Scheduling Techniques for Media-on-Demand*

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

March 2, 2006

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. This paper addresses scheduling problems arising from these new systems for media-on-demand. For given amount of bandwidth, we reduce the maximal 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 both 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 solutions that are better than those achieved by our algorithms. Finally, we show that our techniques are useful for models in which clients may not receive data from all the channels, and are applicable to media with different lengths and popularities.

*A preliminary version appeared in the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 791–800, 2003.

†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.*

§School of Computer Science, The Interdisciplinary Center, Herzliya, Israel, E-mail: *tami@idc.ac.il.* Work done while the author was at the University of Washington.

1 Introduction

Media-on-demand (MoD) is the demand by clients to read, listen, or view various types 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 reduce the maximal start-up delay for MoD systems that support uninterrupted service. Our main objective is to achieve the smallest maximal 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. The latter can support many clients but cannot guarantee immediate service. This paper improves the tradeoffs between the system resources for broadcast systems and the maximal 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 time units, the maximal start-up delay is L/h units by starting a new transmission every L/h time units.

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 ([20]). Many variants and generalizations of the Pyramid Scheme followed and all of them showed this dramatic improvement by employing the following schedule design principle: *Early segments should be broadcast more frequently than later segments.* We adopt the following discrete model that is described shortly in order to demonstrate the ideas behind the Pyramid Scheme and our techniques.

The system has h channels and broadcasts m movies. Unless specified otherwise, assume that all the m movies have the same length, L, normalized to be one time unit ($L = 1$). Each movie is partitioned into s segments of equal length. The segments are indexed 1 to s in the order they should be viewed. The segments of the movies may be broadcast in any order on any channel. Assume that it takes one time slot to transmit or view a segment and thus, the length of the time slot is $1/s$. Assume further 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. Therefore, clients buffer or view segment i the first time they can do so after their arrival time. Clients may buffer a number of segments before the viewing process begins. We note, that most of the previous results assumed that the buffering and the viewing processes must start together.

The problem thus becomes a scheduling problem. To guarantee a maximal 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.

[1]For convenience, we use the terminology of movies in Video-on-Demand (VoD).

Example I: To demonstrate the usefulness of this principle consider the case of one movie of length 1 time unit and two channels. The traditional solution of broadcasting the movie every 1/2 unit of time guarantees a maximal start-up delay of 1/2. The following cyclic schedule is better:

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

In this schedule, the movie is partitioned into three segments each of length 1/3. 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, and 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. Note that in this case there is no need to buffer the third segment. In both cases the delay is bounded by the length of a segment which is 1/3.

One of our techniques is based on the observation that if clients may start buffering segments before they start viewing the movie then to guarantee a maximal 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 illustrated in the next example, buffering by itself is enough to reduce 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 1 and one channel. The traditional solution broadcasts the movie repeatedly to guarantee a start-up delay of at most 1. The following cyclic schedule, with $d = 4$, is better:

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

In this schedule, the movie is partitioned into 5 segments each of length 1/5. The first and the second segments are transmitted every 4 slots and each of the other 3 segments is 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 3 more slots the client starts viewing the movie. In any case, the client waits at most 4 slots which means a maximal start-up delay of 0.8. 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 segments 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] surveys 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 (e.g., [4]), through Pyramid-based broadcasting protocols, in which movies are partitioned into segments and different segments are broadcast on different channels [20], and finally Harmonic broadcasting protocols in which segment i is allocated bandwidth proportional to $1/i$ (e.g., [11]).

The papers [10, 12] present a simple schedule of one movie on h channels by partitioning the movie into $2^h - 1$ segments. Their schedule implies a maximal start-up delay of $1/(2^h - 1)$ for a movie of length 1. The Pagoda scheme [17] is based on a schedule for 3 channels with maximal start-up delay 1/9. It is then generalized to a schedule that asymptotically guarantees a start-up delay of at most $1/O(2.236^h)$. The new Pagoda scheme ([13]) deals with small values of h. Their maximal start-up delays for $h = 3, 4, 5, 6, 7$ are $1/9, 1/26, 1/66, 1/172, 1/442$ respectively. The Recursive Frequency-Splitting scheme ([19]) improves some of the results of the new Pagoda scheme. In particular, this scheme guarantees maximal start-up delays for $h = 3, 4, 5, 6, 7$ are $1/9, 1/26, 1/73, 1/201, 1/565$ respectively. This scheme is almost equivalent to the greedy scheme presented in [1]. The latter paper presented better results for small values of h. Their best maximal start-up delays for $h = 3, 4, 5, 6, 7$ are $1/9, 1/28, 1/77, 1/211, 1/570$ respectively.

The polyharmonic protocol [16] always forces the receiver to delay the same amount of time before beginning playback. Using channels of differing bandwidth enables a worst case delay to asymptotically approach $1/(e^b - 1)$ for total bandwidth b. Several papers [5, 7, 9] have shown this bound on delay to be optimal. The lower bound result has been recently extended and asymptotically optimal protocols have be presented for the case where the receiving bandwidth is less than the sending bandwidth [6].

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 start-up delay (which is at most $1/s$). A schedule based on this principle is shown to approach the lower bound as $h \to \infty$.

The papers [14, 15] also apply the observation that clients may start buffering segments before they start viewing the movie to achieve better results. However, they demonstrate the usefulness of this observation only for small examples. The first paper ([14]) presents superior results in which schedules achieve a shorter start-up delay for the same amount of bandwidth. On the other hand, the second paper ([15]) presents simpler schedules that can be applied to other variants such as when the receiving bandwidth is less than the sending bandwidth.

1.2 Our contributions.

In this paper, we apply two scheduling techniques that enable us to minimize the maximal 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. All the channels have the same bandwidth as the playback bandwidth.

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 exploits this greediness, we reduce the maximum start-up delay.

The second technique, called *channel sharing*, is based on the observation that usually servers broadcast more than one movie at a time. Let m be the number of movies transmitted by the server. In traditional schemes, about h/m channels are dedicated to each movie. Our idea is to broadcast segments of distinct movies on the same channel in order to reduce the maximum start-up delay. To analyze this technique, we extend the known lower bound for a single movie to a lower bound that depends on the parameter $\rho = h/m$.

The two techniques can be combined together to yield even better broadcasting protocols. We develop simple scheduling algorithms, called *recursive round-robin*, in which the two techniques are implemented. We present asymptotic analysis that shows the optimality of these algorithms. 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 $h \to \infty$ ([1]).

Simulation results imply that these techniques and our algorithms yield good schedules for small, practical, values of h and m. For specific small values of h, that are of much interest, we develop specific schedules, using both techniques, that outperform the best known schedules.

Finally, we show that these techniques are useful for models in which clients may not receive data from all the channels, and are applicable to media with different lengths and popularities.

1.3 Paper organization

Section 2 describes the model, provides lower bounds, and proves some preliminary results. Section 3 describes the shifting and channel sharing techniques. Section 4 presents the asymptotic results for one movie where the optimization goal is to minimize the delay for a given number of channels. Section 5 presents algorithms for many movies. Both the optimization goal of minimizing the delay for a given number of channels and the optimization goal of minimizing the required number of channels for a desired delay are considered. Section 6 demonstrates that the greedy algorithms that are implied by the asymptotic results yield good solutions for practical values for h and m. This section also reports some of our best results for small values of h and m. Section 7 shows that both techniques can be applied to the receive-r model, to systems with different length movies, and to movies with different popularities (and therefore different priorities and desired delays). Section 8 concludes with some open problems.

2 Model and Preliminaries

In this section we present the model and some preliminaries. In Section 2.1 we define the model and summarizes the notations. The known lower bound for one movie is generalized to many movies in Section 2.2. Finally, in Section 2.3 we define the special classes of schedules on which the asymptotic results are based.

2.1 Definitions and notations.

Assume a system with $h \geq 1$ channels each can broadcast segments of $m \geq 1$ movies. The quantity $\rho = h/m$ represents the average bandwidth per movie and can be less, equal, or greater than one. For convenience, it is assumed that all the m movies have the same length which is one unit of time. Clients may receive data from all the channels concurrently and store 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. Each movie is partitioned into $s \geq 1$ *segments*. The length of one segment which is the length of one *time slot* is $1/s$ time units ($1/s$ of the movie length). The z-segment of movie i for $1 \leq i \leq m$ and $1 \leq z \leq s$ is denoted $[z]_i$. When there is only one movie or when it is clear which is the movie, only z is used to denote the z-segment. When z is a specific number, the square brackets are omitted.

Let D denote the maximal start-up delay time to get an uninterrupted playback. For a given s, let $d = s \cdot D$ be the maximal *slot delay*. Note that D is measured by the length of the movie which is one time unit whereas d is measured by the length of a time-slot. In the broadcasting schemes we present, the maximum delay is given in units of time-slots, thus we assume that D is a multiple of $1/s$.

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 $1 \leq j \leq \ell$. The sequence is viewed as a cyclic schedule that generates an infinite schedule. That is, the segments $[[z_1]_{i_1}, \ldots, [z_\ell]_{i_\ell}]$ are 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. Define the *window size* of segment $[z]_i$ to be the maximum size of one of its gaps (with wrap-around). For example, in the schedule from Example II of the introduction,

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

segments 1 and 2 have window size of 4, while segments 3,4, and 5 have window size of 6. In this schedule, all the gaps between consecutive appearances of a specific segment are the same.

The next theorem states a necessary and sufficient condition for a schedule to guarantee a maximum start-up delay D for any one of the m movies.

Theorem 2.1 *Let $\mathcal{S}$ be a schedule that broadcasts $s \geq 1$ segments for each one of the $m \geq 1$ movies on $h \geq 1$ channels. Then $\mathcal{S}$ guarantees a maximum start-up delay $D > 0$ 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 $1 \leq i \leq m$ where $d = s \cdot D$.*

Proof: If there is a window of size $d+z$ that does not contain the z-segment of a particular movie, then a client arriving after the beginning and during the slot immediately prior to the window can not receive the beginning of the z-segment of this movie in time to view it even if it waits d time slots to start the viewing process. This proves the only if claim.

On the other hand, if all the gaps are bounded as stated in the theorem, a client that arrives at any time will have each segment either in its buffer or on one of the channel when it needs to view this segment. This is because a client may receive and/or store data from all the h channels concurrently. This proves the if claim. □

Objective functions: We distinguish between two types of optimization goals:

- Fix h and m and try to minimize D as a function of s. Moreover, try to approach the lower bound on D as s grows.
- Fix D and m and try to minimize h (or equivalently $\rho = h/m$) as a function of s. Moreover, try to approach the lower bound on ρ as m grows.

Indeed, both optimization goals are equivalent in the sense that an optimal algorithm for one of them can be converted to be optimal for the other. Nevertheless, for different algorithms, it is sometimes possible to come with a rigorous and/or elegant analysis only for one of them. Moreover, for practical values of m, h, and D, the algorithms are modified differently depending on the objective function, to achieve better results.

2.2 Lower bounds.

In [5], the following lower bound on the maximal start-up delay is shown for h channel and one movie.

Theorem 2.2 ([5, 7, 9]) *The start-up delay is at least $D \geq \frac{1}{e^h-1}$ for h channels and 1 movie.*

The generalization to the case of m movies is summarized in the following.

Theorem 2.3 *The start-up delay for h channels and m movies is at least $D \geq \frac{1}{e^{h/m}-1}$. Equivalently, $\rho = \frac{h}{m} \geq \ln\left(1+\frac{1}{D}\right)$ for m movies and maximal delay D.*

Proof: Assume that each movie contains B bits $1,\ldots,B$ for some large integer B. The slot-delay is measured by d bits and the general delay is therefore $D = d/B$. By Theorem 2.1, bit z must appear at least once in any window of size $d+z-1$. Hence, bit z requires at least $\frac{1}{d+z-1}$ of the bandwidth, and all the bits of one movie require at least bandwidth

$$f(B,d) = \sum_{z=1}^{B} \frac{1}{d+z-1} \ .$$

It follows that

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

For m movies, all the bits require bandwidth $m \cdot f(B,d)$ and therefore

$$h \geq m \cdot f(B,d) \geq m \ln\left(1+\frac{1}{D}\right) \ .$$

□

Note that in the above proof, we made the assumption that movies are composed of bits and that the smallest possible segment is one bit. This assumption is valid since the number of bits B could be as large as desired.

Examples: For one channel and one movie, the lower bound for the guaranteed start-up delay is $1/(e-1) \approx 0.582$. For four channels and one movie, the lower bound is $1/(e^4-1) \approx 0.0186$. If this lower bound can be achieved by a schedule, then clients will be able to view a one hour movie without interruption with start-up delay of at most 68 seconds.

2.3 The RRR class of schedules.

A schedule is called *perfect*, if all the gaps between any two appearance of a segment $[z]_i$ are equal for any segment $1 \leq z \leq s$ and movie $1 \leq i \leq m$. The asymptotic results in this paper are based on a class of perfect schedules that generalize the round-robin schedule denoted by RR. These schedules apply the round-robin schedule recursively and are denoted RRR (recursive round-robin). We give two ways to represent them using parentheses or trees (see also [2]).

An RR^0-schedule, represented by a single segment z, is one where every time slot is allocated to z. For $k > 0$, an RR^k-schedule is either an RR^{k-1}-schedule or the composition of Δ RR^{k-1}-schedules, for some $\Delta > 1$. In the latter case, the RR^k-schedule Z is represented by $(Z_1, Z_2, \ldots, Z_\Delta)$, where $Z_1, Z_2, \ldots, Z_\Delta$ are the representations of the RR^{k-1}-schedules. The schedule represented by $Z = (Z_1, Z_2, \ldots, Z_\Delta)$ is one where the slots are partitioned in a round-robin fashion into Δ sets. The slots of set i, $1 \leq i \leq \Delta$, are assigned, in a recursive manner, to the schedule represented by Z_i.

Example: The RR^3 schedule S represented by $(((A,B),(C,D,E)),F)$ is

$$[A\ F\ C\ F\ B\ F\ D\ F\ A\ F\ E\ F\ B\ F\ C\ F\ A\ F\ D\ F\ B\ F\ E\ F] \ .$$

We demonstrate this by unfolding the recursion. This schedule is a composition of the two RR^2-schedules $((A,B),(C,D,E))$ and F, where F appears in every other slot and the schedule represented by $((A,B),(C,D,E))$ appears in every other slot. The schedule represented by $((A,B),(C,D,E))$ is an RR^2-schedule which is a composition of the two RR^1 schedules (A,B) and (C,D,E). Thus, in the schedule S, the schedule represented by (A,B) appears in every fourth slot and the schedule represented by (C,D,E) appears also in every fourth slot. The RR^3-schedule S with $\Delta = 2$ is therefore identical to the RR^2-schedule $((A,B),F,(C,D,E),F)$ with $\Delta = 4$. Finally, the schedule represented by (A,B) is a composition of the two schedules A and B each appearing every 8 slots, and the schedule represented by (C,D,E) is a composition of the three schedules C, D, and E, each appearing every 12 slots. The total length of this schedule is 24.

A schedule S is called an RRR-schedule if there exists $k \geq 0$ such that S is an RR^k schedule. If an RRR schedule is an RR^k schedule but not an RR^{k-1}-schedule, then the *depth* of the schedule is k.

The tree representation of an RR^0-schedule is a single node marked with the segment z. The tree representation of an RR^k-schedule S which is a composition of the Δ RR^{k-1}-schedules $S_1, S_2, \ldots, S_\Delta$, is a tree whose root has a degree Δ, and the Δ subtrees are the tree representations of the schedules $S_1, S_2, \ldots, S_\Delta$. In particular, the tree representation of a traditional round-robin schedule (RR^1-schedule) is a star tree.

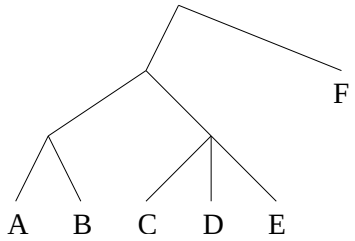

Figure 1: A tree representation of an RR^3-schedule.

Example: The RR^3 schedule S whose parenthesis representation is $(((A, B), (C, D, E)), F)$ can also be represented by the tree in Figure 1.

It is not hard to verify the following observations:

Observation 2.4

1. *Every RRR schedule is a perfect schedule.*
2. *The length of an RR^0-schedule is 1 and for $k > 0$, the length of an RR^k-schedule represented by $(Z_1, Z_2, \ldots, Z_\Delta)$ is $\Delta \cdot LCM(\ell_1, \ell_2, \ldots, \ell_\Delta)$ where ℓ_i is the length of the schedule represented by Z_i.*
3. *In a tree representation of an RRR schedule, the window size of segment z is the product of the degrees of all of its ancestors from the root to its parent.*

Detailed proofs of these observations can be found in [1, 2].

3 Schedule Designing Techniques

This section presents our two main techniques to design near-optimal schedules. The shifting technique is described in Section 3.1 and the channel sharing technique is described in Section 3.2. We illustrate these techniques with examples and also show in Section 3.3 how both techniques may be applied together.

3.1 The shifting technique

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[2]. As a result, the window size of the z-segment of any movie must be at most z. In particular, m channels must be dedicated for the first segments of the m movies. The maximum delay is the length of a segment since clients that arrived a bit after the starting time of a segment must wait almost a whole segment to get an uninterrupted playback.

For example, the optimal schedule for $h = 2$ and $m = 1$ is schedule (1) given in the introduction. Its guaranteed start-up delay is $1/3$ because $s = 3$. In general, *harmonic schedules* follow the scheduling design principle and for some range $[1..s]$, each segment, $1 \leq z \leq s$ 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-segment could be $d + z - 1$. Thus, the first segment can have window size d, and in general, the range, $[d..d + s' - 1]$ for $s' \geq s$ number of segments can be scheduled instead of the range $[1..s]$. A reduced delay occurs if $d/s' < 1/s$ for some $d > 1$.

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 *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 $z - x + 1$ in S'. The guaranteed start-up delay of a valid schedule for one movie is stated in the following lemma.

Lemma 3.1 *Let S be a valid 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: A client must wait x slots to get segment 1 that is 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 is $z + x - 1$. The lemma follows because there are $s = y - x + 1$ segments and therefore the length of each segment is $1/(y - x + 1)$. □

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

$$\begin{bmatrix} 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & \cdots \\ 3 & 6 & 7 & 3 & 8 & 9 & 3 & 6 & 7 & 3 & 8 & 9 & \cdots \end{bmatrix}$$

which is represented by $C_1 : (2, (4, 5))$ and $C_2 : (3, (6, 8), (7, 9))$. This schedule is valid for the range [2..9] because 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 shifted schedule on the range [1..8] is:

$$\begin{bmatrix} 1 & 3 & 1 & 4 & 1 & 3 & 1 & 4 & 1 & 3 & 1 & 4 & \cdots \\ 2 & 5 & 6 & 2 & 7 & 8 & 2 & 5 & 6 & 2 & 7 & 8 & \cdots \end{bmatrix}$$

which is represented by $C_1 : (1, (3, 4))$ and $C_2 : (2, (5, 7), (6, 8))$. In this shifted schedule the length of a segment is $1/8$ and the guaranteed start-up delay is the length of 2 segments which

[2]One exception is in polyharmonic broadcasting ([16]), where clients always buffer data before they start viewing. However, unlike our scheme, their channels have different bandwidths.

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 (see Example I in the introduction). Note that the delay time of the client is used in this scheme to buffer segments. For example, a client arriving before the fourth slot of the above schedule, buffers segments 2 and 4 while it is waiting for the broadcast of the first segment in the fifth slot.

Interestingly, the shifting technique can be used to reduce the maximal delay even for one channel. Without shifting, the best that can be done 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 until the next transmission starts. Consider the following valid schedule on the range [4..8]:

$$\left[\begin{array}{cccccccccccc} 4 & 6 & 5 & 7 & 4 & 8 & 5 & 6 & 4 & 7 & 5 & 8 & \cdots \end{array}\right]$$

which is represented by $((4,5),(6,7,8))$. This schedule is valid since 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]:

$$\left[\begin{array}{cccccccccccc} 1 & 3 & 2 & 4 & 1 & 5 & 2 & 3 & 1 & 4 & 2 & 5 & \cdots \end{array}\right]$$

which is represented by $((1,2),(3,4,5))$. By Lemma 3.1, the guaranteed start-up delay is $4/(8-4+1) = 0.8$ which is less than 1. This is exactly Example II of the introduction.

3.2 The channel sharing technique

In the *channel sharing technique*, segments of different movies may be scheduled 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 optimal schedule (without shifting) schedules the segments [1..9] of each movie. This implies a delay of 1/9 [13, 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 $i \in \{1, 2\}$.

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

3.3 Combining both technique

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 allocating two channels per movie (see Example I in the introduction). With the shifting technique alone, a valid [2..9]-schedule with delay at most 1/4 exists (see the example in Section 3.1). The following is a schedule of the range [3..17] for both movies, on 4 channels. By Lemma 3.1, the delay is at most $3/(17-3+1) = 1/5$. For this ratio of $\rho = h/m = 2$ the lower bound for the delay is $1/(e^2-1) \approx 0.156$ (theorem 2.3). We can approach this bound, but with more segments than 15.

Input: Integers $\Delta \geq 1$ and $x \geq \Delta$.
Output: A two level tree T that represents a valid RR^2 schedule on the range $[x..y(x,\Delta)]$.
The tree structure: $T_1, \ldots, T_\Delta$, the Δ star subtrees of the root.
Assignment to Tree T_1:
$x_1 = x$ minus the first number to be assigned to T_1.
$\Delta_1 = \lfloor x_1/\Delta \rfloor$ – the degree of T_1.
$y_1 = x_1 + \Delta_1$ – the last number to be assigned to T_1.
Assignment to Tree T_i for $1 < i \leq \Delta$:
$x_i = y_{i-1} + 1$ – the first number to be assigned to T_i.
$\Delta_i = \lfloor x_i/\Delta \rfloor$ – the degree of T_i.
$y_i = x_i + \Delta_i$ – the last number to be assigned to T_i.
Tree T_Δ: $y(x,\Delta) = y_\Delta$ – the last segment assigned to the tree T.

Figure 2: algorithm RR^2 for one channel and one movie.

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

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, most of the results of this paper are based on RRR schedules.

4 Asymptotic Results for One Movie

In this section we present asymptotic results for the case of one movie. The result for one channel ($h = 1$) is presented in Section 4.1 and the result for many channels ($h \geq 1$) is presented in Section 4.2. The results are based on the analysis of algorithm RR^2 that is described in this section. For $h = 1$ and $m = 1$, this algorithm guarantees a maximum delay that asymptotically approaches the lower bound $1/(e-1)$ when the number of segments tends to infinity ($s \to \infty$). Then by iterating the RR^2 algorithm h times, the guaranteed delay approaches the lower bound $1/(e^h - 1)$ for any $h \geq 1$ and $m = 1$. Our analysis shows that algorithm RR^2 approaches the lower bound for any fixed value of $h \geq 1$, whereas the best previous result approaches the lower bounds only for $h \to \infty$ ([1]).

4.1 One channel and one movie.

Algorithm RR^2 ($h = 1$ and $m = 1$): Informally, the 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 is the two parameters

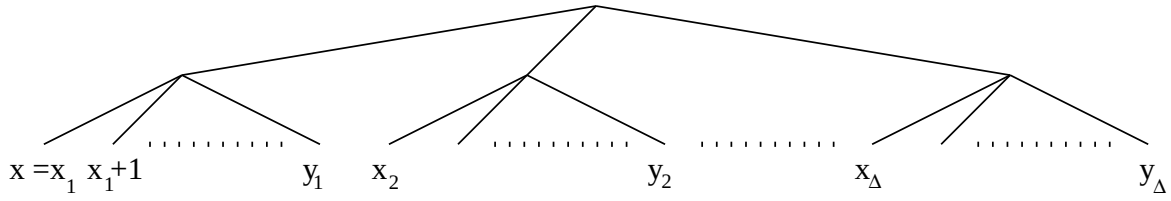

Figure 3: An illustration of how algorithm RR^2 schedules segments to the Δ subtrees.

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. A formal description of the algorithm is described in Figure 2 and an illustrative description appears in Figure 3.

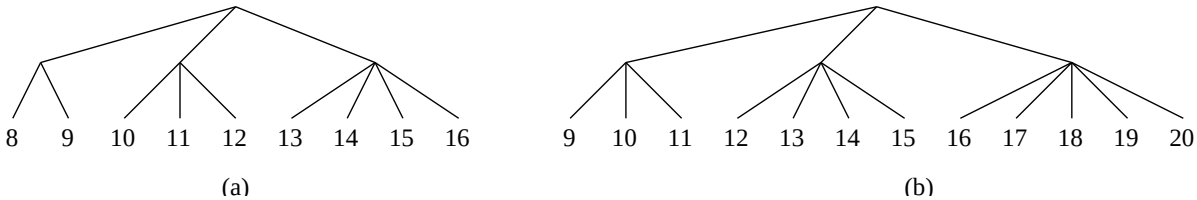

Figure 4: The RR^2 algorithm for (a) $\Delta = 3$ and $x = 8$ and (b) $\Delta = 3$ and $x = 9$.

Figure 4 illustrates the resulting trees for two examples with $\Delta = 3$ 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].

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: Part (3) of Observation 2.4 implies that the window size of all the segments in T_i is $\Delta \cdot \Delta_i$. In the algorithm, Δ_i is set such that $x_i \geq \Delta \cdot \Delta_i$. The lemma follows since $z \geq x_i$ for all $z \in T_i$. □

For example, In Figure 4, the 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%, the delay is improved by more than 15%.

For some larger values of Δ and x Algorithm RR^2 performs as follows. For $\Delta = 10$ and $x = 100$, the overall range is [100..255] which implies a guaranteed delay of $100/156 \approx 0.641$. For $\Delta = 20$ and $x = 400$, the overall range is [400..1065] which implies a guaranteed delay of $400/664 \approx 0.602$. We emphasize that, for a given Δ, the choice of x is very crucial. For example, if $x = 401$ for $\Delta = 20$, then the overall range is [401..1071] which implies a better guaranteed delay of $401/670 \approx 0.599$. Recall that the lower bound on the guaranteed delay is $1/(e - 1) \approx 0.582$. Hence, for 670 segments Algorithm RR^2 generates a schedule that is only 0.284% above optimal.

We now analyze Algorithm RR^2. The following lemma gives us a bound on the number of scheduled segments.

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

$$y(x,\Delta) \geq \left(1+\frac{1}{\Delta}\right)^{\Delta}(x-\Delta)+\Delta-1\ . \tag{3}$$

Proof: Let $y_0 = x-1$. By the algorithm, $y_1 = x_1 + \lfloor x_1/\Delta \rfloor - 1$ and to eliminate the floor,

$$y_1 \geq x_1 + (x_1/\Delta) - 2 = y_0 + ((y_0+1)/\Delta) - 1\ .$$

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 (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 show in the next theorem that this delay is asymptotically optimal.

Theorem 4.3 *There is a positive constant c_1 such that for any $\Delta \geq 2$ and $x \geq 2\Delta^2$, the RR^2 algorithm guarantees a maximum delay bounded above by*

$$\left(1+\frac{c_1}{\Delta}\right)\left(\frac{1}{e-1}\right). \tag{4}$$

Proof: Lemmas 4.2 and 3.1 imply that the maximum delay $x/(y(x,\Delta)-x+1)$ satisfies

$$\begin{aligned}\frac{x}{y(x,\Delta)-x+1} &\leq \frac{x}{\left(1+\frac{1}{\Delta}\right)^{\Delta}(x-\Delta)+\Delta-1-x+1}\\ &= \frac{1}{\left(\left(1+\frac{1}{\Delta}\right)^{\Delta}-1\right)\left(1-\frac{\Delta}{x}\right)}\ .\end{aligned}$$

By the well known fact that for $\Delta \geq 1$,

$$e \leq \left(1+\frac{1}{\Delta}\right)^{\Delta+1} \tag{5}$$

and the hypothesis that $x \geq 2\Delta^2$, it follows that

$$\begin{aligned}\frac{x}{y(x,\Delta)-x+1} &\leq \frac{1}{\left(\frac{e}{1+\frac{1}{\Delta}}-1\right)\left(1-\frac{1}{2\Delta}\right)}\\ &= \left(1+\frac{c_{1,\Delta}}{\Delta}\right)\left(\frac{1}{e-1}\right)\end{aligned}$$

where

$$c_{1,\Delta} = \frac{(3e-1)\Delta^2 - \Delta}{(2e-2)\Delta^2 - (e+1)\Delta + 1} \ .$$

This equality was derived and verified using Mathematica [21]. Since $c_{1,\Delta}$ has a finite limit as Δ goes to infinity there is a constant $c_1 \geq c_{1,\Delta}$ for all Δ. This c_1 satisfies the bound (4). □

4.2 Many channels and one movie.

Algorithm RR^2 ($h > 1$ and $m = 1$): Informally, for $h > 1$, Algorithm RR^2 for $h = 1$ is iterated as follows. For the first channel, the range $[x..y(x,\Delta)]$ is scheduled, for the second channel, the range $[(y(x,\Delta)+1)..y(y(x,\Delta)+1,\Delta)]$ is scheduled, 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, $1 \leq i \leq h$, the range $[(y_{i-1}(x,\Delta)+1)..y_i(x,\Delta)]$ is scheduled on channel i using Algorithm RR^2 for $h = 1$. In total, the iterated RR^2 algorithm schedules the range $[x..y_h(x,\Delta)]$ on h channels.

We now analyze the iterated RR^2 algorithm in a similar way we did for the basic RR^2 algorithm. The following lemma gives us a bound on the number of scheduled segments.

Lemma 4.4 *Let $y_h(x,\Delta)$ be the last segment assigned by the iterated RR^2 algorithm on input $\Delta \geq 2$ and $x \geq \Delta$ on h channels, then*

$$y_h(x,\Delta) \geq \left(1 + \frac{1}{\Delta}\right)^{h\Delta} (x-\Delta) + \Delta - 1 \ . \tag{6}$$

Proof: The proof is by induction on h. Lemma 4.2 covers the case $h = 1$. Assume the lemma is true for $h-1$. By Lemma 4.2 and since $y_{h-1}(x,\Delta)+1$ is the first segment to be scheduled on channel h, it follows that

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

By the induction hypotheses

$$\begin{aligned} y_h(x,\Delta) &\geq \left(1 + \frac{1}{\Delta}\right)^{\Delta} \left(\left(1 + \frac{1}{\Delta}\right)^{(h-1)\Delta} (x-\Delta) + \Delta - 1 + 1 - \Delta)\right) + \Delta - 1 \\ &= \left(1 + \frac{1}{\Delta}\right)^{h\Delta} (x-\Delta) + \Delta - 1 \ . \end{aligned}$$

□

By lemma 3.1, it follows that the range $[x..y_h(x,\Delta)]$ of the iterated RR^2 algorithm implies a maximum delay $x/(y_h(x,\Delta) - x + 1)$. Combined with Lemma 4.2, we show in the next theorem that this delay is asymptotically optimal.

Theorem 4.5 *For $h \geq 1$, there is a positive constant c_h that depends only on h such that for any $\Delta \geq 2$ and $x \geq 2\Delta^2$, the iterated RR^2 algorithm guarantees a maximum delay bounded above by*

$$\left(1 + \frac{c_h}{\Delta}\right)\left(\frac{1}{e^h - 1}\right). \tag{7}$$

Proof: Lemmas 4.4 and 3.1 imply that the maximum delay $x/(y_h(x,\Delta)-x+1)$ satisfies

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

Inequality (5) implies that

$$\frac{x}{y_h(x,\Delta)-x+1} \leq \frac{1}{\left(\frac{e^h}{\left(1+\frac{1}{\Delta}\right)^h}-1\right)\left(1-\frac{\Delta}{x}\right)}.$$

One can verify that $(1+1/\Delta)^h \leq 1+(2^h-1)/\Delta$. This inequality together with the assumption $x \geq 2\Delta^2$ imply that

$$\begin{aligned}\frac{x}{y_h(x,\Delta)-x+1} &\leq \frac{1}{\left(\frac{e^h}{1+\frac{2^h-1}{\Delta}}-1\right)\left(1-\frac{1}{2\Delta}\right)} \\ &= \left(1+\frac{c_{h,\Delta}}{\Delta}\right)\left(\frac{1}{e^h-1}\right)\end{aligned}$$

Where

$$c_{h,\Delta} = \frac{((2^{h+1}-1)e^h-1)\Delta^2-2^h\Delta+\Delta}{2(e^h-1)\Delta^2-(2^{h+1}+e^h-3)\Delta+2^h-1}\ .$$

This equality was derived and verified using Mathematica [21]. Since $c_{h,\Delta}$ has a finite limit as Δ goes to infinity there is a constant $c_h \geq c_{h,\Delta}$ for all Δ. This c_h satisfies the bound (7). □

Theorem 4.3 and Theorem 4.5 imply that for a given $h > 0$, Algorithm RR^2 approaches the lower when s tends to infinity. In Section 6, we demonstrate empirically that Algorithm RR^2 yields almost optimal results for "reasonable" values of segments where x is much smaller than $2\Delta^2$.

5 Algorithms for Many Movies

In this section we consider the general case of $h \geq 1$ and $m \geq 1$. In Section 5.1 we address the first optimization goal of minimizing the maximal delay D for a given number of channels h and in Section 5.2 we address the second optimization goal of minimizing the number of channels h for a given maximal delay D.

5.1 Minimizing the Maximal Delay for a Given h

In this section we present two methods to generalize the asymptotic result from the previous section. In Section 5.1.1 we describe the first method that dedicates channels to movies and does not use channel sharing and therefore $h \geq m$. In Section 5.1.2 we describe the second method that generalizes the iterated RR^2 algorithm to use the channel sharing technique as well.

5.1.1 Algorithm for many movies without channel sharing

Algorithm RR^2 ($h \geq m \geq 1$): Let $h = \lfloor h/m \rfloor m + r$ where $0 \leq r < m$. Then $\lfloor h/m \rfloor + 1$ channels are allocated to each of some r movies and $\lfloor h/m \rfloor$ channels are allocated to each of the remaining $m - r$ movies. The iterated RR^2 algorithm is then applied independently for each movie on its allocated channels.

The next theorem, which is an immediate corollary of Theorem 4.5, shows that the iterated RR^2 algorithm used without channel sharing is very close to optimal.

Theorem 5.1 *For $h \geq m \geq 1$, there is a positive constant $c_{h,m}$ that depends only on h and m such that for any $\Delta \geq 2$ and $x \geq 2\Delta^2$, the iterated RR^2 algorithm used without channel sharing guarantees a maximum delay bounded above by*

$$\left(1 + \frac{c_{h,m}}{\Delta}\right)\left(\frac{1}{e^{\lfloor h/m \rfloor} - 1}\right). \tag{8}$$

Proof: The iterated RR^2 algorithm without channel sharing allocates at least one channel to each movie since $h \geq m$. It allocates at most $h' = \lceil h/m \rceil$ channels to each movie due to the almost exact partitioning. By theorem 4.5, there is a constant $c_{h'}$ such that if $x \geq 2\Delta^2$ then each movie on its allocated channels has maximum delay bounded by $(1+c_h/\Delta)(1/(e^{\lfloor h/m \rfloor}-1))$. The theorem follows by setting $c_{h,m} = c_{h'}$. □

If m divides h, then the above bound is as good as the bounds of Theorems 4.3 and 4.5. When m does not divide h, it is tedious to compare the upper bound with the lower bound $\frac{1}{e^{h/m}-1}$. In the next subsection, we get a bound without the floor when m does not divide h and even when $h < m$.

5.1.2 Algorithm for many movies with channel sharing

In this subsection generalizes the iterated RR^2 algorithm in a way that allows channel sharing. Unlike the previous subsection, h can be smaller than m. In Section 6 we show that the channel sharing method yields excellent results for small values of h and m.

We first analyze this method for a single channel. Note that the straight forward broadcasting scheme simply schedules one entire movie after another to achieve maximum delay m, while the lower bound stated in Theorem 2.3 is $1/(e^{1/m} - 1)$.

Algorithm RR^2 ($h = 1$ and $m \geq 1$): The inputs to the algorithm are the parameters m, $\Delta \geq 2$, and $x \geq \Delta$. The algorithm constructs a two level tree where the degree of the root is Δ. The subtrees at the first level are indicated by $T_1, \ldots, T_\Delta$. The algorithm first assigns to the leaves of the tree m copies of segment x, then m copies of segment $x + 1$, and so on, in the same greedy fashion as in the basic RR^2 algorithm. If m copies of each of the segments $x, x+1, ..., y-1$ and $0 \leq m' < m$ copies of segment y are assigned to the subtrees $T_1, \ldots T_{i-1}$, then subtree T_i has $\lfloor y/\Delta \rfloor$ children and a y-segment is assigned to the first children of T_i. In the end, if m copies of each of the segments $x, x+1, ..., y-1$ and fewer than m copies of segment y are assigned to the subtrees $T_1, \ldots T_\Delta$, then the assigned range is $[x..y-1]$ and therefore the maximum delay is $x/(y-x)$ by Lemma 3.1.

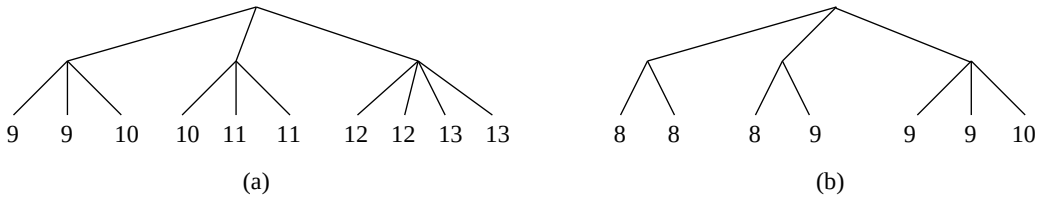

Figure 5: Algorithm RR^2 with channel sharing: (a) $m = 2,\ \Delta = 3,\ x = 9$; (b) $m = 3,\ \Delta = 3,\ x = 8$.

Figure 5 shows two cases of the RR^2 algorithm with channel sharing. In (a), $m = 2$ and the segments 9 through 13 are assigned exactly two times each, yielding a maximum delay of $9/(14-9) = 9/5 < 2$ (Note that 2 is the 'obvious' delay for $h = 1$ and $m = 2$). In (b), $m = 3$ the segment 10 only occurs once so it is eliminated. Hence, the maximum delay is $8/(10-8) = 4$ which is even inferior to the straightforward solution for $h = 1$ and $m = 3$.

Theorem 5.2 *For $h = 1$ and $m \geq 1$, there is a positive constant d_m that depends only on m such that for any $\Delta \geq 2$ and $x \geq 4\Delta^2 m(m+1)$, the RR^2 algorithm with channel sharing guarantees a maximum delay bounded above by*

$$\left(1 + \frac{d_m}{\Delta}\right)\left(\frac{1}{e^{1/m} - 1}\right). \tag{9}$$

Proof: Let x_i, y_i denote the first and last segments assigned to T_i. By definition, $x_1 = x$. It follows that $x_i \geq y_{i-1}$ and

$$y_i \geq x_i + \lfloor \lfloor x_i/\Delta \rfloor / m \rfloor \ .$$

This is because T_i has $\lfloor x_i/\Delta \rfloor$ children and the number of children that are assigned to distinct segments is at least the integer part of the number of children divided by the number of movies. Hence,

$$y_i \geq \left(1 + \frac{1}{\Delta m}\right) x_i - 1 + \frac{1}{m} \ .$$

In a proof similar to that of Lemma 4.2 the solution to this recursive inequality yields

$$y_\Delta \geq \left(1 + \frac{1}{\Delta m}\right)^\Delta (x - \Delta m - \Delta) + \Delta m + \Delta \ . \tag{10}$$

For all the m movies, the range $[x..y_\Delta - 1]$ is scheduled. Thus, the maximum delay is at most

$$\begin{aligned}
\frac{x}{y_\Delta - x} &\leq \frac{x}{\left(1 + \frac{1}{\Delta m}\right)^\Delta (x - \Delta m - \Delta) - x} \\
&= \frac{1}{\left(\left(1 + \frac{1}{\Delta m}\right)^\Delta - 1\right)\left(1 - \frac{\Delta m + \Delta}{x}\right) - \frac{\Delta m + \Delta}{x}} \\
&\leq \frac{1}{\left(\left(1 + \frac{1}{\Delta m}\right)^\Delta - 1\right)\left(1 - \frac{2\Delta m + 2\Delta}{x}\right)} \ .
\end{aligned}$$

Using the fact that for $0 \leq z \leq 1$, $e^z \leq (1 + z/\Delta)^{\Delta+1}$ and the assumption that $x \geq 4\Delta^2 m(m+1)$, it follows that

$$\frac{x}{y_\Delta - x} \leq \frac{1}{\left(\frac{e^{1/m}}{1 + \frac{1}{\Delta m}} - 1\right)\left(1 - \frac{1}{2\Delta m}\right)}$$

$$= \left(1 + \frac{d_{m,\Delta}}{\Delta}\right)\left(\frac{1}{e^{1/m} - 1}\right)$$

where

$$d_{m,\Delta} = \frac{(3e^{1/m} - 1)\Delta^2 m - \Delta}{(2e^{1/m} - 2)\Delta^2 m^2 - (e^{1/m} + 1)\Delta m + 1}$$

This equality was derived and verified using Mathematica [21]. Since $d_{m,\Delta}$ has a finite limit as Δ goes to infinity there is a constant $d_m \geq d_{m,\Delta}$ for all Δ. This d_m satisfies the bound (9). □

When $h > 1$, the RR^2 algorithm with channel sharing can be iterated to utilize multiple channels just as we did in section 4.2.

Algorithm RR^2 ($h \geq 1$ and $m \geq 1$): The input to the algorithm are the parameters h, m, $\Delta \geq 2$, and $x \geq \Delta$. Using the basic RR^2 algorithm with channel sharing, the algorithm first assigns to the leaves of the first tree out of the h trees m copies of segments $x, x+1, \ldots, y_1 - 1$ and fewer than m copies of y_1. The algorithm repeats this process on the second tree for the remaining copies of y_1 plus m copies each of $y_1 + 1, \ldots, y_2 - 1$ and fewer than m copies each of y_2, and so on. The assignment to each of the h trees is done according the basic RR^2 algorithm with channel sharing. In the end, if m copies of each of the segments $x, x+1, ..., y_h - 1$ and fewer than m copies of segment y_h are assigned to the h trees, then the assigned range is $[x..y_h - 1]$ and therefore the maximum guaranteed delay is $x/(y_h - x)$

Theorem 5.3 *For $h \geq 1$ and $m \geq 1$, there is a positive constant $d_{h,m}$ that depends only on m and h such that for any $\Delta \geq 2$ and $x \geq 4\Delta^2 m(m+1)$, the iterated RR^2 algorithm with channel sharing guarantees a maximum delay bounded above by*

$$\left(1 + \frac{d_{h,m}}{\Delta}\right)\left(\frac{1}{e^{h/m} - 1}\right). \tag{11}$$

Proof: Let x_i and y_i be the first and last segments, respectively, scheduled on the channel i. By definition, $x_1 = x$ and $x_{i+1} \geq y_i$. Inequality (10) implies the recurrence:

$$y_i \geq \left(1 + \frac{1}{\Delta m}\right)^{\Delta}(y_{i-1} - \Delta m - \Delta) + \Delta m + \Delta$$

where $y_0 = x$. Solving this recurrence yields

$$y_h \geq \left(1 + \frac{1}{\Delta m}\right)^{h\Delta}(x - \Delta m - \Delta) + \Delta m + \Delta\ .$$

Since m copies of $y_h - 1$ are scheduled, it follows that the maximum guaranteed delay is at most $x/(y_h - x)$. In the same fashion as the proof of theorem 4.5, the maximum delay is bounded above by

$$\frac{x}{y_h - x} \leq \frac{1}{\left(\left(1 + \frac{1}{\Delta m}\right)^{h\Delta} - 1\right)\left(1 - \frac{\Delta m + \Delta}{x}\right)}\ .$$

Inequality (5) implies that

$$\frac{x}{y_h - x} \leq \frac{1}{\left(\frac{e^{h/m}}{\left(1 + \frac{1}{\Delta m}\right)^{h/m}} - 1\right)\left(1 - \frac{\Delta m + \Delta}{x}\right)}\ .$$

Since $x \geq 4\Delta^2 m(m+1)$, it follows that

$$\frac{x}{y_h - x} \leq \frac{1}{\left(\frac{e^{h/m}}{\left(1+\frac{1}{\Delta m}\right)^{h/m}} - 1\right)\left(1 - \frac{1}{2\Delta m}\right)}.$$

Let $p = \lceil h/m \rceil$. The fact that and $(1 + 1/\Delta m)^{h/m} \leq 1 + (2^p - 1)/(\Delta m)$ implies that

$$\begin{aligned} \frac{x}{y_h - x} &\leq \frac{1}{\left(\frac{e^{h/m}}{1+\frac{2^p-1}{\Delta m}} - 1\right)\left(1 - \frac{1}{2\Delta m}\right)} \\ &= \left(1 + \frac{d_{h,m,\Delta}}{\Delta}\right)\left(\frac{1}{e^{h/m} - 1}\right) \end{aligned}$$

where

$$d_{h,m,\Delta} = \frac{((2^{p+1} - 1)e^{h/m} - 1)\Delta^2 m - 2^p\Delta + \Delta}{2(e^{h/m} - 1)\Delta^2 m^2 - (2^{p+1} + e^{h/m} - 3)\Delta m + 2^p - 1}\ .$$

This equality was derived and verified using Mathematica [21]. Since $d_{h,m,\Delta}$ has a finite limit as Δ goes to infinity there is a constant $d_{h,m} \geq d_{h,m,\Delta}$ for all Δ. This $d_{h,m}$ satisfied the bound (11). □

5.2 Minimizing the ratio $\rho = h/m$.

In this section we address the second objective function. Here, the maximum allowed delay D is fixed and the goal is to minimize the number of channels h for a given number of movies m. Equivalently, the goal is to minimize the value of $\rho = h/m$.

Assume first that $D = 1/s$ and that the goal is to schedule the range $[1..s]$ m times to get a guaranteed delay at most D. If $m = LCM(1, \ldots, s)$, then m/z channels can be allocated to schedule the z-segments of all the m movies for any $1 \leq z \leq s$. As a result, the number of required channels is $h = \lceil m \sum_{z=1}^{s} 1/z \rceil$ and $\rho \leq \ln(1/D) + 1$ since the harmonic number $H_s = \sum_{z=1}^{s} \frac{1}{z}$ is less than $1 + \ln s = 1 + \ln(1/D)$ (see e.g., [8] page 264, Eq. 6.66). This is already very close to the lower bound $\rho \geq \ln(1 + 1/D)$ from Theorem 2.3. The problem of course, is that m is way too large. Note that only the channel sharing technique was used in the above analysis. Nevertheless, using the shifting technique with a similar analysis would yield only a slight improvement and still m must be very large. The objective of the rest of this section is to fix m as well and seek a "good" upper bound on ρ. To that end, we define algorithm RR that is based on RR schedules.

Algorithm RR: The basic idea is similar to Algorithm RR^2 with channel sharing using one level trees (stars) that represent RR schedules (while two-level trees represent RR^2 schedules). Given a range $[x..y]$, first schedule m copies of segment x, then m copies of segment $x+1$, and so on. At any point of the process, when there is a need for a new channel (tree) and the next segment to be scheduled is $[z]_i$ for $x \leq z \leq y$ and $1 \leq i \leq m$, then create a new tree of degree z and schedule the next z segments. The set of h trees generated until the last segment y is scheduled is the output of the algorithm. A more formal description of the algorithm appears in Figure 6.

Input:

m – number of movies.

x – the first segment to be scheduled.

y – the last segment to be scheduled.

Output:

h RR schedules in which $[z]_i$ appears in an RR schedule whose length is at most z, for any $x \leq z \leq y$ and $1 \leq i \leq m$.

Ordering the segments:

$L = [x]_1, [x]_2, \ldots, [x]_m, [x+1]_1, \ldots, [x+1]_m, \ldots, [y]_1, \ldots, [y]_m$

The repeated scheduling step:

Let $[z]_i$ be the first segment in L.

Schedule the next z segments in L (or the rest of L if its length is less than z) on an RR schedule of length z and remove these segments from L.

Figure 6: Algorithm RR

Example: Let $D = 1/2$ and $m = 8$. Algorithm RR schedules 8 times each segment in the range [3..8]. The resulting schedule requires the following 10 channels:

$\mathbf{C_1} : (3_1, 3_2, 3_3)$; $\mathbf{C_2} : (3_4, 3_5, 3_6)$; $\mathbf{C_3} : (3_7, 3_8, 4_1)$; $\mathbf{C_4} : (4_2, 4_3, 4_4, 4_5)$; $\mathbf{C_5} : (4_6, 4_7, 4_8, 5_1)$;

$\mathbf{C_6} : (5_2, 5_3, 5_4, 5_5, 5_6)$; $\mathbf{C_7} : (5_7, 5_8, 6_1, 6_2, 6_3)$; $\mathbf{C_8} : (6_4, 6_5, 6_6, 6_7, 6_8, 7_1)$;

$\mathbf{C_9} : (7_2, 7_3, 7_4, 7_5, 7_6, 7_7, 7_8)$; $\mathbf{C_{10}} : (8_1, 8_2, 8_3, 8_4, 8_5, 8_6, 8_7, 8_8)$;

Indeed the delay implied by the range [3..8] is $3/(8-3+1) = 1/2$ (Lemma 3.1). For this schedule, $\rho = 10/8 = 1.25$ while the lower bound on ρ for this choice of D and m is $\ln(1+1/D) \approx 1.098$.

The idea behind the analysis of Algorithm RR is that for a given m and D, the range $[x..m]$ that guarantees the delay D is scheduled for all the m movies. This will assure that the window size given by the algorithm to segment $x \leq z \leq m$ is no less than $z - 1$. The latter property implies that the loss of bandwidth is small and therefore ρ is close to the lower bound.

Theorem 5.4 *Let $m \geq 1$ and $D > 0$ such that $m > 2 + 4/D$. Let $y = m$ and $x = \left\lfloor \frac{(m+1)D}{D+1} \right\rfloor$. Then algorithm RR that schedules m times the range $[x..y]$ guarantees a maximum delay D with h channels where*

$$\rho = \frac{h}{m} \leq \ln\left(1 + \frac{1}{D}\right) + \frac{O(1/D)}{m} \ . \tag{12}$$

Proof: First, we verify that the guaranteed delay for this choice of x and y is at most D. By Lemma 3.1, the guaranteed delay for each movie is $x/(y-x+1)$. Indeed,

$$\frac{x}{y-x+1} \leq D \iff x \leq \frac{(m+1)D}{D+1} \text{ and } y = m \ .$$

Next, we bound the number of channels used by Algorithm RR for m movies and the range $[x..m]$. The crucial argument is that when scheduling the m z-segments for any $x \leq z \leq m$, at

most $z-2$ z-segments are scheduled with window size $z-1$ while the remaining z-segments are scheduled with window size exactly z. To see this, note that $m > z-1$ and therefore, the $(z-1)$-segments need more than one tree. In particular, the last tree that contains a $(z-1)$-segment has degree $z-1$. In the worst case, this tree contains only one $(z-1)$-segment and its remaining $z-2$ leaves are assigned to z-segments. However, all the remaining z-segments are schedules on trees with degree z. This implies that the total bandwidth allocated to all the z-segments is at most $\frac{z-2}{z-1} + \frac{m-(z-2)}{z}$ (where the bandwidth of each channel is 1). Finally, an additional channel is added to the upper bound on h since the last channel might be partially used. Thus,

$$\begin{aligned} h &\leq 1 + \sum_{z=x}^{m} \left(\frac{z-2}{z-1} + \frac{m-(z-2)}{z} \right) \\ &= 1 + \sum_{z=x}^{m} \frac{m(z-1)+(z-2)}{z(z-1)} \\ &\leq 1 + \sum_{z=x}^{m} \frac{m+1}{z} = 1 + (m+1) \sum_{z=x}^{m} \frac{1}{z} \\ &\leq 1 + (m+1) \ln \frac{m}{x-1} \ . \end{aligned}$$

The last inequality is true since $\sum_{z=x}^{m} \frac{1}{z} \leq \int_{t=x-1}^{m} \frac{dt}{t} = \ln \frac{m}{x-1}$.

Recall that $x = \left\lfloor \frac{(m+1)D}{D+1} \right\rfloor$, thus $x - 1 \geq \frac{(m+1)D}{D+1} - 2 = \frac{(m-1)D-2}{D+1}$. Note that $x-1 > 0$ since $m > 2 + 4/D$ implies that $(m-1)D - 2 > 0$. Therefore,

$$\frac{m}{x-1} \leq \frac{m(D+1)}{(m-1)D-2} \ .$$

For $c = \frac{mD}{(m-1)D-2}$, it follows that $\frac{m}{x-1} \leq c\left(1 + \frac{1}{D}\right)$. Hence,

$$\ln \frac{m}{x-1} \leq \ln \left(1 + \frac{1}{D}\right) + \ln c \ .$$

Plugging this bound in the upper bound on h, it follows that

$$h \leq 1 + (m+1) \ln \left(1 + \frac{1}{D}\right) + (m+1) \ln c \ .$$

Equivalently,

$$\rho \leq \ln \left(1 + \frac{1}{D}\right) + \frac{1 + \ln c + \ln(1 + 1/D)}{m} + \ln c \ .$$

We now analyze $c = \frac{mD}{(m-1)D-2}$. It follows that $c = 1 + \frac{D+2}{(m-1)D-2}$ where the second additive term is less than 1 since $m \geq 2 + \frac{4}{D}$. First, this immediately implies that $\ln c < 1$. Second, because $\frac{p}{q} < \frac{p+r}{q+r}$ for $0 < p < q$ and $r > 0$, this also implies that

$$c \leq 1 + \frac{2D+4}{mD} = 1 + \frac{2 + 4/D}{m} \ .$$

The fact that $\ln(1+\alpha) \leq \alpha$ for $\alpha > 0$ implies that $\ln c \leq \frac{2+4/D}{m}$ and that $\ln(1+1/D) \leq 1/D$. Putting all the above bounds in the upper bound on ρ yields the following bound,

$$\rho \leq \ln\left(1+\frac{1}{D}\right) + \frac{4+5/D}{m} \ .$$

The theorem follows since $4 + 5/D = O(1/D)$. □

The following Corollary is implied by the above theorem and Theorem 2.3.

Corollary 5.5 *Let $D > 0$ and let $\rho_{opt} \cdot m$ be the number of channels required by the optimal algorithm that guarantees a delay D for m movies. Then for a large enough m, there exists a segmentation number $s \leq m$ for which Algorithm RR generates a schedule that guarantees a maximum delay D with h channels where*

$$\rho \leq \rho_{opt} + \frac{O(1/D)}{m} \ . \tag{13}$$

The above corollary implies that Algorithm RR performs well when the guaranteed delay D is fixed and the number of movies grows. In Section 6 we demonstrate that the algorithm performs well even for smaller values for m. One of the reasons for the performance discrepancy between the analysis and the simulations is that the analysis in the proof of Theorem 5.4 is loose and there are many possible improvements. The following are two examples: (i) When m is much larger than z then most of the z-segments are allocated bandwidth $1/z$ while in the analysis, it was assumed that most of them get a bandwidth $1/(z-1)$. (ii) If many z-segments get a $1/(z-1)$-bandwidth then many $(z-1)$-segments get the optimal $1/(z-1)$ bandwidth. We believe that the result of Corollary 5.5 is sufficient to demonstrate the near-optimality of Algorithm RR.

6 Practical Values of h, m and s

The results in Section 4 and Section 5 imply that the shifting and the channel sharing techniques can be used to achieve nearly optimal schedules for instances with many movies and multiple 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. 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 and RR^2 that achieve 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 the following 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 $\rho = 1$ in which there is one channel per movie. (iii) The cases of $\rho = 2$ and $\rho = 1/2$. In all cases, a better guaranteed delay is gained when the number of segments increases. Then another interesting study is reported in this section. The goal of this study is to minimize h for a given maximum delay guaranteed D, m movies, and limited segmentation number s. For this study, we simulated Algorithm RR^2 for $D = 3/4, 1/2$, and $1/3$.

Finally, for specific small values of h and m, we designed schedules that beat the currently best known schedules. In Section 6.2 we present some of these new records.

6.1 Simulating RR, RR^2, and the greedy algorithms.

Algorithm RR simply assigns the segments to the channels sequentially. For a given x, it first schedules m segments with window x, then m segments with window $x+1$ and so on until it runs out of channels. When implementing algorithm RR^2, all the possible values for Δ in the range $2, \ldots, x/2$ were checked. This process was repeated for h channels. Different channels may have different number of subtrees. For each channel, the optimal Δ is selected according to the value, $z \geq x$, of the first segment to be assigned to this channel.

The greedy algorithm is based on the one presented in [1] for the harmonic window scheduling problem. The key ideas are:

1. The segments are scheduled sequentially, that is, for a given x first schedule m segments with window x, then m segments with window $x+1$ and so on until there is no room for additional segments.

2. Unlike Algorithms RR and RR^2, the greedy algorithm *does not* fill the channels sequentially. For each segment, it selects the channel that is the "best fit". This channel is selected such that the lost bandwidth (which is the difference between the granted bandwidth and the required bandwidth) is minimized.

The simulation results for the case $m = 1$ and $h = 1$ are given in Figure 7. As expected in this case, Algorithm RR gains nothing from shifting. However, both RR^2 and *Greedy* are within 1.3-ratio from the lower bound already for $s = 8$. With 120 segments (which is one minute per segment for a typical movie) the ratio of both is about 1.13.

Figure 8 gives the simulation results for the case $\rho = 1$, in which the number of channels is equal to the number of movies. Recall that in traditional broadcasting protocols, the maximum guaranteed delay is 1 since the system cannot do better than broadcasting repeatedly each movie on the single channel dedicated to it. For 5 or fewer segments, the delay is reduced by both RR^2 and *Greedy* to 0.8 and 0.75 for 2 and 3 movies, respectively. When compared with the lower bound, the delay guaranteed of algorithm RR^2 for two movies and 9 segments is within 1.3-ratio from the lower bound. The same ratio is achieved by *Greedy* already with 4 segments[3]. We note that the results of Algorithm RR^2 are compared well with the results of Algorithm *Greedy* even though Algorithm RR^2 is much simpler. For large number of segments (40 or more) Algorithm RR^2 is even slightly better.

The case $\rho = 2$ is studied in Figure 9. For this case Algorithm *Greedy* approaches the lower bound already for very small m and s. For example, it is within 1.28 ratio from the lower bound for $m = 3$ and $s = 5$ and within 1.12 ratio from the lower bound for $m = 5$ and $s = 40$. The results for $\rho = 1/2$ are given in Figure 10. Note that in this case there is a single channel per two movies. Thus, in naive scheduling schemes, the maximum guaranteed delay is 2. Both

[3]These results are not reflected in Figure 8.

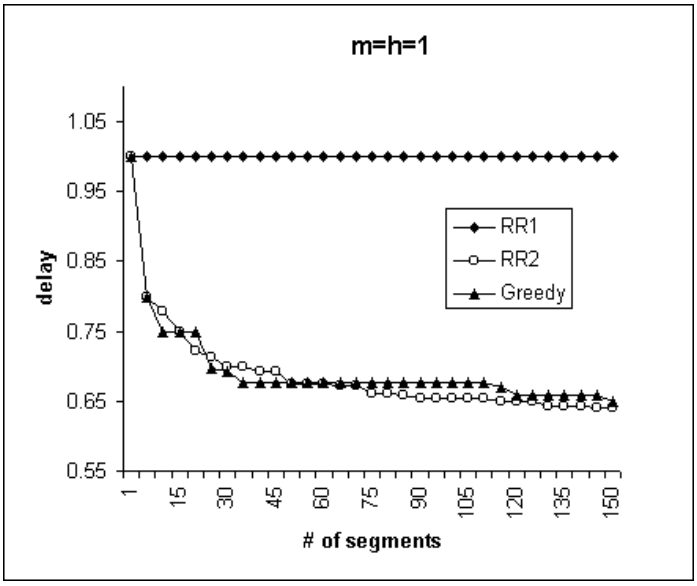

Figure 7: Simulation results for $m = h = 1$. The lower bound is $D \geq 0.582$.

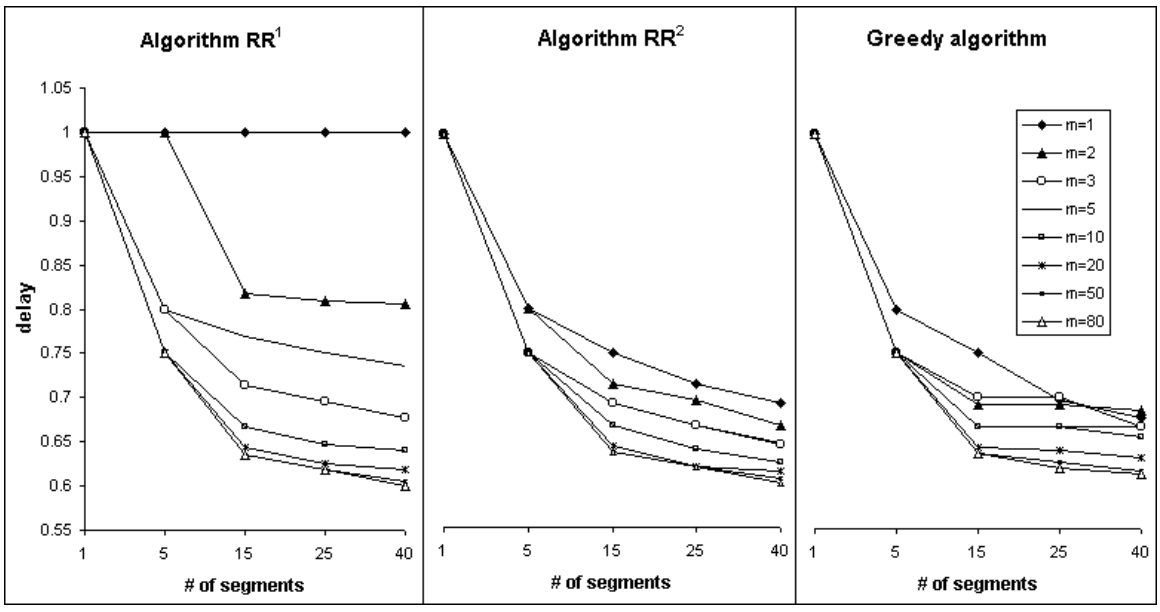

Figure 8: Simulation results for $\rho = 1$. The lower bound is $D \geq 0.582$.

	$delay = 3/4$			$delay = 1/2$			$delay = 1/3$		
m	$s \leq 5$	$s \leq 15$	$s \leq 40$	$s \leq 5$	$s \leq 15$	$s \leq 40$	$s \leq 5$	$s \leq 15$	$s \leq 40$
1	2.000	1.000	1.000	2.000	2.000	2.000	2.000	2.000	2.000
2	1.500	1.000	1.000	1.500	1.500	2.000	2.000	2.000	2.000
3	1.000	1.000	1.000	1.667	1.333	1.333	2.000	1.667	1.667
5	1.000	1.000	1.000	1.400	1.400	1.200	2.000	1.600	1.600
8	1.000	0.875	0.875	1.375	1.250	1.250	1.875	1.625	1.500
10	1.000	1.000	0.900	1.300	1.200	1.200	1.900	1.600	1.500
15	1.000	0.933	0.933	1.333	1.200	1.200	1.867	1.533	1.467
20	1.000	0.900	0.900	1.300	1.200	1.150	1.850	1.500	1.450
30	0.967	0.900	0.900	1.300	1.167	1.133	1.833	1.500	1.467

Table 1: ρ as a function of m, s for some fix delays (RR^2 Algorithm)

Algorithm RR^2 and Algorithm *Greedy* achieve delay 5/3 already for $h = 5$ and $s = 6$. This is within 1.08-ratio from the lower bound.

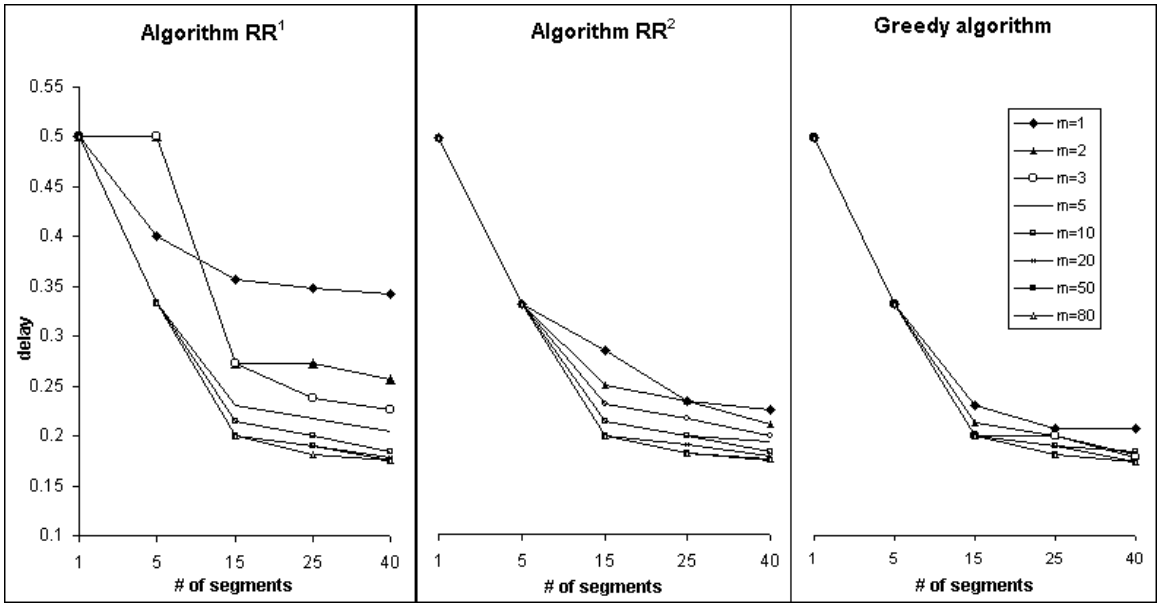

Figure 9: Simulation results for $\rho = 2$. The lower bound is $D \geq 0.1574$.

For the objective function of minimizing h (or ρ) for a given D, Algorithm RR^2 was analyzed for delays 3/4, 1/2, and 1/3. Without the shifting technique, the required number of channels per movie to achieve these delays are $\rho = 2, 2$ and 3 respectively. The required $\rho = h/m$ ratios for Algorithm RR^2 for some values of m and limited s are given in Table 6.1.

Some concluding remarks: The simulations reveal that both techniques, alone or combined, can be applied by very simple algorithms. These algorithms produce good schedules already for small values of s and m. Even the simplest algorithm, RR, can be used to achieve

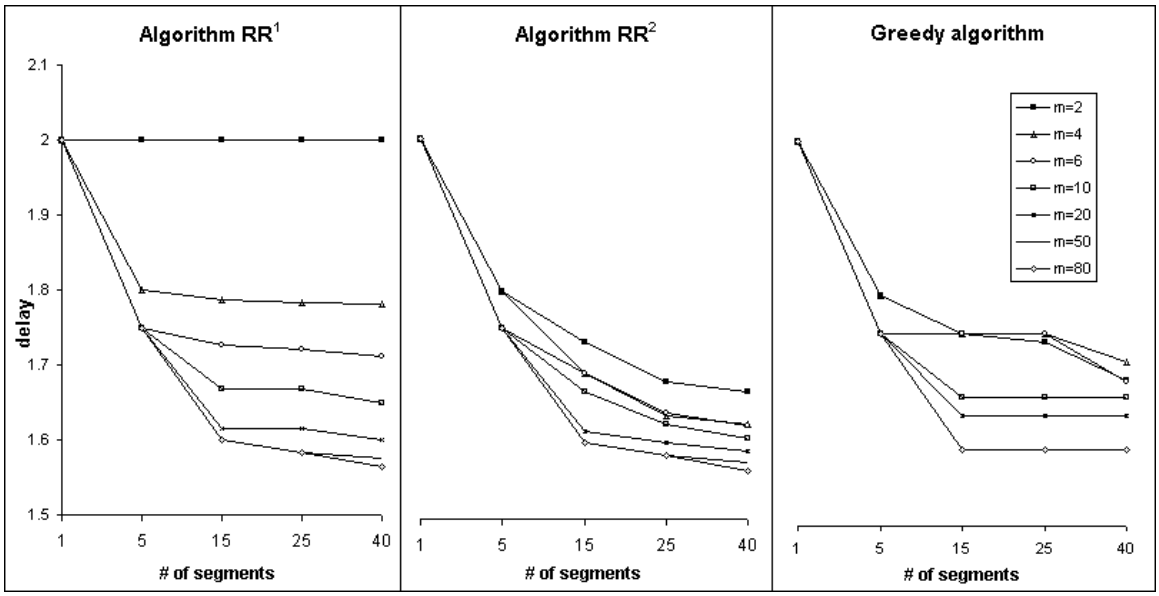

Figure 10: Simulation results for $\rho = 1/2$. The lower bound is $D \geq 1.542$.

good schedules. In fact, as m grows, the performance of the schedules produced by RR and RR^2 are similar. 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.

	$h=1, \quad D \geq 0.582$									
$x..y$ 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 2: Some records for small values of h and $m = 1$

In Table 2 we present the best schedules we found using the shifting technique for a single movie. Most of our results were obtained using the tree representation for recursive round-robin (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 Algorithms RR, RR^2, and *Greedy*. We do not present the schedules themselves, only the resulting guaranteed maximum delays.

Table 3 presents the best schedules we found by combining the shifting technique and the

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 3: Some records for small values of h and m

channel sharing technique. As can be seen, even for two or three movies and small number of segments, the resulting guaranteed delays are reduced significantly compared to the currently best known schedules.

7 Additional Models

This section shows how the shifting and the channel sharing techniques can be applied to obtain better results in some variants model on the basic model considered in this paper. We shortly discuss these models and demonstrate for each variant separately the usefulness of the techniques for some small values of h and m. We remark that improvements exist for some combinations of these models.

7.1 The receive-r model

In the receive-r 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 receive data simultaneously. All the results in this paper assumed that $r = h$. A more realistic assumption is that r is fixed while h grows. The case for which $r = 1$ is the tradition model. In this subsection, we show by example how our technique could be applied for the case $r < h$ as well. In our schedules a stronger assumption on the model is taken (see [18]) in which a client is forced to receive the data from the channels in order. That is, a client first listens to channels 1 to r; once it has received all it needs from channel i for $1 \leq i \leq h - r$, it listens to channel $r + i$ until it listens to the last r channels.

Assume a system with the parameters $m = 1$, $h = 3$, and $r = 2$. The following is the best known schedule without shifting that guarantees a delay of $1/7 \approx 0.1429$.

$$\mathbf{C_1} : 1; \quad \mathbf{C_2} : (2, (4, 5)); \quad \mathbf{C_3} : (3, (6, 7)) \ .$$

Note that since a client listens to channel C_3 only after it receives the first segment from channel C_1, the window size of a segment z in C_3 must be at most $z - 1$. Indeed, the window sizes of segments $3, 6, 7$ are $2, 4, 4$ respectively. With the shifting technique, we designed a schedule that guarantees a delay of $2/17 \approx 0.1176$ with $s = 17$ segments, and another schedule that guarantees a delay of $3/27 \approx 0.1111$ with $s = 27$ segments. These results indicate that similar tradeoffs exist for the receive-r model. The following is the latter schedule for the range [3..29].

$$\mathbf{C_1} : (3, (6, 7), (9, 10, 11))$$

$$\mathbf{C_2} : ((4, 5), ((8, (16, 17)), (12, 13, 14, 15)))$$

$$\mathbf{C_3} : ((18, 19, 20), (21, 22, 23, 29), (24, 25, 26, 27, 28))$$

Note that the schedule of channel C_1 has a cycle of length 9. Thus, only after 9 slots clients may receive segments from channel C_3. As a result, in a valid schedule for $r = 2$, the window size of any segment z in C_3 must be no more than $z - 9$. Indeed, this is the case in the above example.

Unfortunately, we have no lower bounds on the guaranteed delay for the receive r model when $r < h$. The lower bound for $m = 1$ movies and $r = h = 2$ is ≈ 0.1565 and for $r = h = 3$ the lower bound is ≈ 0.0524. The delay guaranteed by the above schedule outperforms the case $r = h = 2$ since the system has one additional channels but is substantially inferior to the schedule of the range [2..28] that guarantees a delay of ≈ 0.074 for the case $r = h = 3$ (see Table 2).

7.2 Different length movies

So far it was assumed that all the m movies have the same length normalized to 1. Clearly, by adding idle time at the end of the shorter movies, the schedules can be applied for different length movies as long as their lengths do not differ by much. Here, we show an example that demonstrate a gain in the guaranteed delay from sharing channels by movies of distinct length. In this example, the shifting technique is not used.

Consider a system with $h = 6$ channels and $m = 2$ movies: M_1 whose length is L_1 and M_2 whose length is L_2 where $L_2 = 11L_1/10$. Without channel sharing, a solution that dedicated 3 channels per movie guarantees a delay of $L_1/9$ for the first movie and delay of $L_2/9$ for M_2 by scheduling the range [1..9] of each movie on 3 channels.

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

With channel sharing, the schedule from Section 3.2

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

guarantees a delay of $L_2/10$ for M_2 and $L_2/10 = 11L_1/100 < L_1/9$ for M_1 by adding idle time to M_1 so its length is also L_2. The following schedule has a better performance. It partitions M_1 into 10 segments and partitions M_2 into 11 segments. It follows that all the 21 segments have the same size since $L_2 = 11L_1/10$. The first 5 channels have the same schedules as above. The schedule for channel C_6 is:

$$\mathbf{C_6} : (((7_1, 7_2), 11_2),\ 5_1,\ (10_1, (7_1, 7_2)),\ 5_2,\ ((7_2, 7_1), 10_2))$$

This is a valid schedule but not an RRR schedule (as some segments appear more than once).

Its cycle is of length 20:

$$[7_1\ 5_1\ 10_1\ 5_2\ 7_2\ 11_2\ 5_1\ 7_1\ 5_2\ 10_2\ 7_2\ 5_1\ 10_1\ 5_2\ 7_1\ 11_2\ 5_1\ 7_2\ 5_2\ 10_2]$$

This schedule guarantees a delay of $L_1/10 = L_2/11$ for both movies which is in improvement for both.

7.3 Movies with different popularities

All the schedules based on the shifting technique presented in this paper yield the same guaranteed delay for all the m movies because the scheduled range was the same for all of them. In fact, a different shifting can be applied to each movie as long as the segmentation number s is the same since the length of a slot $(1/s)$ must be fixed. That is, if the scheduled range for movie i is $[x_i..y_i]$ and $s = y_i + 1 - x_i$ for $1 \leq i \leq m$, then the guaranteed delay for movie i is x_i/s. These delays do not have to be the same and if some of the movies are more popular they may be granted with a smaller guaranteed delay.

More formally, associate with each movie a parameter p_i such that $\sum_{i=1}^{m} p_i = 1$. The parameter p_i can be interpreted as the popularity of movie i which is an independent probability that the next client wishes to view this movie. The goal is to minimize the weighted average guaranteed delay. That is, let D_i denote the granted guaranteed delay for movie i, then the goal is to minimize $\sum_{i=1}^{m} p_i D_i$.

$$\mathbf{C_1} : (3_1, (11_1, 9_1, 9_2), (6_1, 6_2))$$

$$\mathbf{C_2} : (4_1, 4_2, (8_1, 8_2), (12_1, 12_2, 13_2))$$

$$\mathbf{C_3} : (((7_1, 7_2), 11_2),\ 5_1,\ (10_1, (7_1, 7_2)),\ 5_2,\ ((7_2, 7_1), 10_2))$$

The above example for 3 channels and 2 movies demonstrates a possible gain from having different ranges when the movies have different popularities. This schedule is a valid schedule with 10 segments for both movies. The scheduled range for the first movie is [3..12] implying a guaranteed delay 3/10 and the scheduled range for the second movie is [4..13] implying a guaranteed delay 4/10. For $p_1 = p_2 = 1/2$, the weighted average guaranteed delay is $(1/2)(3/10 + 4/10) = 7/20 = 0.35$. This is already a very good delay for $s = 10$ and $\rho = 1.5$ for which the lower bound is ≈ 0.287. However, for $p_1 > p_2$ the advantage of this schedule is more apparent. For $p_1 = 3/4$ and $p_2 = 1/4$, the weighted average guaranteed delay is $(3/4)(3/10) + (1/4)(4/10) = 13/40 = 0.325$ and for $p_1 = 9/10$ and $p_2 = 1/10$ the weighted average guaranteed delay is $(9/10)(3/10) + (1/10)(4/10) = 31/100 = 0.31$.

8 Open Problems

This paper presented two simple techniques for broadcast schedules that can improve the system performance without any required enhancement. The performances of these techniques approach the known lower bounds for the guaranteed delay and yield schedules with almost optimal delay for practical systems. We conclude with some open problems.

- We are still looking for the best results for some practical values of h and m and s. We believe that there exist better algorithms that would outperform our algorithms for small values. For example, schedules do not have to be RRR schedules and segments may appear in more than one channel.
- For all the variants from Section 7: the receive-r model, different length movies, and different popularity movies, we do not have asymptotic upper bounds and lower bounds.
- We have examples showing that the shifting technique can be used to reduce the average delay. We omit them since the details are too long. We also developed additional technique for this model.

References

[1] A. Bar-Noy and R. E. Ladner. Windows Scheduling Problems for Broadcast Systems. *SIAM Journal on Computing (SICOMP),* 32(4):1091–1113, 2003.

[2] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly Optimal Perfectly-Periodic Schedules. *Distributed Computing,* 15(4):207–220, 2002.

[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, Editors),* Academic 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,* 4(3):112–121, 1996.

[5] L. Engebretsen and M. Sudan. Harmonic Broadcasting is Optimal. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* 431–432, 2002.

[6] W. Evans and D. G .Kirkpatrick. Optimally Scheduling Video-on-Demand to Minimize Delay shen Server and Receiver Bandwidth may Differ. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* 1041–1049, 2004.

[7] L. Gao, J. Kurose, and D. Towsley. Efficient Schemes for roadcasting Popular Videos. In *Multimedia Systems* 8(4): 284–294, 2002.

[8] R. L. Graham, D. E. Knuth, and O. Patashnik. Concrete Mathematics a Foundation for Computer Science. Addison-Wesley.

[9] A. Hu. Video-on-Demand Broadcasting Protocols: A Comprehensive Study. In *Proceedings of IEEE INFOCOM*, 508–517, 2001.

[10] K. A. Hua, Y. Cai, and S. Sheu Exploiting Client Bandwidth for More Efficient Video Broadcast. In *Proceedings of the 7th International Conference on Computer Communication and Networks (ICCCN),* 848–856, 1998.

[11] L. Juhn and L. Tseng. Harmonic Broadcasting for Video-on-Demand Service. *IEEE Transactions on Broadcasting,* 43(3):268–271, 1997.

[12] L. Juhn and L. Tseng. Fast Data Broadcasting and Receiving Scheme for Popular Video Service. *IEEE Transactions on Broadcasting,* 44(1):100–105, 1998.

[13] J. Pâris. A Simple Low-Bandwidth Broadcasting Protocol for Video-on-Demand. In *Proceedings of the 8th International Conference on Computer Communications and Networks (IC3N),* 118–123, 1999.

[14] J. Pâris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. In *Proceedings of the 10th International Conference on Computer Communications and Networks (IC3N),* 418–423, 2001.

[15] J. Pâris. A Simple but Efficient Broadcasting Protocol for Video-on-Demand. In *Proceedings of the 24th International Performance of Computers and Communication Conference (IPCCC 2005),* Phoenix, 167-174, 2005.

[16] J. Pâris, S. W. Carter, and D. D. E. Long. A Low Bandwidth Broadcasting Protocol for Video on Demand. In *Proceedings of the 7th International Conference on Computer Communications and Networks (IC3N),* 690–697, 1998.

[17] J. Pâris, S. W. Carter, and D. D. E. Long. A Hybrid Broadcasting Protocol for Video on Demand. In *Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN),* 317–326, 1999.

[18] J. Pâris and D. D. E. Long. Limiting the Receiving Bandwidth of Broadcasting Protocols for Video-on-Demand. In *Proceedings of the Euromedia Conference*, 107-111, 2000.

[19] Y. C. Tseng, M. H. Yang, and C. H. Chang. A Recursive Frequency-Splitting Scheme for Broadcasting Hot Video in VOD Service. *IEEE Transactions on Communications,* 50(8):1348–1355, 2002.

[20] S. Viswanathan and T. Imielinski. Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting. *ACM Multimedia Systems,* 4(3):197–208, 1996.

[21] S. Wolfram. Mathematica a System for Doing Mathematics by Computers. *Addison-Wesley Publishing Company.*