Scheduling Techniques for Media-on-Demand

Amotz Bar-Noy *   Richard E. Ladner †   Tami Tamir ‡

Abstract

Broadcasting popular media to clients is the ultimate scalable solution for media-on-demand. Recently, it was shown that if clients can receive data at a rate faster than what they need for playback and if they can store later parts of the media in their buffers, then much higher scalability may be obtained. In the paper we focus on scheduling problems arising from these new systems for media-on-demand.

For given amount of bandwidth, we improve the guaranteed start-up delay time for an uninterrupted playback. We achieve our results by introducing two techniques. In the first, the media is arranged on the channels such that clients gain from buffering later parts of the transmission before the actual start of the playback. In the second, segments of different media may be mixed together on the same channel. We introduce a simple class of recursive round-robin scheduling algorithms that implement our techniques.

Our results improve the best known asymptotic results. Moreover, our scheduling algorithms outperform known results for "practical" values for number of media and number of broadcasting channels. For some specific small values, we present hand designed solutions that are better than those achieved by our algorithms.

1 Introduction

Media-on-demand (MoD) is the demand by clients to read, listen, or view various type of media. In its simplest function, the clients would like to have an uninterrupted playback with as minimal as possible start-up delay. The subject of this paper is to improve the guaranteed start-up delay for MoD systems that support uninterrupted service. Our main objective is to achieve the smallest guaranteed start-up delay for given amount of resources (bandwidth).

There are two main types of systems that support MoD: unicast systems and broadcast systems. The former guarantees an immediate service as long as there are not too many clients (that is, it is not scalable). The latter can support many clients but cannot guarantee immediate service. In this paper, we improve the tradeoffs between the system resources for broadcast systems and the guaranteed start-up delay.

In the simplest implementation of MoD systems, clients who wish to view a movie[1], select the channel that would start broadcasting this movie the earliest after their request time. Movies are broadcast on one channel or several channels. Thus if h channels are allocated to a movie of length L units of time, the guaranteed start-up delay is L/h units by starting a new transmission every L/h time units.

Recently, Viswanathan and Imielinski proposed the Pyramid Scheme that for a given amount of bandwidth guarantees exponentially smaller start-up delay to clients that can buffer parts of the movie and can receive data from several channels concurrently ([13]). Many variants and generalizations of the Pyramid Scheme followed and all of them showed this dramatic improvement. We adopt the following discrete model that we describe shortly in order to demonstrate the ideas behind the Pyramid Scheme and our techniques.

The system has h channels and it broadcasts m movies, each of length L. Each movie is partitioned into s segments of equal length. The segments of the movies may be broadcast in any order on any channel. We assume that it takes one time slot to transmit or view a segment and the length of the time slot is L/s. We further assume that all the channels are synchronized in the sense that the starting points for the time slots coincide in all of them. Clients may buffer or view segments from any channel since they may receive data from all of them. In fact, without loss of generality, clients buffer or view segment i the first time they can do so after their arrival time. Clients start buffering segments at the first starting point of a segment but they may start the viewing process later.

The problem thus becomes a scheduling problem. To guarantee a start-up delay of one time slot, segment i must appear at least once in one of the channels in any window of i slots. This way, the client waits at most one slot to the next starting time of a slot, it then starts viewing and buffering. Since segment i is available within the next i time slots, an uninterrupted playback is guaranteed. The Pyramid scheme and all the other

*Computer and Information Science Department, Brooklyn College, 2900 Bedford Avenue Brooklyn, NY 11210. E-mail: *amotz@sci.brooklyn.cuny.edu*.

†Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195. This work was partially supported by NSF grant No. CCR-0098012. E-mail: *ladner@cs.washington.edu*.

‡Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195. E-mail: *tami@cs.washington.edu*.

[1] We use the terminology of movies for convenience.

solutions employ the following schedule design principle:

> *Early segments should be broadcast more frequently than later segments.*

Example I: To demonstrate the usefulness of this principle consider the case of one movie of length L and two channels. The traditional solution of broadcasting the movie every $L/2$ time guarantees a start-up delay of $L/2$. The following schedule is better:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 & \cdots \end{bmatrix} \tag{1.1}$$

In this schedule the movie is partitioned into three segments each of length $L/3$. The first channel transmits the first segment every slot and the second channel transmits the second and the third segments alternately. A client needs only to wait for a beginning of a slot. If the client arrives before a slot in which the second segment is transmitted on the second channel, it buffers it and at the same time it views the first segment from the first channel. Then it views the second segment from its buffer while buffering the third segment from the second channel; finally, it views the third segment from its buffer. If the client arrives before a slot in which the third segment is transmitted on the second channel, it buffers it and at the same time it views the first segment from the first channel. Then it views the second segment from the second channel and finally it views the third segment from its buffer. In both cases the delay is bounded by the length of a segment which is $L/3$.

One of our techniques is based on the observation that if clients start buffering segments before they start viewing the movie then to guarantee a start-up delay of d time slots, segment i must appear at least once in one of the channels in any window of $d + i - 1$ slots. Surprisingly, as we illustrate in the next example, buffering by itself is enough to improve the start-up delay time. That is, even if clients may receive data from only one channel.

Example II: Consider the case of one movie of length L and one channel. The traditional solution broadcasts the movie repeatedly to guarantee a start-up delay L. The following cyclic schedule, with $d = 4$, is better:

$$\begin{bmatrix} 1 & 3 & 2 & 4 & 1 & 5 & 2 & 3 & 1 & 4 & 2 & 5 & \cdots \end{bmatrix} \tag{1.2}$$

In this schedule the movie is partitioned into five segments each of length $L/5$. The first and the second segments are transmitted every 4 slots and the other three segments are transmitted every 6 slots. A client waits for a starting time of a slot and then starts buffering segments it does not have in its buffer. After three more slots the client starts viewing the movie. In any case, the client waits at most 4 slots which means a guaranteed start-up delay of $0.8L$. One can verify that the above schedule works by checking all the possible arrival times. Assume for example that a client arrives in the middle of the slot in which the fourth segment is transmitted. This client buffers the first, second, and fifth slots before starting the viewing process. Then while viewing the first segment from its buffer the client buffers the third segment and while viewing the third segment from its buffer the client buffers the fourth segment. Thus, the client is viewing segment i on time for $1 \leq i \leq 5$.

1.1 Related work and prior results.

MoD systems, and in particular the solution of broadcasting, have been studied extensively in recent years. The paper [3] survey broadcasting protocols and describes the development of these protocols, starting with Staggered broadcasting protocols, in which the movies are simply transmitted repeatedly on the channels ([4]), through Pyramid-based broadcasting protocols, in which movies are partitioned into segments and different segments are broadcast on different channels ([13]), and finally Harmonic broadcasting protocols in which segment i is allocated bandwidth proportional to $1/i$ ([7, 1]).

The papers [6, 8] present a simple schedule of one movie on h channel by partitioning the movie into $2^h - 1$ segments. Their schedule implies a guaranteed start-up delay of $L/(2^h - 1)$ for a movie of length L. The Pagoda scheme ([11]) is based on a schedule for 3 channels with guaranteed start-up delay $L/9$. It is then generalized to a schedule that asymptotically guarantees a start-up delay of $L/O(2.236^h)$. The new Pagoda scheme ([9]) deals with small values of h. Their guaranteed start-up delay for $h = 3, 4, 5, 6$ is $L/9, L/26, L/66, L/172$ respectively.

Engebretsen and Sudan showed in [5] that harmonic broadcasting is actually a special case of priority encoded transmission, and conclude a lower bound of $D \geq 1/(e^h - 1)$ on the guaranteed start-up delay.

Harmonic broadcasting is implemented in [1] by a reduction from the *window-scheduling* problem. Specifically, the movie is partitioned into s equal-sized segments that are scheduled on the channels such that the gap between any two consecutive appearances of segment i is at most i. For a given number of channels, the goal is to maximize s, and as a result, minimize the guaranteed start-up delay (which is $1/s$). A schedule based on this principle is shown to approach the lower bound as $h \to \infty$.

1.2 Our contributions.

In this paper we present two new scheduling techniques, each of which enables us to minimize the guaranteed start-up delay. These techniques yield broadcasting protocols with improved delay and can be implemented with no need to 'upgrade' the system: clients are required, like in previous schemes, to be able to receive data from several channels concurrently, and to store the received data in a local buffer.

The first technique, called *shifting*, is based on increasing the number of segments to which the movie is partitioned. The client is required to buffer the segments greedily, that is, at the earliest time a segment is transmitted on some channel, even if this occurs before the client starts viewing the movie. By arranging the segments on the channels in a way that exploit this greediness, we reduce the maximum start-up delay.

In the second technique, called *channel sharing*, we use the fact that the server usually broadcasts more than a single movie at a time. Let m be the number of movies transmitted by the server. In traditional schemes, there are h/m channels dedicated to each movie. Our idea is to broadcast segments of distinct movies on the same channel. As long as we keep the ratio $\rho = h/m$ fixed, we do not need to enhance the system, we gain freedom in the way segments are assigned on the channels. To analyze this technique, we extend the lower bound for a single movie to a lower bound that depends on ρ. We prove that the delay approaches the lower bound as m grows.

The two techniques can be combined together to yield even better broadcasting protocols. We develop simple scheduling algorithms, denoted *recursive round-robin*, in which the two techniques are implemented. We present asymptotic analysis and simulation results of these algorithms. We would like to emphasize that our analysis shows that these algorithms approach the lower bound for any fixed value of $h \geq 1$. Whereas previous results approach the lower bounds for $h \rightarrow \infty$ ([1]). The simulation results imply that these techniques yield good schedules already for small, practical, values of h and m. For specific small values of h, that are of much interest, we develop specific schedules that outperform the best known schedules.

Finally, we show that these techniques can be used also to reduce the average delay and to improve the guaranteed start-up delay when clients have limited number, $r < h$, of data receivers (the receive-r model is presented in [12]).

2 Model and Preliminaries

We assume a system with h channels each can broadcast segments of m movies. The quantity $\rho = h/m$ represents the bandwidth per movie. Unless specified otherwise, we assume that all the m movies have the same length, L, normalized to be one time unit (that is, $L = 1$). Clients may receive data from all the channels concurrently and buffer segments in their local memory. When they view the movie they can either view it directly from one of the channels or from their buffers. In our broadcasting schemes, each movie is partitioned into s *segments*. The length of one segment which is the length of one *time slot* is $1/s$ time units (since $L = 1$). The z-th segment of movie i for $1 \leq i \leq m$ and $1 \leq z \leq s$ is denoted $[z]_i$. [2]

Let D denote the guaranteed start-up delay time to get an uninterrupted playback. The delay D is a fractional number between 0 and 1 since $L = 1$. It is sometimes more convenient to measure the delay in time-slot units; we denote by $d = s \cdot D$ the guaranteed *slot delay*.

A schedule for a channel is a sequence: $[[z_1]_{i_1}, \ldots, [z_\ell]_{i_\ell}]$ such that $1 \leq i_j \leq m$ and $1 \leq z_j \leq s$ for $1 \leq j \leq \ell$. This is a cyclic schedule that represents an infinite schedule. That is, the segments $[[z_1]_{i_1}, \ldots, [z_\ell]_{i_\ell}]$ will be broadcast repeatedly. A matrix of h rows represents h schedules for h channels. The gap between any two consecutive appearances of segment $[z]_i$ in the matrix is the difference between their respective column indices. We define the *window size* of segment $[z]_i$ to be the maximum size of one of its gaps (with wrap-around). For example, 1 and 2 have window sizes of 4, while 3,4, and 5 have window sizes of 6. in the schedule 1.2. In this example, the gaps between any one segment do not vary.

Given h, m and delay D, we now state the necessary condition for a schedule to guarantee a maximum start-up delay D for any client and any movie. Let A be the schedule that broadcasts s segments for each one of the m movies. Recall that $0 \leq D \leq 1$, and for convenience assume that the slot delay $d = s \cdot D$ is an integer.

THEOREM 2.1. *The schedule A guarantees a maximum start-up delay D if and only if the window size of segment $[z]_i$ is at most $d + z - 1$ for each $1 \leq z \leq s$, and $1 \leq i \leq m$.*

Proof. If there is a window of size $d + z$ that does not contain the segment z of a movie, then a client arriving after the beginning and during the slot immediately prior to the window will not receive the beginning of the z-th segment of this movie in time to view it even if it waits d time to start the viewing process. On the other hand, if all the gaps are bounded as stated in the theorem, any client will have any segment either in its

[2] When there is only one movie or when it is clear which is the movie, we use only z to denote the z-th segment. When z is a specific number we omit the square parenthesis.

buffer or on one of the channel when the client wishes to view this segment. This is because a client may receive and/or store data from all the h channels concurrently.

Objective functions: We distinguish between two types of optimization goals. In the first we fix h and m and try to minimize D as a function of s. We also try to approach the lower bound on D as s grows. In the second, we fix D and m and try to minimize h (or equivalently ρ) as a function of s.

2.1 Lower bounds. Engebretsen and Sudan [5] have shown that for h channels and one movie the guaranteed start-up delay is bounded by $1/(e^h - 1)$. Our generalization to the case of m movies is stated in the following theorem. We make the assumption that movies are composed of bits and that the smallest possible segment is one bit.

THEOREM 2.2. *The guaranteed start-up delay for h channels and m movies is $D \geq \frac{1}{e^{h/m}-1}$. Equivalently, for m movies and guaranteed delay D, we have, $h \geq m \ln\left(1 + \frac{1}{D}\right)$.*

Proof. Assume that each movie contains B bits. The delay is measured by d bits and the general delay D is therefore $\frac{d}{B}$. We number the bits from 1 to B. By Theorem 2.1, bit z must appear at least once in any window of size $d + z - 1$. Hence, bit z requires $\frac{1}{d+z-1}$ of the bandwidth, and all the bits of a movie require bandwidth $f(B, d) = \sum_{z=1}^{B} \frac{1}{d+z-1}$. Thus, for m movies we must have $h \geq m f(B, d)$.

$$f(B,d) = \sum_{z=d}^{d+B-1} \frac{1}{z} \geq \int_{x=d}^{d+B} \frac{dx}{x} = \ln(1 + \frac{B}{d})$$

which yields our result.

2.2 The RRR class of schedules. A channel or a matrix schedule is called *perfect*, if all the gaps between any two appearance of a segment $[z]_i$ are equal. The asymptotic results in this paper are based on a class of perfect schedules that generalize the round-robin schedule. We now present this class of schedules, denoted RRR (recursive round-robin), and give two ways to represent an RRR schedule, using parentheses or trees (see also [2]).

The simplest perfect schedule is the round-robin schedule in which each segment appears exactly once and all the segments in the schedule are distinct. That is, if the length of the schedule is ℓ then the window size of each one of the ℓ segments is ℓ. Let $z_1, \ldots, z_\ell$ denote the scheduled segments. The parenthesis representation of an RR^1-schedule is simply $(z_1, z_2, \ldots, z_\ell)$. The tree representation of an RR^1-schedule is a star tree whose root has a degree ℓ and each segment is associated with one of the leaves.

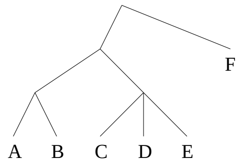

Figure 1: A tree representation of an RR^3 schedule.

A schedule S is called an RR^k schedule, if it is a round-robin schedule that iterates over the RR^{k-1}-schedules $S'_1, \ldots, S'_\Delta$ for some $\Delta \geq 1$. For example, the schedule $[A, C, B, D, A, E, B, C, A, D, B, E]$ is an RR^2 schedule with $\Delta = 2$, $S'_1 = (A, B)$, and $S'_2 = (C, D, E)$. It is generated by iterating between these RR^1-schedules. The length of an RR^k schedule is $\Delta \cdot LCM(\ell'_1, \ldots, \ell'_\Delta)$ where ℓ'_i is the length of S'_i.

The parenthesis representation of an RR^k schedule is $((S'_1), (S'_2), \ldots, (S'_\Delta))$. The tree representation of an RR^k schedule is a tree whose root has a degree Δ, and the Δ subtrees are the tree representations of $S'_1, S'_2, \ldots, S'_\Delta$. For example, $[A, F, C, F, B, F, D, F, A, F, E, F, B, F, C, F, A, F, D, F,$ $B, F, E, F]$ is an RR^3 schedule whose parenthesis representation is $(((A, B), (C, D, E)), (F))$ and whose tree representation is illustrated in Figure 1. This schedule has $\Delta = 2$ and the two RR^2 schedule composing it are $S'_1 = ((A, B), (C, D, E))$ and $S'_2 = (F)$.

Finally, a schedule is an RRR schedule if it is an RR^k schedule for some integer $k \geq 1$.

3 Schedule Designing Techniques

Our first technique, called *the shifting technique*, is based on the observation that buffering may start before the actual playback. In previous schemes it is assumed that clients start the viewing process at the same time they start the buffering process[3]. As a result, the window size of the z-th segment of any movie must be bounded by z. In particular, m channels need to be dedicated for the first segments of the m movies. The guaranteed delay was the length of a segment since clients that arrived a bit after the starting time of a segment had to wait almost a whole segment to get an uninterrupted playback.

For example, the optimal schedule for $h = 2$ and $m = 1$ is schedule (1.1) given in the introduction. Having $s = 3$, its guaranteed start-up delay is 1/3.

[3]One exception is in polyharmonic broadcasting ([10]), where clients always buffer data before they start viewing. However, unlike our scheme, their channels have different bandwidths.

Generally, harmonic schedules follow the scheduling design principle and for some range $[1..s]$, each segment, $z, 1 \leq z \leq s$, is scheduled with window at most z. This implies a guaranteed start-up delay $1/s$. When buffering can start before the viewing process we can do better. By Theorem 2.1, for a delay of d slots, the window size of a z-th segment could be $d+z-1$. Thus, the first segment can have window size d, and in general, instead of scheduling $[1..s]$ we can schedule another range, $[d..d+s'-1]$ for a larger s' number of segments. The key-idea is that for any $d>1$ we will have $d/s' < 1/s$, which implies a reduced delay.

We say that a schedule S on the range $[x..y]$ is *valid* if the window size for z is at most z for $x \leq z \leq y$. The guaranteed start-up delay of a valid schedule for one movie is stated in the following lemma. The *shifted schedule* S' of a schedule S on the range $[x..y]$ is a schedule on the range $[1..y-x+1]$ in which each z in S is replaced by $z-x+1$.

LEMMA 3.1. *Let S be a valid channel schedule on the range $[x..y]$ and let S' be its shifted schedule on the range $[1..y-x+1]$. Then S' guarantees a start-up delay of $D \leq x/(y-x+1)$.*

Proof. There are $s = y-x+1$ segments and therefore the length of each segment is $1/(y-x+1)$. A client must wait x slots to get segment 1 that was segment x in the valid schedule S. After this time, the client is guaranteed to get segment z on time for $1 \leq z \leq y-x+1$ due to the validity of S in which z was $z+x-1$.

Consider for example the following 2-channel schedule for the range $[2..9]$:

$$C_1 : (2, (4, 5)); \qquad C_2 : (3, (6, 7), (8, 9)).$$

Note that the window size of $z \in \{2, 3, 4, 6\}$ is z and the window size of $z \in \{5, 7, 8, 9\}$ is less than z. The requirement for a valid schedule is that the window size of z is at most z. Thus, the above is a valid schedule on the range $[2..9]$. After shifting, the schedule on the range $[1..8]$ is:

$$C_1 : (1, (3, 4)); \qquad C_2 : (2, (5, 6), (7, 8)).$$

In this schedule the length of a segment is $1/8$ and the guaranteed start-up delay is 2 segments which yields a delay of $1/4$. This improves the best delay of $1/3$ for two channels when viewing must begin immediately when receiving segment 1. Note that the delay time of the client is used in this scheme to buffer segments.

Interestingly, the shifting technique can be used to reduce the maximal delay even for one channel. Without shifting, the best we can do with one channel is a periodic broadcast of the whole movie. Unlucky clients that arrive right after the beginning of the broadcast have to wait the whole transmission. Consider the following valid schedule on the range $[4..8]$: $((4, 5), (6, 7, 8))$. One can verify that the window size of z is at most z for $4 \leq z \leq 8$. This schedule implies the following shifted schedule on the segments $[1..5]$: $((1, 2), (3, 4, 5))$. By Lemma 3.1, the guaranteed start-up delay is $4/(8-4+1) = 0.8$ which is less than 1.

In our second technique, called *the channel sharing technique*, we schedule segments of different movies on the same channel. The following example demonstrates the usefulness of channel sharing for $h = 6$ and $m = 2$. When each channel broadcasts only segments of one movie, the best known schedule (without shifting) schedules the segments $[1..9]$ of each movie. This implies a delay of $1/9$ [9, 1]. The following schedule guarantees a delay of $1/10$ by scheduling 10 segments of each movie on the 6 channels. One could verify that indeed the window size of segment $[z]_i \in \{1, \ldots, 10\}$ is at most z for $i \in \{1, 2\}$.

$$C_1 : (1_1); \qquad C_2 : (1_2); \qquad C_3 : (2_1, 2_2);$$
$$C_4 : (3_1, 3_2, (6_1, 6_2)); \qquad C_5 : (4_1, 4_2, (8_1, 8_2), (9_1, 9_2));$$
$$C_6 : (5_1, 5_2, 7_1, 7_2, (10_1, 10_2));$$

The two techniques can be combined to get better schedules. We demonstrate this with an example for $h = 4$ and $m = 2$. Without the shifting technique, the guaranteed start-up delay is $1/3$ by using two channels per movie. With the shifting technique alone, we showed above a valid $[2..9]$-schedule with delay at most $1/4$.

The following is a schedule of the range $[3..17]$ of both movies, on 4 channels. By Lemma 3.1, the delay is at most $3/(17-3+1) = 1/5$.

$$C_1 : ((3_1), (3_2), (6_1, 6_2))$$
$$C_2 : ((4_1), (4_2), (8_1, 8_2), (12_1, 12_2, 13_1))$$
$$C_3 : ((7_1, 7_2), (11_1)), \ \ (5_1), \ \ ((10_1), (7_1, 7_2)), \ \ (5_2), \ \ ((7_2, 7_1), (10_2))$$
$$C_4 : ((11_2, 9_1, 9_2), \ \ (15_1, 15_2, 16_1, 16_2, 17_1), \ \ (13_2, 14_1, 14_2, 17_2)).$$

Note that the above schedule is not an RRR schedule since the segments 7_1 and 7_2 appear more than once in the schedules. Moreover, it is not even a perfect schedule. The gaps for these segments are 6, 6, and 7. However, except for some of the schedules that we just report their performance, all of the results of this paper are based on RRR schedules.

4 Asymptotic Results for One Movie

We first define the RR^2 Algorithm for $h = 1$ and $m = 1$, whose maximum delay asymptotically approaches

$1/(e-1)$ when the number of segments tends to infinity ($s \to \infty$). Then by iterating the RR^2 Algorithm h times, we approach the lower bound $1/(e^h - 1)$ for any $h \geq 1$ and $m = 1$.

4.1 One channel and one movie. Informally, the RR^2 Algorithm constructs a tree representation of an RR^2 schedule. The root has Δ subtrees for some $\Delta \geq 1$. Then in a greedy fashion, starting with some segment $x \geq \Delta$, the algorithm assigns as many as possible segments to each one of the Δ subtrees. More formally, the input to the algorithm are the two parameters Δ and $x \geq \Delta$. The root of the tree has Δ subtrees denoted by $T_1, \ldots, T_\Delta$. Let x_i be the first segment assigned to T_i, in particular $x_1 = x$. Let $\Delta_i = \lfloor x_i/\Delta \rfloor$ be the degree of subtree T_i and let $y_i = x_i + \Delta_i - 1$ be the last segment assigned to subtree T_i. Then the segments $x_i, \ldots, y_i$ are assigned as leaves of the subtree T_i. Finally, let $y(x, \Delta) = y_\Delta$ be the last segment assigned to the tree.

The following lemma asserts the validity of the schedule produced by the RR^2 Algorithm.

LEMMA 4.1. *The window size of any segment z is at most z for $x \leq z \leq y(x, \Delta)$.*

Proof. By the definition of trees as schedules, it follows that the window size of all the segments in T_i is $\Delta \cdot \Delta_i$. In the algorithm, Δ_i was set such that $x_i \geq \Delta \cdot \Delta_i$. The lemma follows since $z \geq x_i$ for all $z \in T_i$.

Figure 2 illustrates the resulting trees for two examples with small $\Delta = 3$ and $x = 8$ or $x = 9$. In the left side the range is [8..16] and in the right side the range is [9..20]. These values for x and y imply delay $8/(16 - 8 + 1) = 8/9$ for the shifted scheduled represented by the left tree and delay $9/(20 - 9 + 1) = 3/4$ for the shifted schedule represented by the right tree. The number of segments is 9 for the left tree and 12 for the right tree. Thus, by increasing the number of segments by 33%, we improved the delay by more than 15%.

The following lemma gives us a bound on the minimal number of scheduled segments.

LEMMA 4.2. *Let $y(x, \Delta)$ be the last segment assigned by the RR^2 Algorithm on input Δ and $x \geq \Delta$, then*

$$y(x, \Delta) \geq \left(1 + \frac{1}{\Delta}\right)^{\Delta} (x - \Delta) + \Delta - 1 \tag{4.3}$$

Proof. Let $y_0 = x - 1$. By the algorithm, $y_1 = x_1 + \lfloor x_1/\Delta \rfloor - 1$. To eliminate the floor, we get $y_1 \geq x_1 + (x_1/\Delta) - 2 = y_0 + ((y_0 + 1)/\Delta) - 1$. More generally,

$$y_k \geq y_{k-1} + \frac{y_{k-1} + 1}{\Delta} - 1 = y_{k-1}\left(1 + \frac{1}{\Delta}\right) + \frac{1}{\Delta} - 1 \ .$$

The solution to this recursive inequality is

$$y_k \geq \left(1 + \frac{1}{\Delta}\right)^k (x - \Delta) + \Delta - 1 \ .$$

Since $y(x, \Delta) = y_\Delta$, inequality (4.3) follows.

By Lemma 3.1, it follows that the range $[x..y(x, \Delta)]$ of the RR^2 Algorithm implies a maximum guaranteed delay $x/(y(x, \Delta) - x + 1)$. Combined with Lemma 4.2, we get that this delay is asymptotically optimal.

THEOREM 4.1. *There is a positive constant c_1 such that for any Δ and $x \geq 2\Delta^2$, the RR^2 Algorithm has maximum delay bounded above by* $\left(1 + \frac{c_1}{\Delta}\right)\left(\frac{1}{e-1}\right)$.

4.2 Many channels and one movie. For $h > 1$, we iterate the RR^2 Algorithm as follows. For the first channel, we schedule the range $[x..y(x, \Delta)]$. For the second channel we schedule the the range $[y(x, \Delta) + 1..y(y(x, \Delta) + 1, \Delta)]$, and so on until h channels are used. More formally, define $y_0(x, \Delta) = x - 1$ and $y_i(x, \Delta) = y(y_{i-1}(x, \Delta) + 1, \Delta)$ for $i > 0$. In the Iterated RR^2 Algorithm, for each i, $1 \leq i \leq h$, we schedule the range $[(y_{i-1}(x, \Delta) + 1)..y_i(x, \Delta)]$ on channel i using the RR^2 Algorithm. In total, the Iterated RR^2 algorithm schedules the range $[x..y_h(x, \Delta)]$ on h channels.

LEMMA 4.3. *Let $y_h(x, \Delta)$ be the last segment assigned by the Iterated RR^2 Algorithm on input Δ and $x \geq \Delta$ on h channels, then*

$$y_h(x, \Delta) \geq \left(1 + \frac{1}{\Delta}\right)^{h\Delta} (x - \Delta) + \Delta - 1 \tag{4.4}$$

By lemma 3.1, it follows that the range $[x..y_h(x, \Delta)]$ of the Iterated RR^2 Algorithm implies a maximum delay $x/(y_h(x, \Delta) - x + 1)$. We see in the following theorem, using Lemma 4.2, that this delay is asymptotically optimal.

THEOREM 4.2. *For each $h \geq 1$, there is a positive constant c_h such that for any Δ and $x \geq 2\Delta^2$, the Iterated RR^2 Algorithm has maximum delay bounded above by*

$$\left(1 + \frac{c_h}{\Delta}\right)\left(\frac{1}{e^h - 1}\right) . \tag{4.5}$$

Theorem 4.1 and Theorem 4.2 imply that for a given $h > 0$, we can approach the lower bound as close as we wish for s tends to infinity. Previous results approach the lower bounds only for $h \to \infty$ ([1]). In Section 6, we will demonstrate empirically that the RR^2 Algorithm yields almost optimal results for reasonable Δ and x much smaller than $2\Delta^2$.

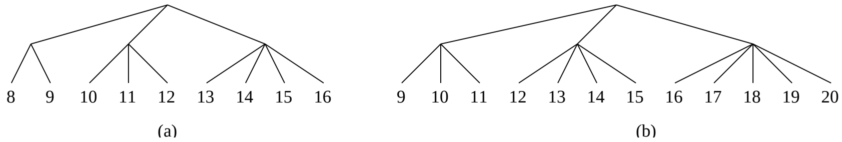

Figure 2: The RR^2 algorithm for (a) $\Delta = 3$ and $x = 8$ and (b) $\Delta = 3$ and $x = 9$.

5 Algorithms for Many Movies

We now consider the general case of $h \geq 1$ and $m \geq 1$. There are two methods for generalizing the asymptotic result from the previous subsections. In the first method, we assume $h \geq m$ and do not use channel sharing. If $h = \lfloor \rho \rfloor m + r$ with $0 \leq r < m$, then we allocate $\lfloor \rho \rfloor + 1$ channels to each of r movies and $\lfloor \rho \rfloor$ channels to each of the remaining $m - r$ movies. RR^2 Algorithm is then applied independently for each movie on its allocated channels.

In the second method, we generalize the RR^2 algorithm in a way that allows channel sharing. Here, we do not require $h \geq m$. In this generalization for Δ and $x \geq \Delta$, we first assign m copies of x, then m copies of $x + 1$, and so on, in the same greedy fashion as in the RR^2 Algorithm. We will see section 6 that the channel sharing method yields excellent results for small values of h and m.

The maximum delay we get from using this method approaches the lower bound given in Theorem 2.2 for $m > 1$. Specifically,

THEOREM 5.1. *For $h, m \geq 1$, there is a positive constant $c_{h,m}$ such that for any Δ and $x \geq 2m\Delta^2$, the RR^2 Algorithm used with channel sharing has maximum delay bounded above by*

$$\left(1 + \frac{c_{h,m}}{\Delta}\right)\left(\frac{1}{e^{h/m} - 1}\right). \tag{5.6}$$

5.1 Optimizing the ratio $\rho = h/m$. In this section we consider the second objective function, in which we fix the maximum delay guaranteed D and try to optimize h for a given m. That is, we are trying to minimize the value of ρ. This asymptotic result works primarily for large $m > 1$.

Assume first that $D = 1/s$. We wish to schedule the range $[1..s]$ to get a guaranteed delay D. If $m = LCM(1, \ldots, s)$, then we can allocate m/i channels to schedule all the i-th segments of all the m movies. We get $h = m \sum_{i=1}^{s} 1/i$. The problem of course, is that m is way too large. Note that we only used the channel sharing technique. We could get a better bound with the shifting technique but still for large values of m.

For small values of m, we define Algorithm RR. The basic idea behind the algorithm is that we schedule the segments of all movies on star trees that represent RR schedules. For a range $[x..y]$, we m times schedule segment x and then m times segment $x + 1$, and so on. At any point of the process, if the first segment to be scheduled is $[z]_i$ for $x \leq z \leq y$ and $1 \leq i \leq m$, then we create a new tree of degree z and schedule the next z segments in the order. The number of trees generated after the last segment y was scheduled is the output h of the algorithm.

Example: Let $D = 1/2$ and $m = 8$. The algorithm schedules 8 times all the segments in the range $[3..8]$. The resulting greedy schedule requires the following 10 channels:

$$C_1 : (3_1, 3_2, 3_3); \quad C_2 : (3_4, 3_5, 3_6); \quad C_3 : (3_7, 3_8, 4_1);$$
$$C_4 : (4_2, 4_3, 4_4, 4_5); \quad C_5 : (4_6, 4_7, 4_8, 5_1);$$
$$C_6 : (5_2, 5_3, 5_4, 5_5, 5_6); \quad C_7 : (5_7, 5_8, 6_1, 6_2, 6_3);$$
$$C_8 : (6_4, 6_5, 6_6, 6_7, 6_8, 7_1);$$
$$C_9 : (7_2, 7_3, 7_4, 7_5, 7_6, 7_7, 7_8);$$
$$C_{10} : (8_1, 8_2, 8_3, 8_4, 8_5, 8_6, 8_7, 8_8);$$

Indeed the delay implied by the range $[3..8]$ is $3/(8 - 3 + 1) = 1/2$. For this schedule, $\rho = 10/8 = 1.25$. The lower bound for ρ for this choice of D and m is $\ln(1 + 1/D) = 1.098$.

For a desired delay D and a fixed m, when we use Algorithm RR for the range $[x..y]$ where $y = m$ and $x = \left\lfloor \frac{D(m+1)}{D+1} \right\rfloor$, the resulting maximum delay approaches the lower bound from Theorem 2.2.

THEOREM 5.2. *For $m \geq 1$ and $0 < D \leq 1$, such that $mD > 1$, Algorithm RR uses h channels to transmit the m movies, each with maximum delay D, and*

$$\rho = \frac{h}{m} \leq \ln\left(1 + \frac{1}{D}\right) + O\left(\frac{1}{m}\log m\right). \tag{5.7}$$

6 Practical Values of h, m and s

The results in Section 4 imply that the shifting and the channel sharing techniques can be used to achieve nearly optimal schedules for instances with many movies or channels or when the segmentation is not limited (i.e., with very large s). In this section we show that these techniques yield very good schedules already for practical systems, with small values of h, m and s. When the two techniques are combined, the resulting performance is close to the lower bound even for very small values of h, m and s. To show this, we implemented Algorithms RR^1 and RR^2 that achieved our asymptotic results. We also adjusted the greedy algorithm presented in [1] to support the two techniques. In Section 6.1, we discuss the simulation results for some important and interesting cases: (i) The case of $h = 1$ and $m = 1$, in which clients receive data only from one channel. This case studies the performance of the shifting technique alone. (ii) The case of $\rho = 1$, that is, there is one channel per movie. (iii) The case of $\rho = k$ for $k = 2$ and $k = 1/2$. Finally, for specific small values of h and m, we designed schedules that beat the greedy algorithms and the currently known best schedules. In Section 6.2 we present some of these results.

6.1 Simulating RR^1, RR^2, and the greedy algorithm. Algorithm RR^1 simply assigns the segments to the channels sequentially. For a given x, it first schedules m segments with window requirement x, then m segments with window requirement $x+1$ and so on until it runs out of channels. When implementing algorithm RR^2 (described in Section 4.1), we check for a given x all the possible values for Δ in the range $2, \ldots, x/2$. We repeat this for h channels. Different channels may have different number of subtrees. For each channel, we search for the optimal Δ according to the value, $z \geq x$, of the first segment to be assigned to this channel.

Our greedy algorithm is based on the one presented in [1] for the harmonic window scheduling problem. The key ideas are: (i) Unlike RR and RR^2, we *do not* fill the channels sequentially. For each segment we select the channel that suits it most. This channel is selected such that the lost bandwidth (which is the gap between the granted and the required one) is minimized, and the potential to grant the exact window to following segments remains as high as possible. (ii) For a given x we first schedule m segments with window request x, then m segments with window request $x + 1$ and so on until we fail assigning some segment.

The simulation results for the case $m = h = 1$ are given in Figure 3. Algorithm RR^1 gains nothing from shifting. However, both RR^2 and *Greedy* are within 1.3-ratio from the lower bound even for $s = 8$. With 120 segments (which is on average movies one minute per segment) the ratio of both is about 1.13.

In Figure 4 we give the simulation results for the case $\rho = 1$ in which the system allocates one channel per movie. Recall that in traditional broadcasting protocols, the maximal delay is L, since we cannot do better then broadcasting repeatedly the movie on the single channel dedicated to it. This delay is reduced by both RR^2 and *Greedy* to 0.8 and 0.75 even for 2 and 3 movies. When compared with the lower bound, RR^2's delay for two movies and 9 segments is within 1.3-ratio from the lower bound. The same ratio is achieved by greedy even with 4 segments. A ratio of 1.105 is achieved for $s = 14$ and $m = 13$. We note that the results of RR^2 are compared well with the results of *Greedy* even-though RR^2 is much simpler. For large number of segments (40 or more) RR^2 is even slightly better.

The simulations for the cases $\rho = 2$ and $\rho = 1/2$ show, again, that both *Greedy* and RR^2 yield very good results even for very small m and s. To sum up, in all the studied cases, the simulations reveal that our techniques, alone or combined, can be applied by very simple algorithms. These algorithms were shown to produce good schedules already for small values of s and m. Even the simplest algorithm, RR^1, can be used to achieve good schedules. In fact, as m grows, the schedules produced by RR^2 are identical to the RR^1 schedules. Independent of ρ, the delay approaches the lower bound as the number of segments, or the number of sharing movies increases.

6.2 Some records for small values of h and m. In Table 1 we present the best schedules we found using the shifting technique for a single movie. Most of our results were obtained thinking in terms of the tree representation for RRR schedules which proved to be an efficient tool in designing schedules (see [1]). We note that all of these schedules outperform those created by the any of the algorithms mentioned in this paper. Due to space limitation we do not present the schedules themselves, only the resulting delay.

In Table 2, we present the best schedules we found by combining shifting and channel sharing. As can be seen, even for two or three movies and small number of segments the resulting delay is reduced significantly compared to the currently best known schedules.

7 Additional Models and Open Problems

First we show how our techniques can be used to obtain better results in various models sometimes with a different objective function to optimize. Here, we shortly discuss these models. and do not present the schedules that illustrate these results.

- For the case where the objective is to reduce the *average* delay, we developed another technique of *swapping* transmissions of paired segments. In each such pair of segments there is one with a too-wide window and another that is broadcast too often. The average delay can be improved using shifting,

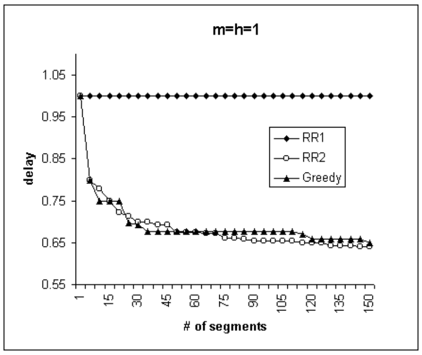

Figure 3: Simulation results for $m = h = 1$. The lower bound is $D \geq 0.582$.

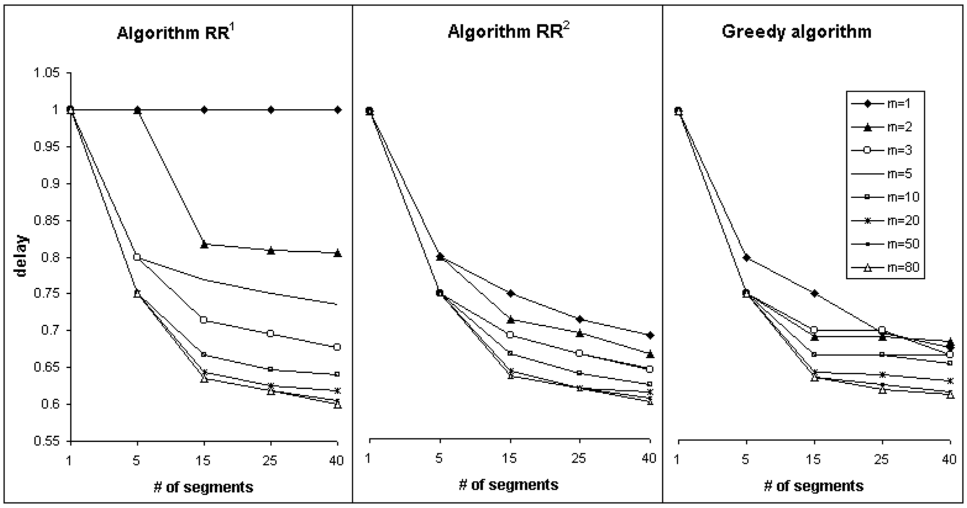

Figure 4: Simulation results for $\rho = 1$. The lower bound is $D \geq 0.582$.

	$h=1, \quad D \geq 0.582$									
$x..y$ range	1..1	4..8	6..13	10..23	12..28	16..38	24..59	36..91	48..122	75..194
s	1	5	8	14	17	23	36	56	75	120
delay	1	0.8	0.75	0.715	0.707	0.696	0.667	0.643	0.64	0.625

	$h=2, \quad D \geq 0.157$					$h=3, \quad D \geq 0.0524$			
$x..y$ range	4..22	5..29	10..63	15..98	24..160	2..28	3..45	4..63	8..134
s	19	25	54	84	137	27	43	60	127
delay	0.211	0.2	0.185	0.179	0.175	0.074	0.07	0.067	0.063

Table 1: Some records for small values of h and $m = 1$

ρ	1/2	1	2	
m, h	$2, 1$	$2, 2$	$2, 4$	$3, 6$
$s : x..y$ range	5 : 9..13	4 : 3..6	9 : 2..10	15 : 3..17
previous record	2	1	0.333	
our delay	1.8	0.75	0.222	0.2
lower bound	1.542	0.582	0.157	

Table 2: Some records for small values of h and m

alone or combined with swapping.

- We consider the *receive-r model.* In this model, clients can buffer data from channels but there is a limit, $r \leq h$ on the number of channels from which a client can read simultaneously. Moreover, we adopt the model suggested in [12] in which a client is forced to receive the data from the channels in order. That is, the client first listen to channels 1 to r; once it received all it needs from channel 1, it moves one data receiver to channel $r+1$ and so on. For example, consider the case $h = 3, r = 2$ and $m = 1$. The best known schedule without shifting has delay $1/7 = 0.1429$. With shifting, we achieve delay $2/17 = 0.1176$ with $s = 17$, and delay $3/27 = 0.1111$ with $s = 27$.

- We address the case in which movies may have different lengths. The main problem with this case is that we must make sure that lengths of segments are the same if we want to use the channel sharing technique.

- Finally we consider the case in which movies may have different popularity and hence different desired maximal delay guaranteed.

We conclude this section with some open problems.

- We are still looking for the best results for practical values of h and m and s.

- We believe that in practice there exist better algorithms that would outperform our greedy algorithms.

- For all of the above mentioned models: average delay, the receive-r model, different length movies, and different popularity movies, we do not have asymptotic upper bounds and lower bounds.

- We feel that we did not fully exploit the swapping technique. We also developed another technique, called the alignment technique, that we did not use in this paper. We can show that we can improve the performance by some alignment among the channel schedules especially for the average delay objective function.

- The focus of this paper was to guarantee an uninterrupted playback. In future research, we plan to address interactive MoD, in which clients can have operations like fast forward and rewind.

References

[1] A. Bar-Noy and R. E. Ladner. Windows Scheduling Problems for Broadcast Systems. In *Proc. of the 13-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* 433–442, January, 2002.

[2] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly Optimal Perfectly-Periodic Schedules. In *Proc. of the 20-th ACM Symposium on Principles of Distributed Computing (PODC)*, 107–116, 2001.

[3] S. W. Carter, D. D. E. Long and J. Pâris. Video-on-Demand Broadcasting Protocols. In *Multimedia Communications: Directions and Innovations (J. D. Gibson, Ed.)*, Acad. Press, San Diego, 179–189, 2000.

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal*, Vol. 4(3), 112-121, 1996.

[5] L. Engebretsen and M. Sudan. Harmonic Broadcasting is Optimal. In *Proc. of the 13-th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[6] K. A. Hua, Y. Cai, and S. Sheu Exploiting client bandwidth for more efficient video broadcast. In *Proc. of the 7-th International Conference on Computer Communication and Networks (ICCCN)*, 848-856, 1998.

[7] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Trans. on Broadcasting*, Vol. 43, No. 3, 268-271, 1997.

[8] L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Trans. on Broadcasting*, Vol. 44(1), 100-105, 1998.

[9] J. Pâris. A simple low-bandwidth broadcasting protocol for video-on-demand. In *Proc. of the 8-th International Conference on Computer Communications and Networks (IC3N)*, 118-123, 1999.

[10] J. Pâris, S. W. Carter, and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. In *Proc. of the 7-th Int. Conf. on Computer Communications and Networks (IC3N)*, 690-697, 1998.

[11] J. Pâris, S. W. Carter, and D. D. E. Long. A hybrid broadcasting protocol for video on demand. In *Proc. of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, 317-326, 1999.

[12] J. Pâris and D. D. E. Long. Limiting the receiving bandwidth of broadcasting protocols for video-on-demand. In *Proc. Euromedia Conf.*, 107-111, 2000.

[13] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Multimedia Systems*, Vol. 4(3), 197-208, 1996.