

A Modular Algorithm-Theoretic Framework for the Fair and Efficient Collaborative Prefetching of Continuous Media

Soohyun Oh Yo Huh Beshan Kulapala Goran Konjevod Andrea Richa Martin Reisslein

Abstract

Bursty continuous media streams with periodic playout deadlines (e.g., VBR-encoded video) are expected to account for a large portion of the traffic in the future Internet. By prefetching parts of ongoing streams into client buffers these bursty streams can be more efficiently accommodated in packet-switched networks. In this paper we develop a modular algorithm-theoretic framework for the fair and efficient transmission of continuous media over a bottleneck link. We divide the problem into the two subproblems of (i) assuring fairness, and (ii) efficiently utilizing the available link capacity. We develop and analyze algorithm modules for these two subproblems. Specifically, we devise a bin packing algorithm for subproblem (i), and a “layered prefetching” algorithm for subproblem (ii). Our simulation results indicate that the combination of these two algorithm modules compares favorably with existing monolithic solutions. This demonstrates the competitiveness of the decoupled modular algorithm framework, which provides a foundation for the development of refined algorithms for fair and efficient prefetching.

Keywords

Client buffer, continuous media, fairness, playback starvation, prefetching, prerecorded media, video streaming.

I. INTRODUCTION

Continuous media are expected to account for a large portion of the traffic in the Internet of the future and next generation wireless systems. These media have a number of characteristics that make their transport over networks very challenging, especially when the media are streamed in real-time. An alternative to real-time streaming is the download of the entire media file before the commencement of playback. This download significantly simplifies the network transport, but results in long response times to user requests, which are unattractive for many usage scenarios. We focus in this paper on the real-time streaming with minimal response times (start-up delays). Continuous media are typically characterized by periodic playout deadlines. For instance, a new video frame has to be delivered and displayed every 33 msec in NTSC video and every 40 msec in PAL video to ensure continuous playback. A frame that is not delivered in time is useless for the media playback and results in interruptions of the playback. For the network transport the continuous media are typically compressed (encoded) to reduce their bit rates. The efficient encoders, especially for video, produce typically highly variable frame sizes, with ratios of the largest frame size to the average frame size for a given video stream in the range between 8 and 15 and coefficients of variation

This work is supported in part by the National Science Foundation under Grant No. Career ANI-0133252 and Grant No. ANI-0136774 and the state of Arizona through the IT301 initiative.

Please direct correspondence to M. Reisslein.

The authors are with Arizona State University, Goldwater Center, MC 5706, Tempe, AZ 85287-5706, phone: (480)965-8593, fax: (480)965-8325, e-mail: {soohyun, yohuh, beshan, goran, aricha, reisslein}@asu.edu, web: <http://www.fulton.asu.edu/~mre>.

(defined as the ratio of standard deviation to mean) in the range from 0.8 to 1.3 [1, 2]. This highly variable traffic makes efficient real-time network transport very challenging since allocating network resources based on the largest frame size of a stream would result in low network utilization for most of the time. Allocating resources based on the average bit rates, on the other hand, could result in frequent playout deadline misses as the larger frames could not be delivered in time. An additional characteristic of a large portion of the continuous media delivered over networks is that it is prerecorded, e.g., stored video clips, such as news or music video clips, or full length videos, such as movies or lectures are streamed, as opposed to live media streams, e.g., the feed from a conference or sporting event.

An important characteristic of many of the user devices (clients) used for the media playback is that they have storage space. This storage space—in conjunction with the fact that a large portion of the media are prerecorded—can be exploited to prefetch parts of an ongoing media stream. This prefetching, which is often also referred to as work-ahead, can be used to smooth out some of the variabilities in the media stream and to relax the real-time constraints. The prefetching builds up prefetched reserves in the clients which help in ensuring uninterrupted playback. The prefetching (smoothing) schemes studied in the literature fall into two main categories: non-collaborative prefetching schemes and collaborative prefetching schemes.

Non-collaborative prefetching schemes, see for instance [3–16], smooth an *individual* stream by pre-computing (off-line) a transmission schedule that achieves a certain optimality criterion (e.g., minimize peak rate or rate variability subject to client buffer capacity). The streams are then transmitted according to the individually pre-computed transmission schedules. Collaborative prefetching schemes [17–20], on the other hand, determine the transmission schedule of a stream on-line as a function of *all* the other ongoing streams. For a single bottleneck link, this on-line collaboration has been demonstrated to be more efficient, i.e., achieves smaller playback starvation probabilities for a given streaming load, than the statistical multiplexing of streams that are optimally smoothed using a non-collaborative prefetching scheme [18]. We also note that there are transmission schemes which collaborate only at the commencement of a video stream, e.g., the schemes that align the streams such that the large intracoded frames of the MPEG encoded videos do not collude [21].

As discussed in more detail in the review of related work in Section I-A, most studies on collaborative prefetching in the literature consider the Join-the-Shortest-Queue (JSQ) scheme. The JSQ scheme is designed to achieve efficiency by always transmitting the next video frame for the client that has currently the smallest number of prefetched frames in its buffer. While *efficiency*, i.e., achieving a high utilization of the network resources and supporting a large number of simultaneous media streams with small playback starvation probabilities, is important for media streaming, so is the *fair* sharing of these resources among the supported streams. Without fairness, the supported streams may suffer significantly different playback starvation probabilities. Fairness in collaborative prefetching has received relatively little interest so far. The only study in this direction that we are aware of is the work by Antoniou and Stavrakakis [20], who introduced the deadline credit (DC) prefetch scheme. In the DC scheme the next frame is always transmitted to the client that has currently the smallest priority index, which counts the current number of prefetched frames in a client's

buffer minus the number of playback starvations suffered by the client in the past. By considering the “history” of playback starvations at the individual clients, the DC scheme can ensure fairness among the ongoing streams.

In this paper we re-examine the problem of fair and efficient collaborative prefetching of continuous media over a single bottleneck link. The single bottleneck link scenario is a fundamental problem in multimedia networking that arises in many settings, e.g., in the on-demand streaming of video over a cable plant [18,22] and the periodic broadcasting of video in a near video on demand system [19,23]. In addition, a solid understanding of the prefetching over a single bottleneck link is valuable when considering multihop prefetching. Also, the policy for prefetching over a wired bottleneck link is typically a module of the protocols for streaming video in a wireless networks, e.g., from a base station to wireless clients [24,25].

In this paper we develop and analyze a *modular* algorithmic framework for collaborative prefetching. In contrast to the DC scheme, where both fairness and efficiency are addressed by a single scheduling algorithm which considers a single priority index, we break the problem into the two subproblems of (i) ensuring fairness by avoiding continuous starvation of a client, and (ii) maximizing the bandwidth utilization. This decoupled, modular algorithm framework—which our complexity analysis and numerical results demonstrate to be competitive as it compares favorably with the existing monolithic approaches—has the advantage that different algorithms can be used for the two subproblems. Thus, our modular structure facilitates the future development of advanced collaborative prefetching schemes by allowing for the independent development and optimization of algorithm families for the two subproblems of achieving fairness and efficiency. Such future algorithm developments may, for instance, introduce different service classes and thus generalize the notion of fairness, where all clients receive the same grade of service. On the efficiency side, future algorithm developments may, for instance, take the relevance of the video data for the perceived video quality into consideration and strive to achieve high efficiency in terms of the perceived video quality.

This paper is organized as follows. In the following subsection we review the related work on collaborative prefetching. In Section II, we describe the problem set-up and introduce the notations used in the modeling of the collaborative prefetching. In Section III, we address the subproblem (i) of ensuring fairness. We develop and analyze a BIN-PACKING-ROUND algorithm which computes the minimum number of slots needed to schedule at least one frame for each stream with the minimum number of transmitted frames so far. Next, in Section IV, we develop and analyze the LAYERED-PREFETCHING-ROUND algorithm which maximizes the number of additional frames to be transmitted (prefetched) in the residual bandwidths of the minimum number of time slots found in Section III. In Section V, we conduct simulations to evaluate the developed modular solution to the problem of fair and efficient continuous media prefetching. Our simulation results indicate that the combination of our algorithm modules compares favorably with the JSQ and DC schemes. Our approach reduces the playout starvation probability approximately by a factor of two compared to the JSQ scheme. Also, the combination of our algorithm modules achieves about the same (and in some scenarios a slightly smaller) starvation probability and the same fairness as the

DC scheme which has been enhanced in this paper with some minor refinements. In Section VI, we outline an LP rounding approach to subproblem (ii). This approach accommodates different optimization goals, taking for instance the frame sizes into consideration when defining the frame transmission priority, through a profit function. In Section VII, we summarize our findings.

A. Related Work

In this section we give an overview of the existing literature on the collaborative prefetching of continuous media. The problem was first addressed in the patent filing by Adams and Williamson [22] and in the conference paper [17] by Reisslein and Ross. Both works independently proposed the Join-the-Shortest-Queue (JSQ) scheme for the problem. The JSQ scheme is a heuristic which is based on the earliest deadline first scheduling policy. The JSQ scheme proceeds in rounds, whereby the length of a given round is equal to the frame period of the videos. In each round, the JSQ scheduler continuously looks for the client which has currently the smallest reserve of prefetched frames and schedules one frame for this client. (Note that the scheduled frame is the frame with the earliest playout deadline among all the frames that are yet to be transmitted to all the clients.) If a client does not permit further transmissions in the round, because the next frame to be transmitted for the client does not fit into the remaining link capacity of the round or the client's prefetch buffer, then this client is removed from consideration. This scheduling process continues until all of the clients have been removed from consideration. This JSQ scheme has been evaluated through simulations with traces of bursty MPEG encoded video in [18]. It was demonstrated that collaborative prefetching employing the JSQ scheme gives smaller playback starvation probabilities for a given load of video streams than the statistical multiplexing of the individually optimally smoothed streams. Also, it was demonstrated that for a given tolerable playback starvation probability, the JSQ scheme supports more streams.

In the following years the JSQ scheduling principle for continuous media has been employed in video-on-demand (VOD) system designs, see for instance [26, 27]. Lin *et al.* [26] employ a least-laxity-first policy in their design. The laxity is defined as the deadline of a given chunk of video data minus the current time minus the time needed to transmit the chunk. Scheduling the chunk with the smallest laxity is thus roughly equivalent to the JSQ principle. Lin *et al.* design a comprehensive VOD system that on the protocol side incorporates the least-laxity-first policy and a variety of other mechanisms in their overall design.

There have also been efforts to adapt the JSQ scheme, which was originally designed for a centralized VoD system with one server to a more general architecture with multiple distributed servers sharing the same bottleneck link. The protocol design by Reisslein *et al.* [28] for the distributed prefetching problem employs quotas limiting the transmissions by the individual servers. The protocol design by Bakiras and Li [29] smoothes the videos over individual MPEG Groups of Pictures (GoPs) to achieve a constant bit rate for a small time duration. These constant bit rates for a given GoP are then exploited to conduct centralized scheduling according to the JSQ scheme.

The JSQ scheme has also been employed in periodic broadcasting schemes, which are employed in Near-Video-on-Demand (NVOD) systems. Saporilla *et al.* [23] partition a given video into seg-

ments using a fixed broadcast series (which specifies the relative lengths of the segments). Li and Nikolaidis [30] adaptively segment the video according to the bit rates of the various parts of a given VBR video. In both designs the transmissions of all the segments of all the offered videos share a common bottleneck link and the JSQ scheme is employed for the scheduling of the transmissions on the bottleneck link.

Fitzek and Reisslein [24] as well as Zhu and Cao [25] have employed the JSQ scheme as a component in their protocols designs for the streaming of continuous media over the shared downlink transmission capacity from a base station to wireless and possibly mobile clients. In these designs the JSQ scheme is combined with additional protocol components that account for the timevarying transmission conditions on the wireless links to the individual clients.

Recently, Antoniou and Stavrakakis [20] developed a deadline credit (DC) scheme which is designed to achieve efficient resource utilizations (similar to the JSQ scheme) and at the same time ensure that the resources are shared in a fair manner among the supported clients. As we describe in more detail, after having introduced our notation in Section II, the DC scheme differs from the JSQ scheme in that it uses a differently slotted time structure and transmits the next frame for the stream with the smallest number of on-time delivered frames.

More recently, Bakiras and Li [19] developed an admission control mechanism for their JSQ based prefetching scheme first presented in [29]. This admission control mechanism aggregates the individual client buffers into one virtual buffer and then employs effective bandwidth techniques to evaluate the probability for overflow of the virtual buffer, which corresponds to starvation of client buffers.

We note in passing that there have been extensive analyses of employing the join-the-shortest-queue policy in queueing systems consisting of multiple parallel queues, each being serviced by one or multiple servers, see for instance [31,32] and references therein. The problem considered in these studies differs fundamentally from the problem considered here in that there are multiple parallel servers in the queueing models, whereas we have only one server in our problem setting. In addition, there are multiple differences due to the periodic playout deadlines of variable size video frames in our problem setting and the Poisson arrivals of jobs with exponentially distributed service times considered in the queueing models.

II. SYSTEM SET-UP AND NOTATIONS

Figure 1 illustrates our system set-up for the streaming of prerecorded continuous media. The multimedia server contains a large number of continuous media streams in mass storage. To fix ideas we focus on video streams. Let J denote the number of video streams in progress. The video streams are encoded using some encoding scheme (such as MPEG, H.263, etc.). For our initial algorithm framework development and analysis we assume that the streams are of infinite length, i.e., have an infinite number of video frames. (In Section IV-C we discuss how to accommodate finite length streams in our algorithms.) Let $x_n(j)$ denote the size of the n th frame of video stream j . Note that for a constant-bit-rate (CBR) encoded video stream j the $x_n(j)$'s are identical, whereas for a variable-bit-rate (VBR) encoded video stream j the $x_n(j)$'s are variable. Because the video

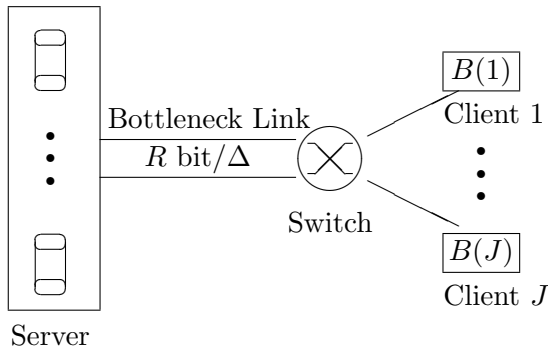


Fig. 1. J prerecorded video streams are multiplexed over a bottleneck link of capacity R bits/ Δ , and prefetched into client buffers of capacity $B(j)$ bits, $j = 1, \dots, J$.

streams are prerecorded the sequences of integers $(x_0(j), x_1(j), x_2(j), \dots)$ are fully known when the streaming commences. We denote $\bar{x}(j)$ for the average frame size of video j and let $x_{\max}(j)$ denote the largest frame of video stream j , i.e., $x_{\max}(j) = \max_n x_n(j)$. We denote $P(j)$ for the ratio of the largest (peak) to the average frame size of video j , i.e., $P(j) = x_{\max}(j)/\bar{x}(j)$, and let P denote the largest peak-to-mean ratio of the ongoing streams, i.e., $P = \max_j P(j)$. Throughout this study our focus is on VBR video, which allows for more efficient encoding compared to CBR video [33]. Let Δ denote the basic frame period of the videos in seconds. We assume that all videos have the same basic frame period Δ . (Our algorithms extend to videos where the frame periods are integer multiples of the basic frame period, as it typical for variable frame period video, in a straightforward fashion by inserting zeros for the sizes of the missing frames.)

We denote R for the transmission capacity of the bottleneck link in bits per basic frame period (of length Δ seconds) and assume throughout that the switch and the links connecting the switch to the clients are not a bottleneck. We also assume that the transmission capacity of the link is large enough to accommodate the largest video frame in one frame period, i.e., $\max_j x_{\max}(j) \leq R$, which is reasonable as in practical scenarios the link supports a moderate to large number of streams J , whereby each individual stream contributes a moderate fraction of the total load, even when this stream peaks in its bitrate. We denote $B(j)$, $j = 1, \dots, J$, for the capacity of the prefetch buffer (in bits) in client j , which we assume initially to be infinite (finite $B(j)$ are accommodated in Section IV-C).

For our model we initially assume that all J streams start at time zero; all with an empty prefetch buffer. (In Section IV-C we discuss how to accommodate a practical streaming scenario where ongoing streams terminate and new streams start up.) The video frame scheduling and transmission proceeds in slots (rounds) of length Δ . The transmission schedule for a given slot is computed before the slot commences and the video frames are transmitted according to the precomputed schedule during the slot. The video frames arriving at a client are placed in the client's prefetching buffer. For our model we assume that the first video frame is removed from the buffer, decoded, and displayed at the end of the first slot (denoted by $t = 0$). (In future work we

plan to extend our model to accommodate start-up latencies.) Each client displays the first frame of video stream (denoted by $n = 0$) during the second slot (denoted by $t = 1$), then removes the second frame from its prefetch buffer at the end of this second slot, decodes it, and displays it during the third slot, and so on. If at any one of these epochs there is no complete video frame in the prefetch buffer, the client suffers playback starvation and loses (a part or all of) the current frame. (The client may try to conceal the missing encoding information by employing error concealment techniques [34].) At the subsequent epoch the client will attempt to display the next frame of the video. Throughout, a video frame is not scheduled if it would arrive after its playout deadline, i.e., frame n of a stream is only scheduled up to (and including in) slot n . If frame n can not be scheduled before (or in) slot n , then it is dropped at the server (i.e., not transmitted) and the client will suffer a frame loss (play back starvation) in slot n .

More formally, we let $b_t(j)$, $j = 1, \dots, J$, denote the number of bits in the prefetch buffer of client j at the beginning of slot t , $t = 0, 1, 2, \dots$ (and note that $b_0(j) = 0$ for $j = 1, \dots, J$). Let $\beta_t(j)$, $j = 1, \dots, J$, denote the number of bits that are scheduled for transmission to client j during slot t . With these definitions

$$b_{t+1}(j) = [b_t(j) + \beta_t(j) - x_t(j)]^+, \quad (1)$$

where $[y]^+ = \max(y, 0)$. Note that the buffer constraint $b_t(j) + \beta_t(j) \leq B(j)$ must be satisfied for all clients j , $j = 1, \dots, J$, for all slots t , $t \geq 0$. Also, note that the link constraint $\sum_{j=1}^J \beta_t(j) \leq R$ must be satisfied for all slots t , $t \geq 0$

Let $p_t(j)$, $j = 1, \dots, J$, denote the length (run time) of the prefetched video segment (in terms of basic frame periods) in the prefetch buffer of client j at the beginning of slot t . (If all frame periods of a stream are equal to the basic frame period, then $p_t(j)$ gives the number of prefetched video frames.) Let $\psi_t(j)$, $j = 1, \dots, J$, denote the length of the video segment (in basic frame periods) that is scheduled for transmission to client j during slot t . Thus,

$$p_{t+1}(j) = [p_t(j) + \psi_t(j) - 1]^+. \quad (2)$$

Let $h_t(j)$, $j = 1, \dots, J$, denote the number of video frames that have been transmitted to client j up to (and including in) slot t (and note the initial condition $h_{-1}(j) = 0$ for $j = 1, \dots, J$). Let h_t^* denote the minimum number of frames that have been transmitted to any of the clients up to (and including in) slot t , i.e., $h_t^* = \min_j h_t(j)$

Let $\theta_t(j)$, $j = 1, \dots, J$, denote the lowest indexed frame for stream j that is still on the server and has not been dropped at the beginning of slot t . In other words, $\theta_t(j)$ is the frame with the earliest playout deadline that can still be transmitted to meet its deadline. (In Section III we discuss in detail how to maintain these variables.) Let θ_t^* denote the earliest deadline frame among the ongoing streams on the server at the beginning of slot t , i.e., $\theta_t^* = \min_j \theta_t(j)$.

Let $q_t(j)$, $j = 1, \dots, J$, denote the number of video frames of stream j that have missed their playout deadline up to (and including in) slot t . The counter $q_t(j)$ is incremented by one whenever client j wants to retrieve a video frame from its buffer, but does not find a complete frame in its

buffer. We define the *frame loss (starvation) probability* of client j as

$$P_{\text{loss}}(j) = \lim_{t \rightarrow \infty} \frac{q_t(j)}{t}.$$

We define the *average frame loss probability* $P_{\text{loss}} = \frac{1}{J} \sum_{j=1}^J P_{\text{loss}}(j)$.

A. Outlines of JSQ and DC Schemes

Before we proceed with the development of our modular algorithm-theoretic framework for collaborative prefetching, we briefly outline the existing schemes for collaborative prefetching—the JSQ scheme [17] and the DC scheme [20]—in terms of our notation. These outlines are intended to facilitate the comparisons with our analytical framework throughout this paper; for details on the JSQ and DC schemes we refer to the respective references.

The JSQ scheme proceeds in rounds, with the length of a round equal to the basic frame period of the video Δ (in seconds). For each round, the JSQ scheme precomputes the transmission schedule by considering to transmit a frame for the client with the smallest number of prefetched frames $p_t(j)$. If the frame will meet its playout deadline and fits into the remaining link capacity for the round and buffer capacity of the client, the considered frame is scheduled and the JSQ scheme looks again for the client with the smallest $p_t(j)$ (which may be the same or a different client). If the frame will not meet its deadline, it is dropped and the next frame of the client is considered. If the considered frame does not fit into the remaining link bandwidth or the buffer space, the client is removed from consideration for this round and the client with the next smallest $p_t(j)$ is considered. This process continues until all clients have been removed from consideration. The computational complexity of the JSQ scheme with the originally proposed linked list data structure [17, 18] is $O(J^2)$. We have developed a novel data structure based on a group of linked lists, whereby each list keeps the streams j with the same $p_t(j)$. This novel data structure, for which we refer the interested reader to [35] for details due to space constraints, reduces the complexity of the JSQ scheme to $O(J)$.

The DC scheme proceeds in slots, with the length of a slot significantly shorter than the frame period Δ of the video. When considering the DC scheme we express the frame period Δ in units of slots (not seconds, as done for JSQ). A slot length of 1/100th of a frame period, i.e., $\Delta = 100$ slots is considered in [20], but we found that shorter slot lengths give better results for the DC scheme and thus consider $\Delta = 1000$ slots and 2000 slots in our numerical work, see Section V. At the beginning of each slot the DC scheme looks for the stream with the smallest priority index, which is defined as the current number of prefetched frames $p_t(j)$ minus the number of dropped frames $q_t(j)$. We note that in our notation, $h_t(j) - p_t(j) + q_t(j) = t$. Hence the priority index of the DC scheme is $h_t(j) - t$. Since t is the same for all clients, the DC scheme essentially considers the client with the smallest number of on-time transmitted frame $h_t(j)$. The DC scheme transmits the considered frame if it will meet its playout deadline and fit into the remaining client buffer space. If the frame is transmitted, the DC algorithm completes the transmission of the frame and decides on the next frame to transmit at the beginning of the next slot. If a considered frame is not transmitted, the DC scheme considers the client with the next smallest priority index. The complexity of one execution of the DC algorithm is $O(J \log J)$, which is due to the sorting of the priority counters. The number

of times that the algorithm is executed in a frame period depends on the slot length and the frame size distribution. In the worst case the algorithm is executed Δ times in a frame period. Thus, the computational effort in a frame period is $O(\Delta J \log J)$.

III. AVOIDING STARVATION WITH BIN PACKING

The key objectives of our algorithm-theoretic framework for prefetching are to minimize starvation and to treat the clients fairly, i.e., the number of instances of playback starvation should be minimized and equally distributed among the J clients. In other words, the starvation probabilities of the clients should be roughly equal. (In ongoing work we are extending this notion of fairness to service differentiation with different classes of service, whereby the clients in each class experience about the same level of playback starvation.) The basic idea of our algorithm module for achieving fairness is to schedule exactly one frame per client for the clients which have so far received the minimum number of frames. More formally, we establish a correlation between the classical bin packing problem and the minimum number of slots needed to increase the minimum number of transmitted frames to a client by one. Let \mathcal{M} be the set of streams with the minimum number of transmitted frames, i.e., $\mathcal{M} = \{j | h_t(j) = h_t^*\}$.

In the classical bin packing problem, n objects with different sizes and m bins with a fixed capacity are given, and the problem is to find the minimum number of bins required to pack all objects into the bins. Any object must be placed as a whole into one of the bins. To pack an object into a bin, the residual bin capacity must be larger than or equal to the size of the object. In our video frame scheduling problem, we can think of the bottleneck link in each slot as a bin, the transmission capacity of the bottleneck link in a slot (i.e., R) is the bin capacity, and the video frames to be transmitted to the clients in \mathcal{M} are the objects.

A. Specification of BIN-PACKING-ROUND Algorithm

The BIN-PACKING-ROUND algorithm proceeds in loops. Each iteration of the loop completes a *bin-packing round*. More specifically, suppose a bin-packing round starts with the beginning of slot t_s and recall that $h_{t_s-1}^*$ denotes the minimum number of video frames transmitted to any of the clients up to (and including in) slot $t_s - 1$. During the bin-packing round one video frame is scheduled for each of the clients in \mathcal{M} , i.e., the bin-packing round ends when the number of frames scheduled for each of the clients in \mathcal{M} has been incremented by one. This may take one or more slots, and we refer to the slots in a given bin-packing round as *scheduling steps*. Note that in the “avoiding starvation” subproblem addressed in this section we do not prefetch additional frames, i.e., once each client in \mathcal{M} has been scheduled a frame we move on the next bin-packing round, even though there may be residual capacity in the bins (slots on bottleneck link) making up the bin-packing round. In Section IV we efficiently fill this residual capacity with additional frames.

The schedule for a given bin-packing round starting at the beginning of slot t_s is computed with the BIN-PACKING-ROUND(t_s) algorithm, which is summarized in Figure 2. At the end of the BIN-PACKING-ROUND, we have pre-computed the schedule of frames to be transmitted for each client for each of the scheduling steps in this round.

BIN-PACKING-ROUND(t_s)

1. Initialization

1.1 Given $h_{t_s-1}(j)$ and $q_{t_s-1}(j)$, for all clients $j = 1, \dots, J$

1.2 Let t_s be the first slot and t_e the last slot of this bin-packing round, i.e., $t_e = t_s$.

1.3 For all $j = 1, \dots, J$: $h_{t_e}(j) \leftarrow h_{t_s-1}(j)$, $q_{t_e}(j) \leftarrow q_{t_s-1}(j)$, and $\beta_{t_e}(j) = 0$

1.4 Let $\theta(j)$ be the lowest frame number of stream j on the server for all streams $j = 1, \dots, J$

2. $\mathcal{M} = \{i | h_{t_e}(i) = h_{t_s-1}^*\}$ 3. For each stream $i \in \mathcal{M}$

3.1 Search for the first scheduling step $t' \leq \theta(i)$ ($t' = t_s, \dots, t_e + 1$) such that $\sum_{j=1}^J \beta_{t'}(j) + x_{\theta(i)}(i) \leq R$. If $t' > t_e$ then $t_e = t_e + 1$.

3.2 If such a time slot t' exists, then schedule frame $\theta(i)$ for client i in slot t'

$\beta_{t'}(i) \leftarrow \beta_{t'}(i) + x_{\theta(i)}(i)$

$h_{t''}(i) \leftarrow h_{t''}(i) + 1$ for all $t'' = t', \dots, t_e$

$\theta_{t''}(i) \leftarrow \theta_{t''}(i) + 1$ for all $t'' = \theta(i), \dots, t_e$

3.3 If no such a time slot t' is found, then drop frame $\theta(i)$ of client i

$q_{t''}(i) \leftarrow q_{t''}(i) + 1$ for all $t'' = \theta(i), \dots, t_e$

$\theta_{t''}(i) \leftarrow \theta_{t''}(i) + 1$ for all $t'' = \theta(i), \dots, t_e$

Goto step 3.1

Fig. 2. BIN-PACKING-ROUND algorithm

The basic operation of the algorithm is as follows. The values in step 1.1 are inherited from the end of the previous bin-packing round and $t_s = t_e$ denotes the first time slot considered in the new bin-packing round. The algorithm schedules one frame per client in \mathcal{M} as long as the size of the frame fits into the residual bandwidth $R - \sum_{j=1}^J \beta_{t'}(j)$, where t' is the corresponding time slot found in step 3.1, and the frame playout deadline is met. If necessary the frame is scheduled in a new slot $t_e + 1$. If no such time slot meeting the frame's playout deadline exists, then the frame is dropped and the next frame in the same stream is considered (step 3.3).

After a bin-packing round has been precomputed, the actual transmission of the video frames is launched. Note that for each client j in \mathcal{M} (from the beginning of the bin-packing round) the number of received frames is increased by one at the end of the actual transmission of the frames scheduled for the bin-packing round. That is for each client j in \mathcal{M} the number of transmitted frames is increased from $h_{t_s-1}^*$, at the start of the bin-packing round to $h_{t_e} = h_{t_s-1}^* + 1$ at the end of one bin-packing round. If a given bin-packing round is longer than one time slot (or one scheduling step), then every client not scheduled in a slot t inside the bin-packing round experiences a frame loss in slot t . Note that this does not hold when future frames are prefetched, see Section IV.

B. Analysis of BIN-PACKING-ROUND Algorithm

Recall that we initially assume that all streams start at the beginning of slot 0 with an empty prefetch buffer, i.e., $h_{-1}(j) = 0$ for all clients j . The number of scheduling steps comprising the first bin-packing round is equal to the minimum number of slots needed to transmit exactly one frame for each client. The first bin-packing round ends with $h_t(j) = 1$ for all clients j . The second bin-packing round ends with $h_t(j) = 2$ for all clients j , and so on. Hence, at any slot t , $h_t(j) = y$ or $y + 1$, for all clients $j = 1, \dots, J$ and for some integer y .

During one bin-packing round consisting of r scheduling steps, each client will experience exactly $r - 1$ frame losses (provided no future frames are prefetched, see Section IV) and the number of frame losses during this round is the same for all clients, which is summarized in the following lemma.

Lemma 1: Suppose that all streams start at the beginning of slot 0, then each client has the same number of frame losses in one bin-packing round. Moreover, if we minimize the number of scheduling steps in a bin-packing round, then we can also minimize the number of frame losses by a client in this round.

The classical bin packing problem is well known to be *NP*-hard. Hence, according to Lemma 1, it can be also shown that achieving fairness while attempting to minimize frame losses is *NP*-hard. The following lemma shows that the BIN-PACKING-ROUND is a 1.7 approximation factor algorithm using the analogy between our algorithm and a well-known algorithm for the classical bin packing problem. Let F be the set of frames that will end up being transmitted in this bin-packing round.

Lemma 2: The minimum number of slots to increase h^* by one when using the BIN-PACKING-ROUND algorithm is asymptotically no more than $1.7 \cdot \gamma$, where γ is the minimum number of slots to increase h^* by one when an optimal algorithm is used on the frames in F .

Proof: We are essentially running the FIRST FIT (FF) algorithm that solves the classical bin packing problem on the frames in F . The analogy between our algorithm and the FF algorithm is as follows: The frames in F are our set of objects considered for the bin packing problem and the order in which we consider them in the bin packing problem is exactly the order in which they are considered by the BIN-PACKING-ROUND algorithm, ignoring all the frames dropped in-between. Hence, the number of slots in one bin-packing round calculated using the BIN-PACKING-ROUND algorithm is the same as the number of bins calculated using the FF algorithm for the classical bin packing problem. The approximation ratio on the minimum number of bins for the FF algorithm has been proven to be 1.7 asymptotically [36]. ■

For the classical bin packing problem, we can achieve better performance by running the FF algorithm after sorting frames by non-increasing order of sizes, which gives us an approximation factor of roughly 1.2 [36]. This algorithm is called the First Fit decreasing algorithm. However, the FF decreasing algorithm is not applicable to our problem. The reason is that we cannot guarantee that the frames will always be considered in non-increasing order since frames may be dropped, being replaced by larger frames within a given bin-packing round. As a conclusion, we introduce the following theorem, which follows immediately from Lemma 1.

Theorem 1: We obtain a 1.7-approximation on the maximum number of frame losses per client using the BIN-PACKING-ROUND algorithm, if we consider only the set of frames transmitted in this round.

Before we close this section, we consider the complexity of the BIN-PACKING-ROUND algorithm.

Theorem 2: The BIN-PACKING-ROUND algorithm computes a bin-packing round in $O(J^3)$, where J is the number of clients.

Proof: The worst case scenario is as follows. The number of streams in \mathcal{M} is J . For the i th iteration of the for loop (Step 3), the corresponding stream (say, stream j) has to drop the first $i - 1$ frames, i.e., frames $\theta(j)$, $\theta(j) + 1, \dots, \theta(j) + i - 2$ at Step 3.3, and then schedules frame $\theta(j) + i - 1$ into the next empty slot (i.e., it increases the number of scheduling steps in this bin packing round). Hence the number of comparisons needed to schedule a frame in the i th iteration is $i(i + 1)/2$. Since $i \leq J$ and there are at most J streams in \mathcal{M} , the overall time complexity is $O(J^3)$. ■

For essentially all streaming scenarios of practical interest we can assume that the sum of the average frame sizes of all simultaneously supported streams is less than or equal to the link capacity, i.e., $\sum_{j=1}^J \bar{x}(j) \leq R$. This condition is also commonly referred to as stability condition and means that the long run streaming utilization of the link bandwidth in terms of the ratio of the sum of the long run average bit rates of the supported streams to the link capacity is less than or equal to 100%. Recalling from Section II that we know the largest peak-to-mean frame size ratio P for the prerecorded videos, which is typically in the range from 8 to 15 [1], we can significantly tighten the worst case computational complexity as shown in the following corollary.

Corollary 1: Given the largest peak-to-mean ratio of the frame sizes P and the stability condition $\sum_{j=1}^J \bar{x}(j) \leq R$, the BIN-PACKING-ROUND algorithm computes a bin packing round in $O(JP^2)$.

Proof: Note that

$$\sum_{j=1}^J x_{\max}(j) \leq \sum_{j=1}^J P(j) \cdot \bar{x}(j) \quad (3)$$

$$\leq P \cdot \sum_{j=1}^J \bar{x}(j) \quad (4)$$

$$\leq P \cdot R, \quad (5)$$

where (3) follows from the definition of $P(j)$, (4) follows from the definition of P , and (5) follows from the stability condition. The FF heuristic always uses a number of bins which is at most twice the sum of the sizes of the objects to be packed, divided by the capacity of a bin [36]. Hence, for any set of frames involved in a bin-packing round, the number of time slots in the bin-packing round will be at most

$$2 \left\lceil \frac{\sum_{j=1}^J x_{\max}(j)}{R} \right\rceil \leq 2(P+1) \text{ bins.} \quad (6)$$

Re-tracing the steps in the proof of Theorem 2, we note that the number of comparisons needed to schedule a frame for any given stream in a given bin packing round is bounded by $[2(P+1)] \cdot [2(P+1) + 1]/2$. Hence the overall time complexity of the BIN-PACKING-ROUND algorithm is $O(JP^2)$. ■

IV. MAXIMIZING BANDWIDTH UTILIZATION WITH LAYERED PREFETCHING

While the bin-packing round algorithm module of the preceding section focuses on fairness among the clients, we now turn to the subproblem of maximizing the bandwidth utilization (efficiency) by prefetching future video frames. In this section and in Section VI we introduce two algorithm modules to maximize the bandwidth utilization after the schedule for the bin-packing round has been computed. In this section we define a *layered prefetching round* as a series of computations that schedules video frames after a bin-packing round is calculated in order to better utilize the bandwidth. The basic idea of our prefetching strategy is as follows: After the bin-packing round schedule has been computed, the prefetching round computation starts as long as there is sufficient residual bandwidth in the scheduling steps of the corresponding bin-packing round. It is natural

to schedule a frame with an earlier playout deadline before scheduling a frame with later playout deadline. Therefore, a frame is considered for scheduling only if no frame with earlier playout deadline can fit into the residual bandwidth.

A. Specification of LAYERED-PREFETCHING-ROUND Algorithm

Let $\mathcal{S} = \{S_1, \dots, S_r\}$ be the set of r scheduling steps within a given bin-packing round, each with residual bandwidth α_i , $i = 1, \dots, r$. (If there is no residual bandwidth for scheduling step i , then $\alpha_i = 0$.) Suppose that this bin-packing round starts at slot t_s , and ends at slot $t_s + r - 1$ (i.e., scheduling step S_i corresponds to time slot $t_s + i - 1$). We group the unscheduled frames in the server queues according to their playout deadlines. Recall that $\theta_{t_s+r-1}^*$ denotes the earliest playout deadline of the unscheduled (and not yet dropped) frames on the server at the end of the bin-packing round ending at the end of slot $t_s + r - 1$. We define $G_{\theta' - \theta_{t_s+r-1}^*}$ to be the group of unscheduled frames whose playout deadline is θ' ($\geq \theta_{t_s+r-1}^*$). Hence the frames in G_0 have the earliest playout deadline.

The goal of a prefetching round is to maximize the number of frames scheduled for each group G_i while enforcing the earliest deadline first policy. We first consider the frames in G_0 and schedule as many frames as possible into the residual bandwidths. When no more frames in G_0 fit into any of the residual bandwidths, we consider the frames in G_1 , and schedule them until no more frames fit into the residual bandwidths. This process is repeated for each G_i , ($i = 2, 3, \dots$) until no frames fit into the residual bandwidths or until there are no frames left to be considered. At the end of this process, the scheduled frames form the *layers* in each scheduling step; the frames from G_0 form layer 0, the frames from G_1 form layer 1, and so on.

We consider the scheduling for each group G_ℓ as a variant of the multiple knapsack problem: The set of knapsacks is defined by the set S_1, \dots, S_y , where $y = \min\{t_s + r - 1, \ell + \theta^*\}$; the capacity of knapsack i is defined by $\alpha_i - \sum_{n=0}^{\ell-1} g_{i,n}$, where $g_{i,n}$ is the sum of the sizes of the frames in G_n , $n = 0, \dots, \ell - 1$, assigned to slot i ; the objects to be packed are defined by all frames from the group G_ℓ , where the profit of placing any frame in a knapsack is always equal to 1. Our objective here is to assign objects (frames in G_ℓ) to the knapsacks (slots that meet the deadline $\theta^* + \ell$ of the frames in G_ℓ on bottleneck link) in order to optimize the total profit. We assume here that every video frame has the same importance, i.e., the profit of packing (scheduling) is the same for every frame. Thus, our goal is to maximize the number of objects packed into the knapsacks. (The LP rounding module introduced in Section VI is more general and assigns different video frames different priorities.)

The LAYER algorithm provided in Figure 3: (i) sorts the frame in G_ℓ according to their sizes in non-decreasing order, and (ii) schedules each ordered frame in the first scheduling step that accommodates the frame size and meets its playout deadline. We use again t_s to denote the first time slot (scheduling step) of the bin-packing round, and t_e to denote the last time slot of this bin-packing round. To optimize the bandwidth utilization, our LAYERED-PREFETCHING-ROUND algorithm allows for a frame with a later playout deadline to be transmitted prior to one with an earlier deadline. Recall that $h_t(j)$, $j = 1, \dots, J$, denotes the number of frames that have been

LAYER(ℓ, θ^*, t_s, t_e)

1. $y = \min\{t_e, \ell + \theta^*\}$
2. Given $h_t(j)$, $q_t(j)$, and $\hat{h}_t(j)$ for time $t = t_s, \dots, y$, for all clients $j = 1, \dots, J$
3. $M' = \{i | i \in G_\ell\}$
4. Sort M' in non-decreasing order of frame sizes
5. For all $i \in M'$
 - 5.1 Search for the first slot t' ($t' = t_s, \dots, y$) such that $\sum_{j=1}^J \beta_{t'}(j) + x_\ell(i) \leq R$
 - 5.2 When such a scheduling round t' is found schedule frame ℓ for client i at time t'
 - 5.2.1 $\beta_{t'}(i) \leftarrow \beta_{t'}(i) + x_\ell(i)$
 - 5.2.2 $h_{t''}(i) \leftarrow h_{t''}(i) + 1$, for all $t'' = t', \dots, y$
 - 5.2.3 **if** (ℓ is equal to the frame index that $\theta(i)$ points to)
 - UPDATE($i, \ell, t', y, 1$)
 - 5.2.4 **else**
 - $\hat{h}_{t'}(i) \leftarrow \hat{h}_{t'}(i) + 1$
 - 5.3 When no such a scheduling round t' is found,
 - 5.3.1 **If** $\ell = y$, then drop frame ℓ of video stream i
 - $q_{t''}(i) \leftarrow q_{t''}(i) + 1$, for all $t'' = t', \dots, y$
 - UPDATE($i, \ell, t', y, 0$)
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) - 1$, for all $t = t_s, \dots, t_e$

Fig. 3. LAYER algorithm for layer ℓ

UPDATE (stream-index, frame-index, t_{start}, t_{end} , frame-status)

Initialization:

- $i = \text{stream-index}$
- $\ell = \text{frame-index}$
- $d = \theta(i) - \ell$

if (frame-status = 1) // a frame is scheduled

- if** ($d > 0$)
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) - d + 1$, for all $t = t_{start}, \dots, t_{end}$
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) + d$, for all $t = t_{start}, \dots, t_{end}$
- else if** ($d < 0$)
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) + 1$, for all $t = t_{start}, \dots, t_{end}$

else if (frame-status = 0) // a frame is dropped

- if** ($d > 0$)
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) - d + 1$, for all $t = t_{start}, \dots, t_{end}$
 - $\hat{h}_t(i) \leftarrow \hat{h}_t(i) + d - 1$, for all $t = t_{start}, \dots, t_{end}$

Fig. 4. UPDATE algorithm

scheduled for client j so far. We define $\hat{h}_t(j)$, $j = 1, \dots, J$, to be the number of frames that are currently scheduled out of order for client j . In other words, $\hat{h}_t(j)$ is increased by one whenever a frame for client j is scheduled but some of its preceding frames (frames with earlier playout deadlines) are still unscheduled in the server queue. Let $\dot{h}_t(j) = h_t(j) - \hat{h}_t(j)$, $j = 1, \dots, J$, i.e., $\dot{h}_t(j)$ is the number of frames that have been scheduled for client j in order (without gaps). Note that with the scheduling of future frames with gaps, the equations (1) and (2) need to be modified to account for frame deadlines.

The reason we have $\hat{h}_t(j)$ and $\dot{h}_t(j)$ in addition to $h_t(j)$ is as follows: Suppose that a stream i has a large frame f followed by some small frames. Due to the restricted capacity of the residual bandwidths, frame f may not be scheduled, while the small frames with later playout deadlines are scheduled during the layered prefetching rounds. If we consider only the total number of transmitted frames $h_t(j)$ for each client, then stream i may not have a chance to schedule any of its frames including frame f during the next bin-packing round due to the frames prefetched in the previous rounds. As a result, frame f may miss its deadline and a frame loss occurs. Such a frame loss is unnecessary and unfair to stream i . Hence, $\dot{h}_t(j)$ is used instead of $h_t(j)$ in determining the streams with the smallest number of successfully transmitted frames. We modify the BIN-PACKING-ROUND algorithm accordingly. The modified version of the algorithm is presented in Figure 6.

A prefetching round is computed by calling the LAYER(ℓ) algorithm repeatedly for each layer ℓ , as specified in Figure 5. In practice we limit the number of times the LAYER(ℓ) algorithm is called without significantly affecting the overall performance of the prefetching round, see Section V. We denote W for the lookup window size that determines the number of times the LAYER(ℓ) algorithm is called.

LAYERED-PREFETCHING-ROUND(W, t_s, t_e)

Let θ^* denote the earliest playout deadline of the unscheduled (and not yet dropped) frames on the server at the end of slot $t_s - 1$.

For each group G_ℓ , $\ell = 0, 1, \dots, W - 1$, if there is any available bandwidth, call LAYER(ℓ, θ^*, t_s, t_e).

Fig. 5. PREFETCHING-ROUND algorithm with lookup window size W

BIN-PACKING-ROUND2(t_s)

1. Initialization

1.1 * Given $h_{t_s-1}(j)$, $q_{t_s-1}(j)$, $\hat{h}_{t_s}(j)$, and $\dot{h}_{t_s}(j)$, for all clients $j = 1, \dots, J$

1.2 Let t_s be the starting time and t_e the last slot of this bin-packing round, i.e., $t_e = t_s$.

1.3 For all $j = 1, \dots, J$

$h_{t_e}(j) \leftarrow h_{t_e-1}(j)$

$q_{t_e}(j) \leftarrow q_{t_e-1}(j)$

* $\hat{h}_{t_e}(j) \leftarrow \hat{h}_{t_e-1}(j)$

* $\dot{h}_{t_e}(j) \leftarrow \dot{h}_{t_e-1}(j)$

$\beta_{t_e}(j) = 0$

1.4 Let $\theta(j)$ be the lowest frame number of stream j on the server for all streams $j = 1, \dots, J$

2. * $\mathcal{M} = \{i | h_{t_e}(i) = \min_{1 \leq j \leq J} \hat{h}_{t-1}(j)\}$

3. For each stream $i \in \mathcal{M}$

3.1 Search for the first scheduling step $t' \leq \theta(i)$ ($t' = t_0, \dots, t_e + 1$) such that $\sum_{j=1}^J \beta_{t'}(j) + x_{\theta(i)}(i) \leq R$. If $t' > t_e$ then $t_e = t_e + 1$.

3.2 If such a time slot t' exists, then schedule frame $\theta(i)$ for client i

$\beta_{t'}(i) \leftarrow \beta_{t'}(i) + x_{\theta(i)}(i)$

$h_{t''}(i) \leftarrow h_{t''}(i) + 1$, for all $t'' = t', \dots, t_e$

$\ell = \theta(i)$

Update $\theta(i)$ so that $\theta(i)$ points to the next unscheduled and not dropped frame

* UPDATE($i, \ell, t', t_e, 1$)

3.3 If no such a time slot t' is found, then

$q_{t''}(i) \leftarrow q_{t''}(i) + 1$ for all $t'' = \theta(i), \dots, t_e$

$\ell = \theta(i)$

Update $\theta(i)$ so that $\theta(i)$ points to the next unscheduled and not dropped frame

* UPDATE($i, \ell, t', t_e, 0$)

Go to step 3.1

4. * Return the value of t_e

Fig. 6. BIN-PACKING-ROUND2 algorithm: The lines starting with symbol * are added or modified from the original BIN-PACKING-ROUND

B. Analysis of LAYERED-PREFETCHING-ROUND Algorithm

Dawande et al. [37] give 1/2-approximation algorithms for the multiknapsack problem with assignment restrictions. The following lemma shows that our LAYER(ℓ) algorithm is a 1/2-approximation factor algorithm.

Lemma 3: The algorithm LAYER(ℓ) is a 1/2-approximation algorithm on the maximum number of scheduled frames from layer ℓ , given the residual bandwidths after LAYER(0), LAYER(1), ..., LAYER($\ell - 1$) have been executed in this order.

Proof: Let $\bar{\alpha}_i$ be the residual bandwidth of step S_i at the start of LAYER(ℓ) for $i = 1, \dots, r$. Let N be the number of scheduled frames by the LAYER(ℓ) algorithm and let N^* be the maximum possible number of scheduled frames in G_ℓ , given the residual bandwidths $\bar{\alpha}_1, \dots, \bar{\alpha}_r$ of steps S_1, \dots, S_r . The size of any unscheduled frame at the end of LAYER(ℓ) is larger than the maximum residual bandwidth of the respective scheduling steps after LAYER(ℓ) is executed. Thus we can schedule at most $(\sum_i \alpha'_i / \max_i \alpha'_i)$ more frames, where α'_i is the residual bandwidth of each scheduling step S_i at the end of the LAYER(ℓ) algorithm. Then the maximum number of scheduled

frames

$$N^* \leq N + \frac{\sum_i \alpha'_i}{\max_i \alpha'_i} \leq N + r \cdot \frac{\max_i \alpha'_i}{\max_i \alpha'_i} = N + r,$$

where r is the number of scheduling steps in the bin-packing round. Hence

$$\frac{N}{N^*} \geq \frac{N}{N+r}.$$

If $N \geq r$, it is obvious that the algorithm is a $1/2$ -approximation. Suppose $N < r$. We claim that the optimal solution cannot schedule more than $2N$ frames. Since $N < r$, there exist at least $r - N$ scheduling steps that do not have any frames from layer ℓ after the LAYER(ℓ) algorithm ends. Let S' denote the set of scheduling steps that contain at least one frame from layer ℓ , and let $S'' = S - S'$. We observe that the scheduling steps in S'' do not have enough residual bandwidth to transmit any of the frames left at the server queue at the end of LAYER(ℓ). The scheduling steps in S'' may have enough residual bandwidth to accommodate some or all of frames which have been scheduled in S' during the LAYER(ℓ) algorithm. Suppose we move all these frames to the scheduling steps in S'' , and try to schedule more frames into scheduling steps in S' . Since the sizes of the new frames are larger than the frames originally scheduled in S' , we can schedule at most N new frames into the scheduling steps in S' . Therefore, the claim holds. Hence, The algorithm LAYER(ℓ) is a $1/2$ -approximation algorithm on the maximum number of scheduled frames. ■

Lemma 4: The time complexity of the LAYER(ℓ) algorithm is $O(J^2)$.

Proof: Since the number of elements in each group G_ℓ is at most J , sorting the frames in G_ℓ in non-decreasing order of their sizes takes $O(J \log J)$. The number of searches to schedule each frame in G_ℓ is at most equal to the number of slots in the bin-packing round, which is no larger than J ; since there are at most J frames in each G_ℓ , the total amount time spent in scheduling the frames in G_ℓ is $O(J^2)$. Updating $\hat{h}_t(i)$ and $\hat{h}_t(i)$ takes only constant time by keeping a pointer to the first frame in the server for each stream i . Therefore, the overall time complexity of the algorithm is $O(J^2)$. ■

The following theorem summarizes the above results:

Theorem 3: For given residual bandwidths, the LAYER(ℓ) algorithm gives an $1/2$ -approximation factor on maximizing the number of scheduled frames from group G_ℓ and its run time is $O(J^2)$. The overall time complexity of the layered prefetching round is $O(WJ^2)$, where W is the lookup window size for prefetching frames.

For practical streaming scenarios that satisfy the stability condition and for which P is known, the LAYER(ℓ) algorithm is significantly less complex. Noting that in these practical scenarios the number of slots in a bin packing round is bounded by $2(P+1)$, as shown in the proof of Corollary 1, we obtain the following theorem:

Theorem 4: Given the largest peak-to-mean ratio of the frame sizes P and the stability condition $\sum_{j=1}^J \bar{x}(j) \leq R$, the time complexity of the LAYER(ℓ) algorithm is $O(J(P + \log J))$. The overall running time of a layered prefetching round is $O(PJ(\log J + W))$.

Proof: Since the maximum number of steps considered in a layered prefetching round is $2(P+1)$, the time for scheduling the frames in G_ℓ is $O(JP)$. Since we may also need to sort G_ℓ , in

the worst-case, the complexity of the LAYER(ℓ) algorithm is $O(J(P + \log J))$.

Most of the W groups G_ℓ considered in a layered prefetching round have already been considered (and therefore have already been sorted) in previous rounds: There are only at most $2(P+1)$ “new” groups G_ℓ which need to be fully sorted in the current round (since we only see at most $2(P+1)$ new time slots in this round, our lookup window shifts by at most this much, exposing at most $2(P+1)$ new groups which need to be sorted from scratch in this round). It takes $O(PJ \log J)$ time to sort the respective groups. We still need to schedule frames for W groups G_ℓ in a layered prefetching round. Thus the overall complexity of a layered prefetching round is $O(PJ(\log J + W))$. ■

C. Accommodating Finite Stream Durations and Finite Buffers

For a newly joining stream $J+1$ at time t we initialize $h_t(J+1)$, $\dot{h}_t(J+1)$, and $\hat{h}_t(J+1)$ as follows:

1. Find $\dot{h}_t^* = \min_{1 \leq j \leq J} \dot{h}_t(j)$.
2. If the minimum gap between the current time t and the earliest deadline among the unscheduled frames θ_t^* is larger than zero, i.e., $t - \theta_t^* > 0$ then
 - If the same existing stream attains both \dot{h}_t^* and $t - \theta_t^*$, then we set $h_t(J+1) = \dot{h}_t(J+1) = \dot{h}_t^* - t + \theta_t^*$.
 - Else we set the new video stream’s $h_t(J+1)$ and $\dot{h}_t(J+1)$ to the $\dot{h}_t(j)$ of the client j attaining θ_t^* minus the minimum of the gap.
3. Else we set $h_t(J+1) = \dot{h}_t(J+1) = \dot{h}_t^*$
4. Set $\hat{h}_t(J+1) = 0$.

This approach is based on the following reasoning. Since the new video stream has the end of the current slot as playback deadline of the first frame, it has high urgency for transmission. If the on-going streams have some frames in their buffer, they may not need to transmit a frame urgently in the current slot. So until the deadline of a frame of the new video stream is equivalent to the earliest deadline among all unscheduled frames from the on-going streams, the new video stream should be given priority over the other video streams. However, if there is at least one already on-going stream that has the end of the current slot as a deadline for an unscheduled frame, the new video stream has at least the minimum of \dot{h} .

To accommodate finite client buffer capacities the server keeps track of the prefetch buffer contents through Eqn. (1) (modified for the layered prefetching) and skips a frame that is supposed to be scheduled but would not fit into the client buffer.

V. NUMERICAL RESULTS

A. Evaluation Set-up

In this section we evaluate the algorithms developed and analyzed in the preceding sections through simulations using traces of MPEG-4 encoded video. The employed frame size traces give the frame size in bit in each video frame [1]. We use the traces of QCIF video encoded without rate control and fixed quantization scales of 30 for each frame type (I, P, and B). These traces correspond to video with a low, but roughly constant visual quality. The choice of these traces of low-quality video is motivated by the fact that the traffic burstiness of this low-quality video lies

between the less bursty high-quality video and the somewhat more bursty medium-quality video. The average bit rate of the low-quality encoded videos is in the range from 52 kbps to 94 kbps. To achieve constant average utilizations in our simulations we scaled the frame sizes of the individual to an average bit rate of 64 kbps. The peak to mean ratios, standard deviations, and coefficients of variation (standard deviation normalized by mean frame size) of the frame sizes of the scaled traces are given in Table I.

TABLE I
VIDEO TRAFFIC STATISTICS: PEAK TO MEAN RATIO AND STANDARD
DEVIATION OF FRAME SIZE. AVERAGE BIT RATE IS 64 KBPS FOR ALL
STREAMS

Name of Video	P/M Ratio	Std. Dev.	Coeff. of Var.
<i>Citizen Kane</i>	12.6	2756	1.08
<i>Die Hard I</i>	9.2	2159	0.84
<i>Jurassic Park I</i>	10.5	2492	0.97
<i>Silence of the Lambs</i>	13.4	2234	0.87
<i>Star Wars IV</i>	15.2	2354	0.92
<i>Star Wars V</i>	10.1	2408	0.94
<i>The Firm</i>	10.5	2581	1.01
<i>Terminator I</i>	10.6	2156	0.84
<i>Total Recall</i>	7.7	2260	0.88
<i>Aladdin</i>	8.7	2343	0.92
<i>Cinderella</i>	14.8	2276	0.89
<i>Baseball</i>	7.8	2166	0.85
<i>Snowboard</i>	8.7	2099	0.82
<i>Oprah</i>	8.4	2454	0.96
<i>Tonight Show</i>	15.9	3513	1.32
<i>Lecture-Gupta</i>	11.9	3199	1.25
<i>Lecture-Reisslein</i>	13.8	3058	1.19
<i>Parking Lot</i>	9.2	2577	1.01

TABLE II
FRAME LOSS PROBABILITY
WITH BIN PACKING AS A
FUNCTION OF WINDOW SIZE
 W ; $R_s = 32$, $J = 30$, $B = 64$
KBYTES

W	P_{loss}
16	0.0031
32	0.0029
64	0.0025
128	0.0020
150	0.0018
256	0.0014
384	0.0016

All used traces correspond to videos with a frame rate of 25 frames per second, i.e., the frame period is $\Delta = 40$ msec for all videos. For ease of discussion of the numerical results we normalize the capacity of the bottleneck link by the 64 kbit/sec average bit rate of the streams and denote this normalized capacity by R_s . Note that $R_s = R/(\Delta \cdot 64 \text{ kbps})$, where R is in units of bit/ Δ and Δ is the frame period in seconds.

We conduct two types of simulations, *start-up* simulations and *steady state* simulations. In the start-up simulations all streams start with an empty prefetch buffer at time zero, similar to the scenario initially considered in our algorithm development. Whereas the streams had an infinite number of video frames in our initial model, we fix the number of video frames (stream duration) for the simulations at $T = 15,000$ frames, i.e., 10 minutes. We run many independent trials of this start-up simulation. For each independent trial we randomly pick a video trace for each ongoing stream and a random starting phase into each selected trace. For each trial the loss probability for each individual stream is recorded. These independent loss probability observations are then

used to find the mean loss probability for each client and the 90% confidence interval of the loss probability.

With the steady state simulations all streams start again at time zero with an empty prefetch buffer and a random trace and random starting phase are selected. In addition, each stream has a random lifetime drawn from an exponential distribution with a mean of T frames. When a stream terminates (i.e., the last frame of the stream has been displayed at the client), the corresponding client immediately starts a new stream (with an empty prefetch buffer) at the beginning of the next slot. For the new stream we draw again a random trace, starting phase, and lifetime. With the steady state simulation there are always J streams in progress. We estimate the loss probabilities (and their 90% confidence intervals) of the individual clients after a warm-up period of 60000 frames (i.e., 40 minutes) by using the method of batch means.

All simulations are run until the 90% confidence intervals are less than 10% of the corresponding sample means.

B. Comparison of JSQ and modular BP Approach

In Table II we first examine the effect of the window size W on the performance of the combination of the BIN-PACKING-ROUND and LAYERED-PREFETCHING-ROUND algorithm modules, which we refer to as “bin packing” in the discussion and as “BP” for brevity in the plots. We observe that the loss probability decreases as the window size increases. That is, as the layered prefetching algorithm considers more frames for the scheduling, there is a better chance to fill even a very small remaining transmission capacity. With the resulting higher utilization of the transmission capacity and increased prefetched reserves (provided the buffer capacity B is sufficiently large), the probability of playback starvation is reduced. However, we also observe that the loss probability slightly increases as the window size increases from $W = 256$ to $W = 384$. In brief, the reason for this is that with a very large window size the bin packing algorithm tends to transmit frames that have deadlines far into the future. These prefetched future frames take up buffer space in the client and tend to prevent the prefetching of (large) frames with closer playout deadlines. Thus making the client slightly more vulnerable to playback starvation. Overall the results indicate that a larger window size generally reduces the loss probability; however, extremely large window sizes may degrade the performance slightly

Next, we evaluate the bin packing approach using the steady state simulations and compare its performance with the JSQ approach. Fig. 7 gives the average frame loss probability P_{loss} as a function of the prefetch buffer capacity B . We set the transmission capacity to $R_s = 16$ and consider $J = 14$ and 15 simultaneous streams. The window sizes for the bin packing algorithm are set to $W = 64$ for $B = 32$ Kbytes, $W = 128$ for $B = 64$ Kbytes, $W = 192$ for $B = 96$ Kbytes, and $W = 256$ for $B = 128$ Kbytes. We observe from Fig. 7 that the frame loss probabilities with bin packing are roughly half of the corresponding loss probabilities for JSQ. For $J = 15$ and a buffer of $B = 128$ kbytes, the JSQ scheme gives a frame loss probability of $4.8 \cdot 10^{-3}$ while bin packing gives a loss probability of $2.5 \cdot 10^{-3}$. For the smaller load of $J = 14$, the gap widens to loss probabilities of $2.5 \cdot 10^{-4}$ for JSQ and $4.2 \cdot 10^{-5}$ for bin packing. The explanation for this gap in performance is

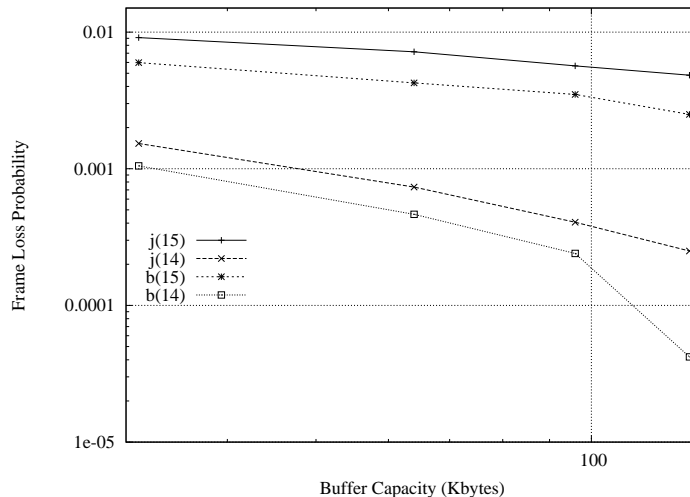


Fig. 7. Average frame loss probability as a function of buffer capacity B for $J = 14$ and $J = 15$ streams for bin packing and JSQ for a link capacity of $R_s = 16$.

as follows. The JSQ scheme stops scheduling frames for a frame period when the first unscheduled frame from none of the streams fits into the remaining bandwidth. The bin packing scheme, on the other hand, continues scheduling frames in this situation, by skipping the frame(s) that are too large to fit and looks for smaller future frames to fit into the remaining bandwidth.

Note that so far we have considered only the average (aggregate) performance of the prefetch algorithm. We now examine the fairness aspects. To test the fair allocation of transmission resources we consider *heterogeneous* streaming scenarios, where the ongoing streams differ in their average bandwidth, traffic variability, stream lifetime, and client buffer capacity. First, we consider heterogeneous average bandwidths. We set the link capacity to $R_s = 32$ streams with an average bit rate of 64 kbps. We stream either 30 streams with an average bit rate of 64 kbps, a mix of 14 streams with an average bit rate of 64 kbps and 8 streams with an average bitrate of 128 kbps, or 15 streams with an average bit rate of 128 kbps. Note that the average system load is 30/32 in all three scenarios. We observe from Fig. 8 that with JSQ the higher average bit rate streams experience larger frame loss probabilities than the lower average bit rate streams. Considering the fairness criterion of distributing the frame losses equally among the clients the higher bandwidth clients are treated unfairly with JSQ. With BP, on the other hand, the frame losses are fairly distributed.

Next, we examine the effect of mixing streams with different variabilities. For this experiment we consider constant bit rate (CBR) streams with a bit rate of 64 kbps and higher variability streams (generated by increasing the variability of the traces in Table I while maintaining the 64 kbps average bit rate.) In Fig. 9 we plot the individual loss probabilities for a mix of CBR and higher variability streams. We observe that with JSQ the clients with the higher variability streams experience smaller frame loss probabilities than the clients with CBR streams. The explanation for this result is as follows. The higher variability streams have a small portion of very large video frames, but also have a very large portion of small frames. As a result, higher variability streams can transmit more (small) frames over any remaining bandwidth. As a consequence the higher variability streams have

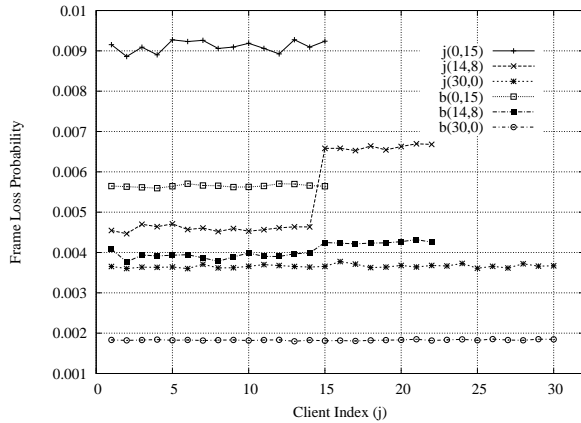


Fig. 8. Frame loss probabilities for individual clients with (a, b) mix of a 64 kbps streams and b 128 kbps streams; $R_s = 32$, $B = 64$ Kbytes, and $W = 150$, fixed

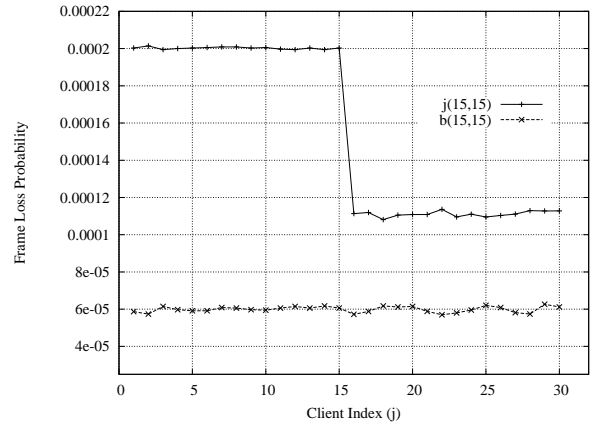


Fig. 9. Frame loss probabilities of individual clients for (a, b) mix of a CBR and b higher variability streams; $R_s = 32$, $J = 30$, $B = 64$ KByte, fixed

typically a larger number of frames prefetched and thus experience fewer instances of play back starvation. Comparing the frame loss probabilities for JSQ and bin packing, we observe that bin packing gives again smaller and roughly equal loss probabilities for the individual clients.

In additional experiments, which we can not include here due to space constraints, we have found that the average stream life time and client buffer size have a relatively small impact on the fairness, see [35].

C. Comparison of DC scheme with modular BP approach

In this section we compare the DC scheme, which we briefly outlined in Section II-A, with our modular bin packing approach. We use the start-up simulation set-up for this comparison as the DC scheme is formulated for this scenario in [20]. We present results for our experiments with $\Delta = 1000$ and 2000 slots per frame period. (We found that these shorter slot lengths give better results; a slot length of 1/100th of the frame period is considered in [20].) In the DC scheme the client buffer capacity is expressed in terms of a maximum deadline credit counter in units of number of video frames (which are of variable size for VBR-encoded video resulting in varying capacity in terms of the deadline credit counter). For the comparison with our scheme where the buffer capacity is a fixed number of bytes, we considered two adaptations of the DC scheme. In the “DC avg.” adaptation we convert the buffer capacity in bytes to a maximum deadline credit counter (in number of video frames) using the average bit rate of the video. In the “DC ref.” adaptation we convert the buffer capacity in bytes to the maximum deadline credit counter using the actual sizes of the frames in the buffer and considered for transmission.

In Table III we compare the DC avg., DC ref., JSQ, and BP approaches in terms of the frame loss probability for a system with a link capacity of $R_s = 16$. We observe from the table that considering the 90% confidence intervals of 10% around the reported sample means, bin packing gives smaller loss probabilities than the DC scheme for $J = 15$. For $J = 14$, the DC scheme with the refined adaptation gives approximately the same performance as the other schemes.

TABLE III

FRAME LOSS PROBABILITY COMPARISON BETWEEN DC, JSQ, AND BP APPROACHES ($R_s = 16$, $B = 64$ KBYTE)

	P_{loss}
DC avg., $\Delta = 1000$ slots, $J = 15$	0.006053
DC ref., $\Delta = 1000$ slots, $J = 15$	0.001703
DC ref., $\Delta = 2000$ slots, $J = 15$	0.001454
JSQ, $J = 15$	0.001219
BP, $J = 15$	0.000960
DC avg., $\Delta = 1000$ slots, $J = 14$	0.001783
DC ref., $\Delta = 1000$ slots, $J = 14$	0.000127
DC ref., $\Delta = 2000$ slots, $J = 14$	0.000110
JSQ, $J = 14$	0.000105
BP, $J = 14$	0.000101

TABLE V

FRAME LOSS PROBABILITY P_{loss} AND COMPUTING TIME T_c COMPARISON BETWEEN DC AND BP APPROACHES ($R_s=64$, $B = 64$ KBYTE)

J	DC ref., $\Delta = 2000$ sl.		BP	
	P_{loss}	T_c	P_{loss}	T_c
61	0.00131	0.0028	0.00019	0.0024
62	0.00318	0.0030	0.00065	0.0025
63	0.00728	0.0038	0.00166	0.0041

TABLE IV

FRAME LOSS PROBABILITY P_{loss} AND COMPUTING TIME T_c COMPARISON BETWEEN DC AND BP APPROACHES ($R_s=32$, $B = 64$ KBYTE)

J	DC ref., $\Delta = 2000$ slots		BP	
	P_{loss}	T_c	P_{loss}	T_c
30	0.00070	0.0008	0.00028	0.0007
31	0.004361	0.0009	0.00185	0.0008

TABLE VI

FRAME LOSS PROBABILITY P_{loss} AND COMPUTING TIME T_c COMPARISON BETWEEN DC AND BP APPROACHES ($R_s=128$, $B = 64$ KBYTE, $J = 122$ AND 123 GIVE WITH BP EXCEEDINGLY SMALL P_{loss} VALUES WHICH ARE OMITTED)

J	DC ref., $\Delta = 2000$ sl.		BP	
	P_{loss}	T_c	P_{loss}	T_c
122	0.00167	0.0118		
123	0.00264	0.0121		
124	0.00485	0.0122	0.000093	0.0094
125	0.00871	0.0129	0.00027	0.0133
126	0.01388	0.0135	0.00073	0.0135
127	0.02004	0.0148	0.00134	0.0141

To gain further insight into the relative performance comparison of the DC and BP approaches we compare in Tables IV through VI the DC ref. scheme with $\Delta = 2000$ slots with the BP approach in terms of the frame loss probability P_{loss} and the computation time T_c . The computation time T_c measures the time needed to compute the scheduling decisions for a frame period on a contemporary PC with Pentium IV processor running at 3.2 GHz. We observe from the tables that the computing times for the DC and BP schemes are roughly the same; there is a slight tendency for the BP approach to be faster, but the differences are rather small. We note that all measured computation times are well below the duration of a frame period, which is 40 msec with PAL video and 33 msec with NTSC video.

Turning to the results for the frame loss probability P_{loss} in Tables IV–VI we observe that the difference in P_{loss} widens as the link capacity R_s increases and a correspondingly larger numbers of streams J are transmitted. Whereas for the $R_s = 32$ scenario the BP gives roughly half the P_{loss} of the DC scheme, the gap widens to over one order of magnitude for the scenario with $R_s = 128$. This

TABLE VII
 MAXIMUM FRAME LOSS PROBABILITY AMONG $J = 10$ STREAMS, $R_s = 10.5$

	$P_{\text{loss}}^{\text{max}}$	$P_{\text{loss}}^{\text{max}} - d$	$P_{\text{loss}}^{\text{max}} + d$
LP	0.006943	0.000694	0.013191
BP	0.017750	0.006805	0.028695

appears to indicate that the BP scheme is better able to exploit the increased statistical multiplexing effect that comes with an increased number of streams.

When interpreting the results in Tables IV–VI from the perspective of the number of supported streams subject to a fixed maximum permissible frame loss probability, we observe that the BP approach gives $P_{\text{loss}} < 0.002$ for $J = S - 1$ streams in all considered scenarios. (When the number of streams is increased to the stability limit, i.e., $J = S$ and beyond the loss probability generally increases significantly.) The DC approach, on the other hand, supports fewer streams with the $P_{\text{loss}} < 0.002$ criterion; for the $S = 128$ scenario up to $J = 122$ streams.

To gain insight into the frame loss patterns we have examined the runs of consecutively lost frames and the runs of consecutive frames without any losses. We found that the frame losses are not bursty; rather the runs of lost frames consist typically of only one frame. For the scenario with $S = 64$ and $J = 62$, for instance, and with the BP approach the lengths of the runs of consecutively lost frames have a mean of 1.016 frames and a standard deviation of 0.019 frames, while the lengths of the runs of frames without any loss have a mean of 1480.1 frames and a standard deviation of 28.41 frames. With the DC approach, on the other hand, the lengths of the runs of lost frames were one frame in all simulations, and the lengths of the runs of frames without any loss had a mean of 331.3 frames and a standard deviation of 2123.6 frames. In more extensive simulations (see [35]) we have also observed that the DC scheme is approximately as fair as the modular bin packing approach.

D. Comparison of modular BP approach with LP solution

To further assess the performance of the modular bin packing approach we compare it with the following linear relaxation of the prefetching problem. Let N_i denote the total number of considered frames in stream i . Recall from Section II that the end of slot t is the deadline of frame t of stream i . We use the variable $y_{l,j}^i$ to denote the fraction of frame j of stream i that is transmitted during slot l . The first constraint-set (8) says that L_{max} is at least the (fractional) number of dropped frames for every stream i . The second constraint-set (9) says that no more bits can be scheduled during any time slot than the bandwidth allows. The last constraint-set (10) says that each frame can only be counted once towards the number of scheduled frames.

$$\min L_{\text{max}} \quad \text{subject to} \quad (7)$$

$$N_i - \sum_j \sum_{l \leq t} y_{l,j}^i \leq L_{\text{max}} \quad \forall i \quad (8)$$

$$\sum_i \sum_{l \leq t} x_t(i) y_{l,j}^i < R \quad \forall j \quad (9)$$

$$\sum_l y_{l,j}^i < 1 \quad \forall(i, j) \quad (10)$$

An optimal (minimum L_{\max}) solution to this LP is a lower bound on the maximum frame loss probability of a client. Solving the LP becomes computationally prohibitive even for moderate numbers of considered frames N_i . We were able to run 20 iterations of a start-up simulation with stream durations of $N_i = 400$ frames. In Table VII we report the maximum (fractional) frame loss probability (with 90% confidence interval) corresponding to the LP solution L_{\max} and the corresponding maximum frame loss probability obtained with the modular bin packing approach for the same 20 experiments. Although the confidence intervals are quite loose, due to the enormous computational effort, the results do indicate that the solutions are generally of the same order of magnitude. One has to keep in mind here that the LP does not (and can not) enforce the delivery of complete video frames whereas the bin packing approach delivers only complete video frames as they are required for successful decoding.

VI. MAXIMIZING UTILIZATION WITH LP ROUNDING

In this section, we consider again the scenario presented in Section II, where all streams start at time zero with empty prefetch buffers. We outline a more general algorithm module to solve the subproblem of maximizing the bandwidth utilization. This more general module is more flexible than the layered prefetching module developed in Section IV. Whereas in the layered prefetching module every prefetched frame increases the profit by one, the LP rounding module developed in this section accommodates more general profit functions. With this more general profit function module we can accommodate different optimization objectives, such as minimize the long run fraction of encoding information (bits) that misses its playout deadline (note that the layered approach was limited to minimizing the frame loss probability). Also, we can assign the frames different priorities, e.g., higher priority for large Intracoded (I) frames.

Our more general solution approach is based on solving a linear relaxation of the original problem and then rounding the fractional solution to obtain an integer solution for the original problem. We reduce our problem to a maximization version of the generalized assignment problem (Max GAP). The Max GAP is defined as follows: There is a set of m items and a set of r knapsacks. Each item has to be assigned to exactly one of the knapsacks; each knapsack i has a capacity $C(i)$, and there is a profit $p(i, j)$ and a size $s(i, j)$ associated with each knapsack $i = 1, \dots, r$ and each item $j = 1, \dots, m$. The optimization criterion is to maximize the total profit. This problem has been shown to be APX-hard [38], even for the case when all frames have equal profit [39]. In our problem, the set of knapsacks is defined by the set \mathcal{S} , the capacity of knapsack i is defined by α_i , $i = 1, \dots, r$, and the objects to be packed are defined by the frames to be transmitted to the clients. The profit of placing a frame in a knapsack can be defined as follows. Let c be the maximum residual bandwidth after a bin-packing round, i.e., $c = \max_{1 \leq i \leq r} \alpha_i$. Then the profit of frame n of stream j can be defined as

$$p_j(n, t) = \frac{1}{(c + 1)^{n-t}} \quad (11)$$

if frame n is scheduled into a time slot $t \leq n$, otherwise, $p_j(n, t) = 0$. Note that the frames in

the same group G_i have the same profit. Our objective here is to assign objects (frames) into the knapsacks (scheduling steps) in order to maximize the total profit. As we will see later, for the given profit function, the objective of the prefetching round is to maximize the number of frames scheduled for each group G_i , where the groups G_i are considered in increasing order of i .

Claim 1: For the profit function defined above, a frame is scheduled only after no frame with earlier deadline can be scheduled into the residual bandwidth.

Proof: To prove this claim, we show that scheduling one frame for group G_i always produces higher profit than scheduling c frames for $G_{i'}$, $i' > i$, where c is the maximum residual bandwidth. Since

$$\frac{1}{(c+1)^i} > c \cdot \frac{1}{(c+1)^{i+1}}, \quad \forall i,$$

for $c > 0$, the claim holds. ■

The above claim implies that for the profit function we defined in (11), the Max GAP is equivalent to the multiknapsack problem presented in Section IV. The approximation bounds obtained by both approaches are the same, as we will see shortly.

The Max GAP approach has the advantage that it allows for different profit functions. These profit functions translate into different optimization criteria, which we explore in Section VI-A in greater detail. On the downside, the solution techniques for the Max GAP are more involved than the ones presented in Section IV.

The algorithm we use for computing the prefetching round is based on solving the linear relaxation and rounding technique given in [40]. Due to space constraints we give here only a brief sketch of our proposed algorithm module. We first convert the problem into a minimization problem, by changing profits into suitable costs. We then solve a linear relaxation of a parametric version of the minimization problem (where the sizes of the bins are scaled up by a factor of 2). We build a bipartite graph with edge costs based on the solution to this linear relaxation, and find a minimum cost matching on this graph. We schedule the frames according to this matching. A key property to be used during the scheduling is that if we exceed the link capacity at any scheduling step, the results in [40] guarantee that there exists a frame to be removed which contributes to less than half of the profit in that step. A full description of this algorithm appears in [35], which gives also the proofs of the following lemmas on the approximation guarantees of the algorithm:

Lemma 5: The prefetching algorithm outlined in this section is a $1/2$ -approximation algorithm on the total profit.

Lemma 6: If we optimize the total profit, then we also maximize the number of frames scheduled for each group G_i , given the frames that have already been scheduled for G_1, \dots, G_{i-1} .

A. Profit Functions

The LP rounding approach allows us to achieve different objectives by simply changing the profit functions for each frame. According to the profit function (11), the profits of the frames with the same playout deadlines are equal. Hence, in order to optimize the total profit according to (11), it is necessary to maximize the number of frames to be scheduled in each playout deadline group. If

instead we want to optimize a different objective, such as the number of bits to be scheduled for each group, the profit function can be changed accordingly. In the case of maximizing the number of bits delivered in time, a new profit function for frame n (which is to be removed from the prefetch buffer at the end of slot t , see Section II) can be defined by

$$p_j(n, t) = \frac{x_n(j)}{(c+1)^{n-t}} \quad (12)$$

if frame n is scheduled into time slot $t \leq n$ and $p_j(n, t) = 0$ if $t > n$. We observe that the profit function (12) has the following properties.

- Among the frames in the same group, the larger the frame size, the higher the profit.
- If frames n_1 , n_2 , and n_3 are in the same group and $x_{n_1} = x_{n_2} + x_{n_3}$, then $p(n_1) = p(n_2) + p(n_3)$.
- The profit of a frame in group G_i is always larger than that of a frame in group G_{i+1} .

For many video encoding schemes, e.g., MPEG with predictive encoding, the large intracoded frames are more important than the smaller predictive encoded frames. In that case, it may be beneficial to schedule a large frame n_2 for group G_{i+1} instead of a small frame n_1 for group G_i . Let the profit function then be

$$p_j(n, t) = \frac{x_n(j)}{(\gamma c)^{n-t}} \quad (13)$$

if frame n is scheduled into time slot $t \leq n$ where $0 < \gamma \leq 1$; $p_j(n, t) = 0$ if $t > n$. Consider an example where $\gamma = 1/4$ and $c = 20$. Suppose that there are two frames n and $n+1$. According to the profit function (13), the profit of frame $n+1$ is equal to or larger than that of frame n if the size of frame $n+1$ is at least five times larger than the size of frame n .

VII. CONCLUSION

We have developed a modular algorithm-theoretic framework for the prefetching of continuous media over a bottleneck link. We have divided the problem into the two separate subproblems of (i) ensuring fairness, and (ii) efficient bandwidth utilization. We have developed algorithm modules for both subproblems. We have investigated the theoretical performance bounds and complexities of the individual algorithm modules and compared the playout starvation probabilities achieved by the combination of the modules with the JSQ and DC prefetching schemes. Our simulation results indicate that the combined modules compare favorably with the existing JSQ and DC schemes, thus demonstrating the competitiveness of our modular approach.

There are several interesting and important avenues for future work. One avenue is to develop new algorithm modules for the first—the ensuring fairness—component in our algorithm framework. In particular, those new algorithm modules could be designed to generalize the notion of fairness from providing all clients with the same service quality to providing different classes of service quality.

Another avenue for future work is a detailed study of the profit function space outlined in Section VI-A. The goal of such a study could be to prioritize the prefetching of the video frames according to their contribution to the perceived decoded video quality to maximize the efficiency of the streaming not only in terms of the number of simultaneously supported streams, but also in terms of the visual quality provided by these streams.

We believe that our modular algorithm framework provides a solid foundation for these future explorations.

REFERENCES

- [1] Frank H.P. Fitzek and Martin Reisslein, "MPEG-4 and H.263 video traces for network performance evaluation," *IEEE Network*, vol. 15, no. 6, pp. 40–54, November/December 2001.
- [2] Patrick Seeling, Martin Reisslein, and Beshan Kulapala, "Network performance evaluation using frame size and quality traces of single-layer and two-layer video: A tutorial," *IEEE Communications Surveys and Tutorials*, vol. 6, no. 3, pp. 58–78, Third Quarter 2004.
- [3] Chris Bewick, Rubem Pereira, and Madjid Merabti, "Network constrained smoothing: Enhanced multiplexing of MPEG-4 video," in *Proceedings of IEEE International Symposium on Computers and Communications*, Taormina, Italy, July 2002, pp. 114–119.
- [4] Han-Chieh Chao, C. L. Hung, and T. G. Tsuei, "ECVBA traffic-smoothing scheme for VBR media streams," *International Journal of Network Management*, vol. 12, pp. 179–185, 2002.
- [5] Wu-Chi Feng and Jennifer Rexford, "Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video," *IEEE Transactions on Multimedia*, vol. 1, no. 3, pp. 302–313, Sept. 1999.
- [6] Matthias Grossglauser, Srini Keshav, and David Tse, "RCBR: A simple and efficient service for multiple time-scale traffic," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 741–755, Dec. 1997.
- [7] Zonghua Gu and Kang G. Shin, "Algorithms for effective variable bit rate traffic smoothing," in *Proceedings of IEEE International Performance, Computing, and Communications Conference*, Phoenix, AZ, Apr. 2003, pp. 387–394.
- [8] Marwan Krunz, Wei Zhao, and Ibrahim Matta, "Scheduling and bandwidth allocation for distribution of archived video in VoD systems," *Journal of Telecommunication Systems, Special Issue on Multimedia*, vol. 9, no. 3/4, pp. 335–355, Sept. 1998.
- [9] Marwan Krunz, "Bandwidth allocation strategies for transporting variable-bit-rate video traffic," *IEEE Communications Magazine*, vol. 37, no. 1, pp. 40–46, Jan. 1999.
- [10] Simon S. Lam, Simon Chow, and David K. Y. Yau, "A lossless smoothing algorithm for compressed video," *IEEE/ACM Transactions on Networking*, vol. 4, no. 5, pp. 697–708, Oct. 1996.
- [11] Rajesh Sabat and Carey Williamson, "Cluster-based smoothing for MPEG-based video-on-demand systems," in *Proceedings of IEEE International Conference on Performance, Computing, and Communications*, Phoenix, AZ, Apr. 2001, pp. 339–346.
- [12] James Salehi, Zhi-Li Zhang, James Kurose, and Don Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 397–410, Aug. 1998.
- [13] Arun Solleti and Kenneth J. Christensen, "Efficient transmission of stored video for improved management of network bandwidth," *International Journal of Network Management*, vol. 10, pp. 277–288, 2000.
- [14] Bobby Vandalore, Wu-Chi Feng, Raj Jain, and Sonia Fahmy, "A survey of application layer techniques for adaptive streaming of multimedia," *Real-Time Imaging Journal*, vol. 7, no. 3, pp. 221–235, 2001.
- [15] Dejian Ye, Zixiang Xiong, Huai-Rong Shao, Qiufeng Wu, and Wenwu Zhu, "Wavelet-based smoothing and multiplexing of VBR video traffic," in *Proceedings of IEEE Globecom*, San Antonio, TX, Nov. 2001, pp. 2060–2064.
- [16] Zhi-Li Zhang, James Kurose, James Salehi, and Don Towsley, "Smoothing, statistical multiplexing and call admission control for stored video," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1148–1166, Aug. 1997.
- [17] Martin Reisslein and Keith W. Ross, "A Join-the-Shortest-Queue prefetching protocol for VBR video on demand," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Atlanta, GA, Oct. 1997, pp. 63–72.
- [18] Martin Reisslein and Keith W. Ross, "High-performance prefetching protocols for VBR prerecorded video," *IEEE Network*, vol. 12, no. 6, pp. 46–55, Nov/Dec 1998.
- [19] Spiridon Bakiras and Victor O.K. Li, "Maximizing the number of users in an interactive video-on-demand system," *IEEE Transactions on Broadcasting*, vol. 48, no. 4, pp. 281–292, Dec. 2002.
- [20] Zoe Antoniou and Ioannis Stavrakakis, "An efficient deadline-credit-based transport scheme for prerecorded semisoft continuous media applications," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 630–643, Oct. 2002.
- [21] Marwan Krunz and Satish K. Tripathy, "Exploiting the temporal structure of MPEG video for the reduction of bandwidth requirements," in *Proceedings of IEEE Infocom*, Kobe, Japan, Apr. 1997, pp. 67–74.
- [22] M. B. Adams and L. D. Williamson, "Optimum bandwidth utilization in a shared cable system data channel," United States Patent Number 6,124,878, filed December 1996, granted September 2000.

- [23] Despina Saparilla, Keith W. Ross, and Martin Reisslein, “Periodic broadcasting with VBR-encoded video,” in *Proc. of IEEE Infocom*, New York, NY, Mar. 1999, pp. 464–471.
- [24] Frank H.P. Fitzek and Martin Reisslein, “A prefetching protocol for continuous media streaming in wireless environments,” *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 10, pp. 2015–2028, Oct. 2001.
- [25] Hao Zhu and Guohong Cao, “A power-aware and QoS-aware service model on wireless networks,” in *Proceedings of IEEE Infocom*, Hong Kong, Hong Kong, Mar. 2004.
- [26] Chow-Sing Lin, Min-You Wu, and Wei Shu, “Transmitting variable-bit-rate videos on clustered VOD systems,” in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, New York, NY, July 2000.
- [27] Yiu-Wing Leung and Tony K. C. Chan, “Design of an interactive video-on-demand system,” *IEEE Transactions on Multimedia*, vol. 5, no. 1, pp. 130–140, Mar. 2003.
- [28] Martin Reisslein, Keith W. Ross, and Vincent Verillotte, “A decentralized prefetching protocol for VBR video on demand,” in *Multimedia Applications, Services and Techniques — ECMAST '98 (Lecture Notes in Computer Science, Vol. 1425)*, D. Hutchison and R. Schäfer, Eds., Berlin, Germany, May 1998, pp. 388–401, Springer Verlag.
- [29] Spiridon Bakiras and Victor O. K. Li, “Smoothing and prefetching video from distributed servers,” in *Proceedings of IEEE International Conference on Networking Protocols (ICNP)*, Toronto, Canada, Oct. 1999, pp. 311–318.
- [30] Fulu Li and Ioanis Nikolaidis, “Trace-adaptive fragmentation for periodic broadcast of VBR video,” in *Proceedings of 9th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Basking Ridge, NJ, June 1999, pp. 253–264.
- [31] Hwa-Chun Lin and C. S. Raghavendra, “An approximate analysis of the join the shortest queue (JSQ) policy,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 301–307, Mar. 1996.
- [32] Weiping Zhu, “Analysis of JSQ policy on soft real-time scheduling in cluster,” in *Proceedings of IEEE Conference on High Performance Computing—Asia*, Beijing, China, 2000, pp. 277–281.
- [33] T. V. Lakshman, Antonio Ortega, and Amy R. Reibman, “VBR video: Tradeoffs and potentials,” *Proceedings of the IEEE*, vol. 86, no. 5, pp. 952–973, May 1998.
- [34] Y. Wang and Q. Zhu, “Error control and concealment for video communication: A review,” *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.
- [35] Soohyun Oh, Yo Huh, Goran Konjevod, Andrea Richa, and Martin Reisslein, “A modular algorithm-theoretic framework for the fair and efficient collaborative prefetching of continuous media (extended version),” Tech. Rep., Dept. of Electrical Eng., Arizona State University, Dec. 2004, available at <http://www.fulton.asu.edu/~mre>.
- [36] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson, “Approximation algorithms for bin-packing — an updated survey,” in *Algorithm Design for Computer System Design, Springer Courses and Lecture Series No. 284*. 1984, pp. 49–106, Springer-Verlag.
- [37] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, “Approximation algorithms for the multiple knapsack problem with assignment restrictions,” *Journal of Combinatorial Optimization*, vol. 4, no. 2, pp. 171–186, June 2000.
- [38] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*, Springer, 1999.
- [39] C. Chekuri and S. Khanna, “A PTAS for the multiple knapsack problem,” in *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, CA, Jan. 2000, pp. 213–222.
- [40] D. B. Shmoys and E. Tardos, “Scheduling unrelated machines with costs,” in *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Austin, TX, Jan. 1993, pp. 448–454.