived based on the concept of frequency splitting. However, segments are assigned one at a time in RFS. To assign a segment, the whole *POOL* has to be scanned to minimize the bandwidth waste. The overall computational cost is $O(n^2)$. Further, in practice our schemes assign multiple segments at a time. The actual computational overhead should be much smaller than $O(n \log k)$. Fig. 2 shows our simulation result on computational cost when running the segment assignments on a Pentium III, 450 MHz, IBM-compatible PC with 128 Mb RAM. As can be seen, the overhead of the proposed schemes are much smaller than that of RFS. Even when k is quite large, only a few milliseconds are needed.

V. Conclusion

We have proposed efficient segment-scheduling schemes to support video broadcasting. The schemes effectively arrange segments by assigning multiple segments at one time. It significantly reduces the computational overhead by slightly sacrificing clients' waiting time as compared to the RFS scheme.

References

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

[2] ——, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *Int'l. Conf. on Multimedia Computing and Systems*, June 1996, pp. 118–126.

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

[4] ——, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, no. 3, pp. 112–121, June 1996.

[5] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *IEEE Multimedia*, 1999, pp. 117–121.

[6] A. Hu, "Video-on-demand broadcasting protocols: A comprehensive study," in *INFOCOM*, Apr. 2001, pp. 508–517.

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

[8] L.-S. Juhn and L.-M. Tseng, "Fast broadcasting for hot video access," *Real-Time Computing Systems and Applications*, pp. 237–243, Oct. 1997.

[9] ——, "Fast data broadcasting and receiving scheme for popular video service," *IEEE Trans. on Broadcasting*, vol. 44, no. 1, pp. 100–105, Mar. 1998.

[10] J.-F. Paris, "A simple low-bandwidth broadcasting protocol for video-on-demand," in *Int'l. Conf. on Computer Communication and Network*, 1999, pp. 118–123.

[11] J.-F. Paris, S.-W. Carter, and D.-D. Long, "A hybrid broadcasting protocol for video on demand," in *Multimedia Computing and Networking Conference*, 1999, pp. 317–326.

[12] W. F. Poon and K. T. Lo, "New batching policy for providing true video-on-demand (T-VoD) in multicast system," in *IEEE ICC*, 1999, pp. 983–987.

[13] W. F. Poon, K. T. Lo, and J. Feng, "Adaptive batching scheme for multicast video-on-demand systems," *IEEE Trans. on Broadcasting*, vol. 47, no. 1, pp. 66–70, Mar. 2001.

[14] Y.-C. Tseng, M.-H. Yang, and C.-H. Chang, "A recursive frequency-splitting scheme for broadcasting hot videos in VoD service," *IEEE Trans. on Communications*, pp. 1348–1355, Aug. 2002.

[15] Y.-C. Tseng, M.-H. Yang, C.-M. Hsieh, W.-H. Liao, and J.-P. Sheu, "Data broadcasting and seamless channel transition for highly-demanded videos," *IEEE Trans. on Communications*, vol. 49, no. 5, pp. 863–874, May 2001.

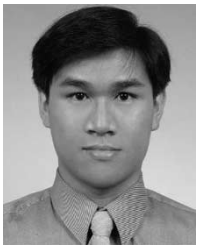
[16] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *IEEE Multimedia Systems*, vol. 4, pp. 197–208, 1996.

[17] W. E. Wright, "An efficient video-on-demand model," *IEEE Computer*, vol. 34, no. 5, pp. 64–70, May 2001.

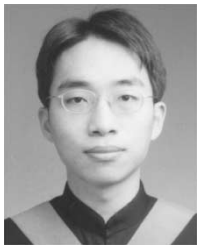
Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively.

He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from August 1997 to July 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing, parallel processing, and distributed computing Systems. He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from August 1996 to July 2000. He was an associate editor of Journal of Information Science and Engineering from August 1996 to July 2002. He is an associate editor of Journal of the Chinese Institute of Engineers. He was a Guest Editor of Special Issue for Wireless Communications and Mobil Computing Journal. He was a Program Chair of IEEE ICPADS'2002. He was a Vice-Program Chair of ICPP 2003.

He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998. He was the Specially Granted Researchers, National Science Council, from 1999 to 2002. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. Dr. Sheu is a senior member of the IEEE, a member of the ACM and Phi Tau Phi Society.



Han-Lih Wang received the B.S. and M.S. degrees in computer science and information engineering from the National Central University, Taiwan, Republic of China, in 1999 and 2001, respectively. He is a senior engineer in Electronic Commerce Universal Inc.



Chi-He Chang received the B.S. degree in computer science from the Chung Yuan Christian University and M.S. degree in computer science and information engineering from National Central University, Taiwan in 1998 and 2000, respectively. He is currently a Ph.D. student at the Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan. His research interests include video on demand, interactive multimedia, and wireless networks.

Yu-Chee Tseng received his B.S. and M.S. degrees in computer science from the National Taiwan University and the National Tsing-Hua University in 1985 and 1987, respectively. He worked for the D-LINK Inc., as an engineer in 1990. He obtained his Ph.D. in computer and information science from the Ohio State University in January of 1994. He was an Associate Professor at the Chung-Hua University (1994–1996) and at the National Central University (1996–1999), and a Full Professor at the National Central University (1999–2000). Since 2000, he has been a Full Professor at the Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan.

Dr. Tseng served as a Program Chair in the ***Wireless Networks and Mobile Computing Workshop***, 2000 and 2001, as a Vice Program Chair in the ***International Conference on Distributed Computing Systems*** (ICDCS), 2004, as a Vice Program Chair in the ***IEEE Int'l. Conf. on Mobile Ad-hoc and Sensor Systems*** (MASS), 2004, as an Associate Editor for ***The Computer Journal***, as a Guest Editor for ***ACM Wireless Networks*** special issue on "Advances in Mobile and Wireless Systems", as a Guest Editor for ***IEEE Transactions on Computers*** special on "Wireless Internet", as a Guest Editor for ***Journal of Internet Technology*** special issue on "Wireless Internet: Applications and Systems", as a Guest Editor for ***Wireless Communications and Mobile Computing*** special issue on "Research in Ad Hoc Networking, Smart Sensing, and Pervasive Computing", as an Editor for ***Journal of Information Science and Engineering***, as a Guest Editor for ***Telecommunication Systems*** special issue on "Wireless Sensor Networks", and as a Guest Editor for ***Journal of Information Science and Engineering*** special issue on "Mobile Computing".

He is a two-time recipient of the Outstanding Research Award, National Science Council, ROC, in 2001–2002 and 2003–2005, and a recipient of the Best Paper Award in International Conference on Parallel Processing, 2003. Several of his papers have been chosen as Selected/Distinguished Papers in conferences and been included for publications in journals. He has guided students to participate in several national programming contests and received several awards. His research interests include mobile computing, wireless communication, network security, and parallel and distributed computing. Dr. Tseng is a member of ACM and a Senior Member of IEEE.