Continuous-Time Collaborative Prefetching of Continuous Media

Soohyun Oh, Beshan Kulapala, Andréa W. Richa, and Martin Reisslein

***Abstract*—The real-time streaming of bursty continuous media, such as variable-bit rate encoded video, to buffered clients over networks can be made more efficient by collaboratively prefetching parts of the ongoing streams into the client buffers. The existing collaborative prefetching schemes have been developed for discrete time models, where scheduling decisions for all ongoing streams are typically made for one frame period at a time. This leads to inefficiencies as the network bandwidth is not utilized for some duration at the end of the frame period when no video frame "fits" into the remaining transmission capacity in the schedule. To overcome this inefficiency, we conduct in this paper an extensive study of collaborative prefetching in a *continuous-time* model. In the continuous-time model, video frames are transmitted continuously across frame periods, while making sure that frames are only transmitted if they meet their discrete playout deadlines. We specify a generic framework for continuous-time collaborative prefetching and a wide array of priority functions to be used for making scheduling decisions within the framework. We conduct an algorithm-theoretic study of the resulting continuous-time prefetching algorithms and evaluate their fairness and starvation probability performance through simulations. We find that the continuous-time prefetching algorithms give favorable fairness and starvation probability performance.**

***Index Terms*—Client buffer, continuous media, continuous-time, fairness, playback starvation, prefetching, prerecorded media, traffic smoothing, video streaming.**

I. Introduction

THE REAL-TIME streaming of continuous media over networks, such as the future Internet and next generation wireless systems, is a challenging problem mainly due to (*i*) the periodic playout deadlines, and (*ii*) the traffic variability. NTSC video, for instance, has a periodic playout deadline (frame period) every 33 msec while PAL video has a 40 msec frame period, whereby a new video frame has to be delivered every frame period to ensure continuous playback. A frame that is not delivered in time is essentially useless for the media playback and results in interruptions of the playback. The continuous media are typically compressed (encoded) to reduce their bit rates for network transport. The efficient encoders, especially for video, produce typically highly variable traffic (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 [1]. As a result, allocating network resources based on the average bit rates would result in frequent playout deadline misses since the larger frames could not be delivered in time, while allocating resources based on the largest frame size would result in low average network utilization.

Manuscript received October 10, 2006; revised September 22, 2007. This work is supported in part by the National Science Foundation under Grant No. Career ANI-0133252.

The authors are with Arizona State University, AZ 85287-5706 USA (e-mail: soohyun@asu.edu; beshan@asu.edu; aricha@asu.edu; reisslein@asu.edu)

Digital Object Identifier 10.1109/TBC.2007.910921

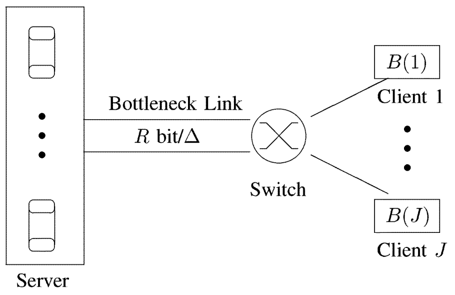


Fig. 1. J prerecorded video streams are multiplexed over a bottleneck link of capacity R bits/Δ, and prefetched into client buffers of capacity $B(j)$ bits, $j = 1, \ldots, J$.

To overcome these challenges, prefetching (work-ahead) schemes have been developed that exploit the facts that (*i*) a large portion of the media are prerecorded, and (*ii*) that many of the media playback (client) devices have storage space, by prefetching parts of an ongoing media stream. The prefetching builds up prefetched reserves in the client buffers, and these reserves 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 [2]–[19], 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 [20]–[29], 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 [28]. 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 [30].

A common characteristic of the existing collaborative prefetching schemes is that they are designed based on a *discrete*-time model, that is, the scheduling decisions are computed at discrete time instants. In particular, the majority of the exiting collaborative prefetching schemes calculate the transmission schedule on a per-frame period basis [20], [24]–[29]. These schemes essentially consider each frame period as a new scheduling problem and attempt to fit as many video frames as possible in the transmission capacity (link bit rate in bit/sec $\times$ frame period in sec) available in the currently considered frame period. Frames are generally not scheduled across frame periods. This leads to inefficiencies as some amount of available transmission capacity at the end of a frame period goes unused as no frame is small enough to fit into the remaining capacity. Similar inefficiencies arise when frames are first smoothed over an MPEG Group-of-Pictures (GoP) and then scheduled using a JSQ like strategy executed on a per-frame period basis [22], or when the scheduling decisions for the transmission of individual frames are computed at discrete slot times [21], [23].

Another common characteristic of the existing collaborative prefetching schemes is that they were primarily designed and studied for minimizing the *number of lost frames*, i.e., the frame loss probability. Since video frames requiring many bits for encoding have generally a larger impact on the delivered video quality than video frames requiring only few encoding bits, the *number of lost bits*, i.e., the information (bit) loss probability, is an important metric when streaming compressed video.

In this paper we conduct an extensive study on collaborative prefetching in a *continuous-time* model considering both the frame loss probability and the information loss probability. The continuous-time model overcomes the outlined inefficiency of the discrete-time model by allowing for video frames to be transmitted continuously across frame periods. The discrete playout deadlines of the video frames need to be still considered, even in the continuous-time model, and our algorithms ensure that frames that would not meet their deadline are not transmitted. We specify a generic framework for continuous-time prefetching and an array of priority functions to be used for making scheduling decisions within the framework. The priority functions are based on numbers of transmitted video frames and numbers of transmitted (or lost) video information bits. We conduct an algorithm-theoretic study of the resulting continuous-time prefetching algorithms and evaluate their fairness as well as frame and information starvation probability performance through simulations. We find that the continuous-time prefetching algorithms give favorable fairness performance and significantly reduce the starvation probability compared to discrete-time prefetching schemes.

This paper is organized as follows. In the following section we describe the problem set-up and introduce the notations used in the continuous-time modeling of the collaborative prefetching. In Section III, we develop the generic framework for continuous-time collaborative prefetching and introduce a wide array of priority functions to be used within the framework. In Section IV, we conduct an algorithm-theoretic complexity analysis of the prefetching framework. In Section V, we conduct an algorithm-theoretic analysis of the prefetching algorithm using the frame-based priority function in the prefetching framework, while in Section VI we analyze the bit-based prefetching algorithms. In Section VII, we present simulation results illustrating the fairness and starvation probability performance of the continuous-time prefetching algorithms and compare with the discrete time algorithms. In Section VIII, we summarize our findings.

II. Continuous-Time Model and Notations

In our system model, which is illustrated in Fig. 1, a number of prerecorded continuous media streams are stored in mass storage in the server. We assume that the server is in one of the following states, *busy-with-transmission*, *busy-with-scheduling*, or *idle*. When the server is in the busy-with-transmission state, a frame is being transmitted to the corresponding client. When the transmission of the frame is complete, i.e., when the server has sent out the last bit of the frame, the server becomes idle. If the server has more frames to be delivered, then the server enters the busy-with-scheduling state. The busy-with-scheduling state can be masked by overlapped it with the preceding busy-with-transmission state, i.e., during transmission of a frame n to client j, the server computes the next frame to be transmitted. We assume in this paper that the time that it takes to decide on the scheduling of the next frame is less than or equal to the transmission time of the smallest video frame, which is reasonable given our time complexity results for the prefetching algorithms. This allows for masking of the schedule computing time for all frames.

In our continuous-time model, frame deadlines are still chosen from a *discrete*, evenly spaced set of time slots of length Δ, the common basic frame period of the videos in seconds. However, scheduling and transmission of frames proceed in a *continuous-time* fashion. A frame will be scheduled while a previously scheduled frame is being transmitted, and right after the current transmission ends (i.e., its last bit is sent out), the newly scheduled frame will start being transmitted. Once a video frame arrives at a client, it is placed in the client's prefetching buffer. For our model we assume that the time is set to be 0 (i.e., $t = 0$) when the server initiates scheduling and transmitting frames for J clients. We also assume that at time $t = 0$, the first frame of each stream has deadline Δ. In other words, the first frame of a stream should arrive at the corresponding client before or on time $t = \Delta$ to be decoded and displayed during the first frame period $[\Delta, 2\Delta)$. If at time $t = \Delta$, a complete video frame with deadline Δ is not in the prefetch buffer, the client suffers playback starvation and loses the frame.

For simplicity of notation, which we summarize in Table I, we assume that time is normalized by the frame period Δ. The frame with playback deadline Δ, is removed from the buffer and decoded at normalized time $t = 1$, and displayed during time $1 \leq t < 2$. Each client displays the first frame (frame 1) of its video stream during the time period [1, 2), and then removes the second frame from its prefetch buffer at time $t = 2$ and displays it during time period [2, 3), and so on. Formally, we let $D_n(j)$ denote the deadline of frame n of stream j and note that with our assumptions, $D_n(j) = n$.

If a video frame with deadline n cannot be scheduled before or at time $n - x_n(j)/R$, where $x_n(j)$ denotes the size of frame with deadline n (or n-th frame) of stream j, then it is dropped at the server and the client will suffer a frame loss during time period $[n, n+1)$. We let $\theta_t(j), j = 1, \ldots, J$, denote the lowest indexed frame for stream j that is still on the server and has not been dropped at time t. In other words, $\theta_t(j)$ is the frame with the earliest playout deadline that can still be transmitted to meet

TABLE I
DEFINITIONS OF NOTATIONS

J	Number of video streams in progress
Δ	Frame period of the videos in seconds
R	Transmission capacity of the bottleneck link in bit per frame period
$D_n(j)$	Deadline of frame n of video stream $j,\ j = 1, \ldots, J$
$x_n(j)$	Size (in bit) of the frame n of video stream j
$x_{\max}(j)$	Size (in bit) of largest frame of video stream j
$x_{\min}(j)$	Size (in bit) of smallest frame of video stream j
$\bar{x}(j)$	Average frame size of video j
P	Largest peak-to-mean ratio of the ongoing streams, i.e., $P = \max_j x_{\max}(j)/\bar{x}(j)$
$B(j)$	Capacity of the prefetch buffer of client j in bit
$p_t(j)$	Number of prefetched video frames in the prefetch buffer of client j at time t
$b_t(j)$	Number of bits in the prefetch buffer of client j at time t
$\theta_t(j)$	Frame index with the earliest playout deadline that can still be transmitted to meet its deadline
$h_t(j)$	Number of frames that have been transmitted to client j up to time t
$h_t^i(j)$	Number of bits that have been transmitted to client j up to time t
$q_t(j)$	Number of frames of stream j that have missed their playout deadline up to time t
$q_t^i(j)$	Number of bits of stream j that have missed their playout deadline up to time t
$P_{\text{loss}}^f(j)$	Frame loss probability
$P_{\text{loss}}^i(j)$	Information loss probability

its deadline. Let θ_t^* denote the earliest deadline frame among the ongoing streams on the server at time t, i.e., $\theta_t^* = \min_j \theta_t(j)$.

We assume that initially, at time $t = 0$, all J streams have an infinite number of frames to be transmitted to the corresponding clients and their prefetch buffers are empty. We let $h_t(j),\ j = 1, \ldots, J$, denote the number of video frames that have been transmitted to client j up to time t (and note the initial condition $h_0(j) = 0$ for $j = 1, \ldots, J$). Let $h_t^i(j), j = 1, \ldots, J$, denote the number of bits that have been transmitted to client j up to time t.

We let $q_t(j),\ j = 1, \ldots, J$, denote the number of video frames of stream j that have missed their playout deadline up to time 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. Let $q_t^i(j), j = 1, \ldots, J$, denote the number of bits of stream j that have missed their playout deadline up to time t. We let $b_t(j),\ j = 1, \ldots, J$, denote the number of bits in the prefetch buffer of client j at time t (and note that $b_0(j) = 0$ for all clients $j = 1, \ldots, J$). Note that the buffer constraint $b_t(j) \leq B(j)$ must be satisfied for all clients $j,\ j = 1, \ldots, J$, for all times t. Although the case of limited prefetch buffer capacities is more realistic, for ease of analysis we initially consider the case of unlimited prefetch buffer capacities (or sufficiently large buffer capacities).

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

$$P_{\text{loss}}^f(j) = \lim_{t\to\infty} \frac{q_t(j)}{h_t(j) + q_t(j)} = \lim_{t\to\infty} \frac{q_t(j)}{t}. \tag{1}$$

Similarly, we define the *information loss probability* of client j as

$$P_{\text{loss}}^i(j) = \lim_{t\to\infty} \frac{q_t^i(j)}{h_t^i(j) + q_t^i(j)} = \lim_{t\to\infty} \frac{q_t^i(j)}{\sum_{\ell=1}^{t} x_\ell(j)}. \tag{2}$$

We define the average frame loss probability as $P_{\text{loss}}^f = 1/J \cdot \sum_{j=1}^J P_{\text{loss}}^f(j)$ and the average information loss probability as $P_{\text{loss}}^i = 1/J \cdot \sum_{j=1}^J P_{\text{loss}}^i(j)$.

III. Prefetching Framework and Priority Functions

In our effort to study the problem of scheduling video frame transmissions for the collaborative prefetching of continuous media in continuous time, we first present a generic prefetching framework. This framework is divided into two basic categories according to the capacities of clients—unlimited buffer capacities and limited buffer capacities. We use the unlimited buffer capacity scenario to study the effects of the limited network capacity on the video prefetching. With limited prefetch buffer capacities, we can study the effects of both the limited network capacity and the limited client buffer capacity. In our algorithm-theoretic analysis of the proposed prefetching algorithms we consider the unlimited prefetch buffer capacity, whereas in our simulations we examine the effect of limited prefetch buffer capacities.

In this section, we first present the generic scheduling frameworks for the continuous-time model and subsequently introduce the different priority functions that are employed within the frameworks. Before we introduce the scheduling framework and priority functions we briefly discuss the difficulties of finding optimal solution to the continuous-time prefetching of continuous media. The objectives of a scheduling algorithm for the continuous-time model are (*i*) to minimize the total number of lost frames (or minimize the total number of lost bits) and (*ii*) to treat the clients fairly. If we consider only objective (*ii*), fairness among clients, then it can be easily achieved by sending one frame per stream. However, this problem becomes considerably more involved when we consider the first objective: If we only try to maximize the number of transmitted frames, then there is an analogy between this problem and the standard job scheduling problem on a single processor. The standard job scheduling problem is defined as follows. There is a stream of tasks. A task may arrive at any time and is assigned a value that reflects its importance. Each task has an execution time that represents the amount of time required to complete the task, and a deadline by which the task is to complete execution. The goal is to maximize the sum of the values of the tasks completed by

```
Unlimited–Buffer–Capacity–Case
    If the server becomes idle at time $t$
1) For all streams $j = 1, \ldots, J$, compute their priorities $PR_t(j)$.
2) Select the stream with the highest associated priority.
3) If the first frame (unscheduled and undropped yet) of the selected stream in the server can meet its deadline,
   then schedule the frame for transmission.
4) Otherwise, drop the frame and goto step 1.
```

Fig. 2. Prefetch algorithm framework for case of unlimited client prefetch buffer capacity.

```
Limited–Buffer–Capacity–Case
    If the server becomes idle at time $t$
1) For all streams $j \in \mathcal{J}$, compute their priorities $PR_t(j)$, where $\mathcal{J}$ contains initially all $J$ streams.
2) Select the stream with the highest associated priority.
3) If the first frame (unscheduled and undropped yet) of the selected stream in the server can meet its deadline,
   then
    3.1 If the frame size is not larger than the available space of the corresponding client's buffer, then schedule
        the frame to be transmitted.
    3.2 Otherwise, remove the stream from $\mathcal{J}$ and goto step 1.
4) Otherwise, drop the frame and goto step 1.
```

Fig. 3. Prefetch algorithm framework for case of limited client prefetch buffer capacity.

their deadlines. The analogy between the two scheduling problems is that the J streams can be viewed as one stream where some of tasks (frames) have the same deadline. Each frame is assigned a value which is equal to one, and an execution time which depends on the frame size. The off-line version of this problem, where the time constraints of all tasks are given as input, is known to be NP-hard [31]. A number of studies have proposed on-line heuristics to handle situations when the system is overloaded [31]–[33]. It is known that the *earliest deadline first* policy gives the optimal solution when the system is underloaded [34].

A. Generic Prefetching Frameworks

Fig. 2 describes the generic prefetch algorithm framework for the case of unlimited prefetch buffer capacity. In this case the only consideration is the discrete deadline of each frame. The priority of a stream will be defined by a priority function as detailed shortly. In the algorithm framework in Fig. 2, we neglect the masking of the busy-with-scheduling state, i.e., assume the schedule computation takes negligible time. If this time is significant, the algorithm execution would need to start earlier such that it is completed when the server completes the current frame transmission at time t. The status of the various counters of numbers of transmitted or lost bits or frames at time t is known at this earlier time when the algorithm starts executing and to be used in the algorithm's calculations.

In the case of limited client prefetch buffer space, the server must be aware of the clients' prefetch buffers to prevent loss caused by an overflow at the client side. Hence the server cannot schedule frames for a client whose prefetching buffer is full even though network capacity is available, as detailed in the algorithm framework in Fig. 3.

Importantly, the outlined generic prefetching frameworks collaboratively consider all ongoing streams by computing the priorities $PR_t(j)$ for all streams $j = 1, \ldots, J$, and selecting the stream with the highest priority for frame transmission.

B. Priority Functions

Within the presented algorithm framework we conduct an extensive study of prefetching algorithms that use different priority functions $PR_t(j)$. We broadly categorize the priority functions into being based on the number of video *frames* or the number of video information *bits*. The aim of the frame-based priority function is primarily to minimize P^f_{loss}, whereas the bit-based priority functions aim at minimizing P^i_{loss}.

In the remainder of this section we introduce the intuition behind the different priority functions, which are summarized in Table II, and give a brief overview of their performance. The individual prefetching algorithms obtained by employing the different priority functions within the prefetching framework are formally specified and analyzed in Sections V and VI and evaluated through simulations in Section VII.

1) Frame-Based Priority Function:

- Transmitted Frames (TF): If we maximize the number of transmitted frames, we would expect to minimize frame loss. Therefore, we propose a greedy local algorithm that prioritizes the streams based on the number of frames transmitted so far, i.e., according to $PR_t(j) = h_t(j)$. With the TF priority function, the stream with the least number of transmitted frames has the highest priority. This algorithm strives to equalize the number of lost frames among the clients. This algorithm has a good approximation ratio when compared to an optimal offline algorithm when considering long streams. We found from our simulations that this algorithm produces essentially the same frame loss probability for all clients. However, we have observed that the information loss probabilities may slightly vary among clients.

2) Bit-Based Priority Functions: Transmitted Bits:

- Normalized Transmitted Bits (NTB): This scheduling policy is designed to minimize the loss probabilities by maximizing the number of normalized transmitted bits, i.e., $PR_t(j) = h^i_t(j)/\bar{x}(j)$, whereby the stream with the

TABLE II
SUMMARY OF CONSIDERED PRIORITY FUNCTIONS

Basis	Priority Function $PR_t(j) =$	Acronym
# of frames Smaller $PR_t(j)$ implies higher priority	$h_t(j)$	Transmitted Frames (TF)
# of transmitted bits Smaller $PR_t(j)$ implies higher priority	$\frac{h_t^i(j)}{\bar{x}(j)}$ $\frac{h_t^i(j)}{h_t^i(j)+q_t^i(j)}$ $\frac{h_t^i(j)\cdot[p_t(j)+1]}{\bar{x}(j)}$ $\frac{h_t^i(j)\cdot[p_t(j)+1]}{h_t^i(j)+q_t^i(j)}$	Normalized Transmitted Bits (NTB) Ratio of Transmitted Bits (RTB) Weighted Normalized Transmitted Bits (WNTB) Weighted Ratio of Transmitted Bits (WRTB)
# of lost bits Larger $PR_t(j)$ implies higher priority	$\frac{q_t^i(j)}{\bar{x}(j)}$ $\frac{q_t^i(j)}{h_t^i(j)+q_t^i(j)}$ $\frac{q_t^i(j)}{[h_t^i(j)+q_t^i(j)]\cdot[p_t(j)+1]}$	Normalized Lost Bits (NLB) Ratio of Lost Bits (RLB) Weighted Ratio of Lost Bits (WRLB)

smallest $PR_t(j)$ has the highest priority. One shortcoming of the priority function based on the average frame size $\bar{x}(j)$ is that they may not accurately reflect the actual average data rate: The average frame size $\bar{x}(j)$ represents the average frame size of the entire video stream. However, the average frame size in the part of the stream that has been transmitted in the recent past and is currently considered for transmission may differ from the average frame size over the entire stream. To overcome this problem we propose the following alternative.

- Ratio of Transmitted Bits (RTB): This priority function is based on the ratio of the number of transmitted bits to the total number of bits that have been considered to be scheduled so far i.e., $PR_t(j) = h_t^i(j)/\{h_t^i(j) + q_t^i(j)\}$. This priority function is designed to minimize the loss probabilities by maximizing the number of transmitted bits.
- Weighted Normalized Transmitted Bits (WNTB): The drawback of the preceding bit-based priority functions is that the frame deadlines are not directly taken into consideration, and hence they may cause increased information loss by prefetching frames far in the future rather than scheduling an imminent frame. As a consequence, unnecessary losses may occur and overall performance may be degraded. Hence, we propose a weighing of the number of transmitted bits by multiplying them with the number of frames currently in the corresponding prefetch buffer, i.e., $PR_t(j) = h_t^i(j) \cdot (p_t(j) + 1)/\bar{x}(j)$. If a client has some frames in its prefetch buffer, it can display frames without starvation for some time. In contrast, an empty prefetch buffer implies that the first frame of the corresponding stream should be immediately transmitted to avoid starvation. This WNTB priority function ensures that even clients with a large number of successfully transmitted bits can still be chosen by the algorithm for transmission if they are close to starvation.
- Weighted Ratio of Transmitted Bits (WRTB) This priority function combines the weighing with the normalization by the actual number of bits considered for scheduling, i.e., uses the priority function $PR_t(j) = h_t^i(j) \cdot (p_t(j) + 1)/\{h_t^i(j) + q_t^i(j)\}$.

3) Bit-Based Priority Functions: Lost Bits: To ensure an extensive evaluation of bit-based priority functions we also consider priority functions employing the number of lost bits. More specifically, we consider the lost bits based counterparts of the NTB, RTB, and WRTB policies.

- Normalized Lost Bits (NLB): The NLB priority function considers the amount of lost bits normalized by the average frame size of the video stream, i.e., $PR_t(j) = q_t^i(j)/\bar{x}(j)$. The stream with the largest accumulated normalized lost bits has the highest priority.
- Ratio of Lost Bits (RLB): The RLB priority function is based on the ratio of the number of lost bits to the total number of bits that have been considered to be scheduled so far, i.e., $PR_t(j) = q_t^i(j)/\{h_t^i(j) + q_t^i(j)\}$. The stream with the largest ratio has the highest priority.
- Weighted Ratio of Lost Bits (WRLB) This priority function combines the weighing by the number of frames currently in the prefetch buffer, achieved through the factor $p_t(j)+1$, with the RLB priority, i.e., the WRLB priority function is $PR_t(j) = q_t^i(j)/\{[h_t^i(j) + q_t^i(j)] \cdot [p_t(j) + 1]\}$.

IV. Time Complexity of Generic Prefetch Algorithm Framework

In this section we analyze the computing time complexity of the generic prefetch algorithm framework. It suffices to analyze time complexity of the generic algorithm, since all the proposed prefetch algorithms follow the basic framework of the generic algorithm but use different priority functions, each of which can be computed in constant time.

In the following, we compute the time taken for each step in the algorithm frameworks presented in Figs. 2 and 3, thus determining the time complexity of the algorithm used for scheduling the next frame for transmission. Since it takes only a constant time to compute the priority of each stream (or client) at Step

```
TF(t)
If the server becomes idle at time $t$, then
  1) For all streams $j = 1, \cdots, J$, compute
      1.1 if $\theta_t(j) \le \lfloor t \rfloor$, then
              $q'_t(j) = q_t(j) + \lfloor t \rfloor - \theta_t(j) + 1$
              $\theta_t(j) = \theta_t(j) + 1$
      1.2 else $q'_t(j) = q_t(j)$
      1.3 $PR_t(j) = \theta_t(j) - q'_t(j)$
  2) $\mathcal{M} = \{i | PR_t(i) = \min_{j \in J} PR_t(j)\}$ //streams with the minimum value of $PR_t(j)$
  3) $\mathcal{M}^* = \{k | \theta_t(k) = \min_{i \in \mathcal{M}} \theta_t(i)\}$ //streams that have the earliest deadline in $\mathcal{M}$
  4) Let $k^* \in \mathcal{M}^*$ be the index of the stream such that $x_{\theta_t(k^*)}(k^*) = \min_{k \in \mathcal{M}^*} x_{\theta_t(k)}(k)$
  5) If $t + \frac{x_{\theta_t(k^*)}(k^*)}{R} \le D_{\theta_t(k^*)}(k^*)$, then //the frame meets its deadline
      5.1 Schedule the frame of stream $k^*$
      5.2 $\theta_t(k^*) = \theta_t(k^*) + 1$
  6) else //the frame does not meet its deadline
      6.1 Drop the frame of stream $k^*$
      6.2 $q_t(k^*) = q'_t(k^*) + 1$
      6.3 $\theta_t(k^*) = \theta_t(k^*) + 1$
      6.3 Go to Step 1
```

Fig. 4. The least Transmitted Frames (TF) prefetch algorithm.

1 for all J streams, it takes $O(J)$ time to compute the current priorities for all J streams. Finding the stream with the highest priority takes $O(J)$ time at Step 2 and checking if the first frame of the selected stream satisfies the deadline constraint takes another constant time. In a naive implementation of the algorithms, we may need many iterations until we find a frame to be scheduled that can meet its deadline.

We can optimize the algorithms by initially checking the deadline constraint for all streams and dropping any frames whose deadline is smaller than t, before we compute the priority of each stream. Note that according to this implementation, there cannot be any undropped frames with deadline smaller than $t-1$ at time t (since the size of the previously scheduled frame is at most $\max_j x_{max}(j) \le R$, ensuring that it can be transmitted within one frame period) and hence the added complexity for checking the deadlines of the unscheduled frames is $O(RJ) = O(J)$ since R is a constant. The overall complexity of the algorithms would then be $O(J)$ for one execution of the continuous-time prefetch algorithm, which decides on the scheduling of a video frame.

The computational complexity of the common discrete-time prefetch algorithms is given in terms of the computational effort required to compute the video frame transmission schedule for a frame period of length Δ. In order to facilitate the comparison of discrete-time and continuous-time prefetch algorithms we characterize the computational complexity of the continuous-time prefetch framework for a frame period as follows. We let $x_{\min} = \min_j x_{\min}(j)$ denote the smallest frame among the J ongoing streams and note that at most $R/x_{\min}$ frames can be scheduled per frame period. Hence, the worst-case computational complexity of the continuous-time prefetching framework is $O((R/x_{\min})J) = O(J)$ for computing the schedule for a frame period. The corresponding complexities are $O(J)$ for JSQ [28], $O(\Delta J \log J) = O(J \log J)$ for DC [21], and $O(P^2 J) = O(J)$ for BP [27]. Thus, continuous-time prefetching has comparable computational complexity to discrete-time prefetching.

V. Specification and Analysis of TF Algorithm

In this section we specify in detail and analyze the prefetch algorithm obtained by employing the frame-based TF priority function in the algorithm framework. The resulting least Transmitted Frames (TF) prefetch algorithm schedules frames as follows: At time t, from among all ongoing streams $j = 1, \ldots, J$, pick the streams that have the minimum number of transmitted frames $h_t(j)$. If there is more than one stream with the minimum $h_t(j)$, then pick the stream that has the frame with the earliest deadline and schedule this frame. If there is still a tie, then pick the stream that has the smallest frame and schedule its frame. The TF prefetch algorithm has the primary goal to achieve small frame loss probabilities and is specified in Fig. 4. The TF algorithm defines the priority of stream j at time t as

$$PR_t(j) = h_t(j) = \theta_t(j) - q'_t(j), \quad \forall j = 1, \cdots, J.$$

For the convenient calculation of this priority we define the counter $q'_t(j)$ to be the number of video frames of stream j that have already missed their playout deadlines up to time t, including the frames that have not yet been identified and dropped by the algorithm. This $q'_t(j)$ counter, which is maintained as specified in Fig. 4, is necessary since the number of dropped frames $q_t(j)$ is updated only when a client wants to retrieve a frame from its buffer. Frames that are still in the server but cannot be delivered to the corresponding clients by their deadlines are hence not necessarily reflected by the counter $q_t(j)$. The equivalence $h_t(j) = \theta_t(j) - q'_t(j)$ holds as long as the frames for each stream are transmitted in their deadline order, which is the case for our prefetch policies.

With the TF prefetch algorithm, the stream with the smallest $PR_t(j)$ value has the highest priority. Note that if the value of $\theta_t(j)$ is the same for all the streams, then the stream with the most frame losses $q'_t(j)$ has the highest priority. If any two streams, say stream i and j, have the highest priority, then the TF algorithm selects the stream with $\min\{\theta_t(i), \theta_t(j)\}$. At the end of the TF algorithm, one stream is selected to schedule and transmit its lowest indexed frame.

A. Fairness Properties of TF Prefetch Algorithm

The following two claims show the fairness of the proposed TF algorithm.

Lemma 1: At any time t, $\max_j h_t(j) - \min_j h_t(j) \leq 1$.

Proof: Suppose that at time t, $\max_j h_t(j) - \min_j h_t(j) = 1$. Let $j_{\max} = \arg\max_j h_t(j)$, and analogously let $j_{\min} = \arg\min_j h_t(j)$. For convenience we drop the subscript t in the following. Since $h(j)$ is equal to $PR(j)$, at time t,

$$\{\theta(j_{\max}) - \theta(j_{\min})\} - \{q'(j_{\max}) - q'(j_{\min})\} = 1$$

Since $h(j_{\max}) > h(j_{\min})$, stream $j_{\min}$ would be considered prior to stream $j_{\max}$. Suppose that frame $\theta(j_{\min})$ misses its deadline. Then, stream $j_{\min}$ still has a lower $PR(j_{\min})$ than stream $j_{\max}$, so that stream $j_{\min}$ would be considered prior to stream $j_{\max}$. If stream $j_{\min}$ is considered again, then stream $j_{\min}$ will schedule frame $\lceil t+1 \rceil$ at this time. Now, streams $j_{\max}$ and $j_{\min}$ have the same priority. ■

The above lemma shows that the proposed algorithm minimizes the difference between the maximum number of transmitted frames and the minimum number of transmitted frames. We define the class of *fair algorithms* as being the set of algorithms for which Lemma 1 holds.

Corollary 1: At any time t, $\max_j q\prime(j) - \min_j q\prime(j) \leq 2$.

Proof: Let $q\prime(j_{\max}) = \max_j q\prime(j) = y+2$ and $q\prime(j_{\min}) = \min_j q\prime(j) = y$ at time t. Suppose that $q\prime(j_{\max}) - q\prime(j_{\min}) = 3$ at time $t+\alpha$, where $0 < \alpha < 1$. (Let us assume that only one frame has been transmitted during time period $[t, t+\alpha]$.) I.e., during $[t, t+\alpha]$, stream $j_{\max}$ drops frame $\theta_t(j_{\max})$ while stream $j_{\min}$ does not drop any frame. There are two cases to be considered.

Case 1) Stream $j_{\min}$ is considered prior to stream $j_{\max}$ during $[t, t+\alpha]$.

$$\begin{aligned} PR(j_{\max}) - PR(j_{\min}) &> 0 \\ \{\theta(j_{\max}) - q\prime(j_{\max})\} - \{\theta(j_{\min}) - q\prime(j_{\min})\} &> 0 \\ \{\theta(j_{\max}) - \theta(j_{\min})\} - \{(n+2) - n\} &> 0 \\ \theta(j_{\max}) - \theta(j_{\min}) &> 2 \end{aligned}$$

Since the deadline of frame $\theta(j_{\max})$ is later than that of frame $\theta(j_{\min})$, stream $j_{\max}$ does not drop a frame during time $[t, t+\alpha]$.

Case 2) Stream $j_{\max}$ is considered prior to stream $j_{\min}$ during $[t, t+\alpha]$

$$\begin{aligned} PR(j_{\max}) - PR(j_{\min}) &< 0 \\ \{\theta(j_{\max}) - q\prime(j_{\max})\} - \{\theta(j_{\min}) - q\prime(j_{\min})\} &< 0 \\ \{\theta(j_{\max}) - \theta(j_{\min})\} - \{(n+2) - n\} &< 0 \\ \theta(j_{\max}) - \theta(j_{\min}) &< 2 \end{aligned}$$

We divide this case 2 into two sub-cases **(i)** $\theta(j_{\max}) = \theta(j_{\min})$ and **(ii)** $\theta(j_{\max}) = \theta(j_{\min}) + 1$. **(i)** $h(j_{\max}) = \theta(j_{\max}) - n + 2$ and $h(j_{\min}) = \theta(j_{\min}) - n$. Then $h(j_{\max}) - h(j_{\min}) = 2$ which violates Lemma 1. **(ii)** $\theta(j_{\max}) > \theta(j_{\min})$. If frame $\theta(j_{\max})$ misses its deadline, so does frame $\theta(j_{\min})$. Hence the corollary holds. ■

Lemma 1 and Corollary 1 taken together show that the TF prefetching algorithm distributes the frame losses evenly among clients. An algorithm for which Lemma 1 holds, but for which Corollary 1 does not hold, could treat clients unfairly by dropping more frames for some clients than for other clients while maintaining the number of transmitted frames equalized across all clients.

B. Efficiency Properties of TF Prefetch Algorithm

We define a *round* to be a time interval during which each stream schedules and transmits exactly one frame using the TF algorithm. More specifically, we define the round such that all J streams have transmitted the same number of frames at the beginning of a round, and each stream has increased its number of transmitted frames by one at the end of the round. Intuitively, if we minimize the time required to complete each round then we minimize starvation. Without loss of generality, we suppose that one round started at time 0 and it took time t to complete this round. If $t \leq 1$, then all J streams would transmit their frames—exactly one frame per stream—without any frame losses. If $r < t < r+1$ where r is an integer, then each stream that transmits its frame at time $t < r$ will experience exactly $r-1$ frame losses, and each stream that transmits its frame at time $t \geq r$ will experience exactly r frame losses. Intuitively, any fair algorithm must proceed in rounds. Otherwise, Lemma 1 would be violated. We label the rounds according to the number of transmitted frames at the start of the round, i.e., round n starts when one of the streams has transmitted n frames (while the other streams still have transmitted $n-1$ frames), and round n ends when all J streams have transmitted exactly n frames.

We analyze the amount of time required to complete one round. We define α to be the maximum value of the ratio of the largest frame size to the smallest frame size over all streams. i.e., $\alpha = \max_j(\max_n x_n(j)/\min_n x_n(j))$. We show that the proposed TF algorithm is an α-approximation on minimizing the time required to complete a round. More specifically, we compare round n for our proposed algorithm and an optimal algorithm. Assume that f_j is the index of the scheduled frame for stream j in this round using the proposed algorithm. Let $t(f_j)$ denote the time required to transmit frame f_j, i.e., $t(f_j) = x_{f_j}(j)/R$. Then the total time taken to complete this round is $T = \sum_j t(f_j)$. Let f_j^* be the index of the scheduled frame for stream j that minimizes the total time for this round, i.e., the optimal solution for this round is $T^* = \sum_j t(f_j^*)$. If no stream drops its frame during this round, then $T = T^*$. Let K be the set of streams such that $f_j^* \neq f_j$. Note that $t(f_j) \leq \alpha \cdot t(f_j^*)$ since $x_{f_j}(j)/x_{f_j^*}(j) \leq \alpha$. Thus,

$$\begin{aligned} T &= \sum_j t(f_j) && (3) \\ &\leq \sum_j t\left(f_j^*\right) + \sum_{i \in K} |t(f_i) - t(f_i^*)| && (4) \\ &\leq \sum_j t\left(f_j^*\right) + (\alpha - 1)\sum_{i \in K} t\left(f_i^*\right) && (5) \\ &\leq \sum_j t\left(f_j^*\right) + (\alpha - 1)\sum_j t\left(f_j^*\right) && (6) \\ &= \alpha \sum_j t\left(f_j^*\right) && (7) \\ &= \alpha T^*, && (8) \end{aligned}$$

which proves the following theorem.

```
NTB(t) If the server becomes idle at time $t$, then
1) For all stream $j = 1, \cdots, J$, compute $PR_t(j) = r_t(j)$
2) $\mathcal{M} = \{i | PR_t(i) = \min_{j \in J} PR_t(j)\}$ //streams with the minimum value of $PR_t(j)$
3) $\mathcal{M}^* = \{k | \theta_t(k) = \min_{i \in \mathcal{M}} \theta_t(i)\}$ //streams that have the earliest deadline in $\mathcal{M}$
4) Let $k^* \in \mathcal{M}^*$ be the index of the stream such that $x_{\theta_t(k^*)}(k^*) = \max_{k \in \mathcal{M}^*} x_{\theta_t(k)}(k)$
5) If $t + \frac{x_{\theta_t(k^*)}(k^*)}{R} \le D_{\theta_t(k^*)}(k^*)$, then //the frame meets its deadline
    5.1 Schedule and transmit the frame of stream $k^*$
    5.2 $\theta_t(k^*) = \theta_t(k^*) + 1$
    5.3 $h_t^i(k^*) = h_t^i(k^*) + x_{\theta_t(k^*)}(k^*)$
6) else //the frame does not meet its deadline
    6.1 Drop the frame of stream $k^*$
    6.2 $q_t^i(k^*) = q_t^i(k^*) + x_{\theta_t(k^*)}(k^*)$
    6.3 $\theta_t(k^*) = \theta_t(k^*) + 1$
    6.4 Go to Step 1
```

Fig. 5. The NTB algorithm.

Theorem 1: The TF prefetch algorithm is an α-approximation on minimizing the time required to complete a round, where α is the maximum ratio of the largest frame size to the smallest frame size from the ongoing streams.

Theorem 2: Any fair algorithm requires at most $\log J$ time to complete a round with high probability.

Proof: We naturally assume that the capacity of the link R is large enough to accommodate the sum of the average frame sizes, i.e., that $\sum_{j=1}^{J} \bar{x}(j) \le R$. We also assume that the probability that the traffic from the ongoing J streams exceeds the link capacity R is less than a small constant ϵ, i.e.,

$$Pr\left[\sum_{j=1}^{J} x(j) > R\right] < \epsilon, \tag{9}$$

where $x(j)$ is the size of an arbitrary frame of stream j.

We recall that we have defined a *unit of time* to be the frame period with length Δ. For example, one time unit starting at time t is the interval $[t, t+\Delta)$ and two time units starting at time t is the interval $[t, t+2\Delta)$ and so on. We show that any fair algorithm completes one round in $\log J$ time units with high probability as follows.

$$\begin{aligned}
&\Pr[\text{At least one stream could not send} \\
&\quad \text{its frame in } \log J \text{ units}] \\
&\le \Pr[\text{At least one stream could not send} \\
&\quad \text{its frame in the first time unit}] \\
&\quad \times \Pr[\text{At least one stream could not send} \\
&\quad \text{its frame in the second time unit}] \\
&\quad \times \cdots \\
&\quad \times \Pr[\text{At least one stream could not send} \\
&\quad \text{its frame in the } \lfloor \log J \rfloor\text{-th time unit}] \\
&\le \epsilon^{\log J} = J^{\log \epsilon}.
\end{aligned}$$

Hence, with high probability it takes at most $\log J$ time to send exactly one frame per stream. This analysis holds for any scheduling algorithm as long as the algorithm is fair. ■

The preceding results imply that we have at most $(\lceil \log J \rceil - 1)$ frame losses per client during one round with high probability. At any time t our algorithm has sent at least $\lfloor t/\log J \rfloor$ frames per client and an optimal algorithm has sent at most $\lfloor t \rfloor$ frames per client. Thus, the number of transmitted frames for each client according to the optimal algorithm is at most $\log J$ times larger than the number of frames transmitted according to our algorithm.

VI. Specification and Analysis of Transmitted Bit-Based Algorithms

In this section we consider the prefetch algorithms obtained with the priority functions based on the number of transmitted bits in the prefetch algorithm framework. The resulting bit-based prefetching algorithms have the primary goal to achieve small information loss probabilities.

A. *Normalized Transmitted Bits (NTB) Prefetch Algorithm*

We define $r_t(j)$ to be the normalized number of transmitted bits for stream j up to time t, i.e.,

$$r_t(j) = \frac{h_t^i(j)}{\bar{x}(j)}. \tag{10}$$

We rewrite the information loss probability of stream j as

$$\begin{aligned}
P_{loss}^i(j) &= \lim_{t\to\infty} \frac{q_t^i(j)}{\sum_{\ell=1}^{t} x_\ell(j)} &(11)\\
&= 1 - \lim_{t\to\infty} \frac{h_t^i(j)}{t\bar{x}(j)} &(12)\\
&= 1 - \lim_{t\to\infty} \frac{r_t(j)}{t}, &(13)
\end{aligned}$$

where (12) follows by noting that $t\bar{x}(j)$ is approximately equal to the total number of bits in stream j up to time t (i.e., for large t: $\sum_{\ell=1}^{t} x_\ell(j) \longrightarrow t\bar{x}(j)$). Hence maximizing $r_t(j)$ minimizes the information loss probability.

We propose the least normalized transmitted bit (NTB) prefetch algorithm that selects the next frame to be transmitted at time t by selecting the stream with $\min_j r_t(j)$, i.e., we employ the priority function $PR_t(j) = r_t(j) = h_t^i(j)/\bar{x}(j)$, as detailed in Fig. 5. With the NTB algorithm, the stream with the smallest $PR_t(j)$ has the highest priority. If any two streams, say stream i and j, have the same smallest $PR_t(j)$, then the NTB policy selects the stream with the earlier playout deadline t^*. If there is again a tie, it selects the stream with the largest frame.

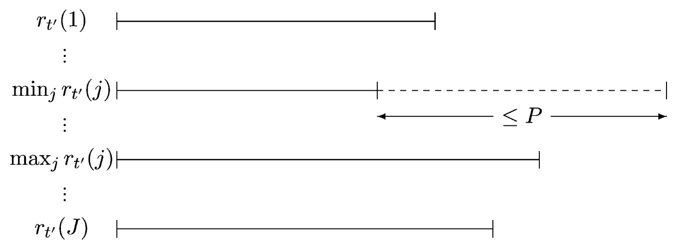


Fig. 6. Illustration of the difference between $\max_j r_{t''}(j)$ and $\min_j r_{t''}(j)$ for proof of Lemma 2.

1) Fairness Properties of NTB Prefetch Algorithm:

Lemma 2: The NTB algorithm satisfies at any time $t > 0$, $\max_j r_t(j) - \min_j r_t(j) \leq P$, where $P = \max_j(\max_n x_n(j)/\bar{x}(j))$.

Proof: At the beginning of transmission (i.e., at time 0), $r_0(j) = 0$ for all J streams and thus the claim holds initially. Suppose that after the k-th frame transmission the claim holds. Also suppose that the $(k+1)$-th frame transmission starts at time t' and the transmission completed at time t''. From the algorithm it follows that

$$\min_j r_{t''}(j) \geq \min_j r_{t'}(j). \tag{14}$$

From the induction hypothesis, we have that

$$\max_j r_{t''}(j) \leq \max\left\{\min_j r_{t'}(j) + P,\ \max_j r_{t'}(j)\right\} \tag{15}$$

$$= \min_j r_{t'}(j) + P, \tag{16}$$

as illustrated in Fig. 6. From (14) and (16), $\max_j r_{t''}(j) - \min_j r_{t''}(j) \leq P$. Hence, at any time t the claim holds. ■

2) Efficiency Properties of NTB Prefetch Algorithm: Suppose that there exists an optimal algorithm A^* that maximizes the total number of transmitted normalized bits while the transmitted normalized bits among clients are even. Let N_t be the total number of transmitted normalized bits to all J clients up to time t using algorithm A^*. Then each client j has received roughly N_t/J normalized bits and has received roughly $N_t\bar{x}(j)/J$ *actual* bits up to time t. Since we are fully utilizing the bandwidth, the total amount of actual bits that our algorithm has transmitted up to time t is the same as the total amount of actual bits that algorithm A^* has transmitted. Now, we claim that at time t, $\min_j r_t(j)$ of our proposed NTB algorithm is no less than $N_t/J - P$, where $P = \max_j(\max_n x_n(j)/\bar{x}(j))$. Suppose that $\min_j r_t(j) < N_t/J - P$, say $\min_j r_t(j) = N_t/J - P - \alpha$ for some positive number α. Since we must have $\max_j r_t(j) \geq N_t/J$,

$$\max_j r_t(j) - \min_j r_t(j) \geq \left(\frac{N_t}{J}\right) - \left(\frac{N_t}{J} - P - \alpha\right) > P,$$

which violates Lemma 2 and in turn proves the following theorem.

Theorem 3: At time t, $\min_j r_t(j)$ of the NTB prefetch algorithm is no less than $N_t/J - P$, where N_t/J is the total number of transmitted normalized bits up to time t using an optimal algorithm.

B. Ratio of Transmitted Bits (RTB) Prefetch Algorithm

We rewrite the information loss probability of stream j as follows

$$P^i_{loss}(j) = \lim_{t\to\infty} \frac{q^i_t(j)}{\sum_{\ell=1}^t x_\ell(j)} \tag{17}$$

$$= 1 - \lim_{t\to\infty} \frac{h^i_t(j)}{h^i_t(j) + q^i_t(j)}. \tag{18}$$

Hence, maximizing the ratio of the number of transmitted bits to the total number of bits minimizes the information loss probability of each stream. In this spirit, we propose the least Ratio of Transmitted Bits (RTB) prefetch algorithm that schedules a frame of the stream with smallest $PR_t(j) = h^i_t(j)/(h^i_t(j) + q^i_t(j))$ for the next transmission.

The RTB policy is the same as the NTB policy, except the way we define denominator of the priority function. The RTB policy uses the ratio of the number of transmitted bits to the total number of bits to be transmitted up to time $t > 0$ while the NTB policy uses the normalized transmitted bits. Initially, we set $PR_0(j) = 1$ for all j. Let $PR_t(j)$ be multiplied by current time t. Then

$$t \cdot PR_t(j) = t \cdot \frac{h^i_t(j)}{h^i_t(j) + q^i_t(j)} = t \cdot \frac{h^i_t(j)}{t\bar{x}(j)} = r_t(j)$$

when t is large. This implies that for a sufficiently large t, the priority of RTB is almost the same as that of NTB. Hence we expect that the RTB would perform as well as the NTB. In fact, our simulation results indicate that the information loss probability of RTB is typically lower than that of NTB. This appears to be due to the fact that the RTB policy relies on the actual number of transmitted bits, instead of the average number of transmitted bits over long time horizons.

C. Weighted Normalized Transmitted Bits (WNTB) Prefetch Algorithm

The weighted normalized transmitted bits (WNTB) algorithm combines the NTB policy with a weighing according to the number of prefetched frames. The WNTB priority function is obtained by multiplying $r_t(j) = h^i_t(j)/\bar{x}(j)$ with the number of prefetched frames, i.e.,

$$PR_t(j) = \frac{h^i_t(j) \cdot (p_t(j) + 1)}{\bar{x}(j)} = (p_t(j) + 1) \cdot r_t(j).$$

The WNTB algorithm schedules a frame for the stream with the smallest $PR_t(j)$ for the next transmission. The priority function of this hybrid algorithm considers the playout deadlines of the frames in addition to the amount of information that has been transmitted so far. A stream without prefetched frames (i.e., $p_t(j) = 0$) has the highest priority. However, if more than one client have zero prefetched frames, then the client that has received less bits should have higher priority. In order to permit this differentiation, we use $(p_t(j) + 1)$ for the weighing instead $p_t(j)$. Similar reasoning leads to the weighted ratio of transmitted bits (WRTB) prefetch algorithm.

We conclude this section on the algorithm-theoretic analysis of the transmitted bits based prefetch algorithms by noting that

TABLE III
VIDEO FRAME SIZE STATISTICS: PEAK TO MEAN RATIO, STANDARD DEVIATION (IN BIT), AND COEFFICIENT OF VARIATION (STANDARD DEVIATION NORMALIZED BY MEAN FRAME SIZE). AVERAGE BIT RATE IS 64 kbps FOR ALL STREAMS

	Low Variability Video Streams			High Variability Video Streams		
Video stream	**P/M Ratio**	**Std. Dev.**	**CoV**	**P/M Ratio**	**Std. Dev.**	**CoV**
Citizen Kane	12.6	2756	1.08	16.5	3928	1.53
Die Hard I	9.2	2159	0.84	12.0	3420	1.34
Jurassic Park I	10.5	2492	0.97	17.1	3679	1.44
Silence of the Lambs	13.4	2234	0.87	28.0	3549	1.37
Star Wars IV	15.2	2354	0.92	23.9	3565	1.39
Star Wars V	10.1	2408	0.94	13.6	3625	1.42
The Firm	10.5	2581	1.01	14.7	3771	1.47
Terminator I	10.6	2156	0.84	15.2	3429	1.34
Total Recall	7.7	2260	0.88	11.9	3471	1.36
Aladdin	8.7	2343	0.92	15.7	3593	1.4
Cinderella	14.8	2276	0.89	33.0	3572	1.4
Baseball	7.8	2166	0.85	13.2	3434	1.34
Snowboard	8.7	2099	0.82	18.6	3394	1.33
Oprah	8.4	2454	0.96	15.5	3692	1.44
Tonight Show	15.9	3513	1.32	23.2	4714	1.85
Lecture-Gupta	11.9	3199	1.25	15.0	4404	1.72
Lecture-Reisslein	13.8	3058	1.19	16.0	4323	1.70
Parking Lot	9.2	2577	1.01	21.5	3933	1.54

to the best of our knowledge the lost bits based prefetch algorithms are theoretically largely intractable.

VII. SIMULATION RESULTS

In this section we evaluate the proposed prefetch algorithms through simulations using traces of MPEG-4 encoded video. All used traces correspond to videos with a frame rate of 25 frames per second, i.e., the frame period is $\Delta = 40$ msec. The simulation implements the continuous-time model specified in Section II with frame playout deadlines that are discrete integer multiples of the frame period $\Delta = 40$ msec. The scheduling decisions are made on a continuous-time basis, driven by the completion of the individual frame transmissions. That is, a new frame transmission is scheduled to commence as soon as the current frame transmission ends, irrespective of the discrete frame playout deadlines. The peak to mean ratios, standard deviations, and coefficients of variation (standard deviation normalized by mean frame size) of the frame sizes of the video traces used for the simulations are given in Table III. To study the effects of video bit rate variability on the prefetch algorithms we transformed the original video traces to obtain high variance video traces. This transformation increased the frame sizes of large frames and decreased the sizes of small frames while keeping the average bit rate fixed at 64 kbps. Throughout, we refer the original video traces as the low variance video traces, and the transformed video traces as the high variance video traces. To simulate a mix of different video lengths (run times), we generated for each video a set of 21 trace segments consisting of the full-length (108,000 frame) trace, the two halves of the full-length trace, the three thirds, the four quarters, the five fifth, and the six sixth of the full length trace. Each segment was scaled to have an average bit rate of 64 kbit/sec.

For a detailed comparison of the different proposed continuous time prefetch algorithms, we first conduct steady state simulations where all clients start at time zero with empty prefetch buffers and there are always J streams in progress, all of which are collaboratively considered by the evaluated prefetching mechanisms. Each client selects uniformly randomly a video title, one of the trace segments for the selected title, and a uniform random starting phase into the selected trace segment. The entire trace segment from the starting phase frame to the frame preceding the starting phase frame is streamed once (with wrap-around at the end of the trace segment). When a stream terminates, i.e., the last frame of the stream has been displayed at the client, the corresponding client immediately selects a new video stream, i.e., random video title, trace segment, and starting phase and begins transmitting the new stream, whereby the prefetch buffer is initially empty. We estimate the frame loss probabilities and the information (bit) loss probabilities and their 90% confidence intervals for the individual clients after a warm-up period using the method of batch means. All simulations are run until the 90% confidence intervals are less than 10% of the corresponding sample means.

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.

A. *Impact of Prefetch Buffer Size and Traffic Variability on Continuous-Time Prefetch Algorithms*

In this section we examine the effect of client prefetch buffer size B and the effect of video traffic variability on the proposed continuous-time prefetch algorithms. We consider supporting $J = 64$ streams, each with buffer capacity B, over a bottleneck link of capacity $R_s = 64$. Table IV shows the average frame loss probability P_{loss}^f and the average information probability P_{loss}^i achieved by the different algorithms for different prefetch buffer sizes for both low and high variability video streams. We observe that generally the loss probability decreases as the buffer size increases and is smaller for the low variance streams. With a larger buffer, the clients can build up larger prefetched reserves, making starvation less likely. Also, the frame sizes of the low variance are more uniform, making them generally easier to schedule.

Examining more closely the loss probabilities for the different proposed prefetch algorithms, we first observe that for

TABLE IV
AVERAGE LOSS PROBABILITIES OF CONTINUOUS TIME PREFETCHING ALGORITHMS FOR DIFFERENT PREFETCH BUFFERS $B = 32, 64, 128$, AND 256 KByte; $J = 64$ STREAMS IN STEADY-STATE TRANSMISSION OVER AN $R_s = 64$ LINK. (a) AVERAGE FRAME LOSS PROBABILITY; (b) AVERAGE INFORMATION LOSS PROBABILITY

(a)

	Low Variability Video Streams				High Variability Video Streams			
Algorithm	**32 KB**	**64 KB**	**128 KB**	**256 KB**	**32 KB**	**64 KB**	**128 KB**	**256 KB**
TF	0.00160	0.00102	0.00000	0.00000	0.00148	0.00089	0.00000	0.00000
NTB	0.00559	0.00403	0.00221	0.00087	0.00440	0.00352	0.00239	0.00180
RTB	0.00458	0.00328	0.00189	0.00066	0.00452	0.00337	0.00205	0.00079
WNTB	0.00409	0.00286	0.00171	0.00060	0.00428	0.00305	0.00176	0.00065
WRTB	0.00302	0.00194	0.00000	0.00000	0.00326	0.00210	0.00000	0.00000
NLB	0.00485	0.00361	0.00221	0.00087	0.00453	0.00356	0.00229	0.00102
RLB	0.00464	0.00330	0.00190	0.00065	0.00451	0.00332	0.00203	0.00082
WRLB	0.00301	0.00176	0.00000	0.00000	0.00303	0.00182	0.00000	0.00000

(b)

	Low Variability Video Streams				High Variability Video Streams			
Algorithm	**32 KB**	**64 KB**	**128 KB**	**256 KB**	**32 KB**	**64 KB**	**128 KB**	**256 KB**
TF	0.00419	0.00268	0.00000	0.00000	0.00422	0.00270	0.00000	0.00000
NTB	0.00587	0.00437	0.00236	0.00094	0.00445	0.00366	0.00249	0.00182
RTB	0.00498	0.00341	0.00195	0.00066	0.00534	0.00372	0.00221	0.00082
WNTB	0.00463	0.00297	0.00171	0.00069	0.00435	0.00349	0.00176	0.00080
WRTB	0.00265	0.00167	0.00000	0.00000	0.00275	0.00176	0.00000	0.00000
NLB	0.00540	0.00384	0.00231	0.00089	0.00535	0.00425	0.00261	0.00112
RLB	0.00501	0.00347	0.00195	0.00067	0.00530	0.00372	0.00219	0.00087
WRLB	0.00267	0.00152	0.00000	0.00000	0.00251	0.00153	0.00000	0.00000

the frame-based TF algorithm, the frame loss probabilities for the small buffers are significantly smaller than the information loss probabilities. This is because the TF algorithm strives to minimize the frame losses, and in doing so, tends to give preference to transmitting small frames and losing large frames. Indeed, we observe from Table IV that for the TF algorithm, the frame loss probability decreases while the information loss probability slightly increases for the high variability streams compared to the low variability streams. In the high variability streams, there are relatively fewer mid-size frames, while there are more small-sized and large-sized frames. As a consequence the TF algorithm, which is purely focused on numbers of transmitted frames tends to transmit more small frames and lose some more large frames. In contrast to the frame-based TF algorithm, we observe that for the bit-based algorithms, there is relatively little difference between the frame and the information loss probability, with the information loss probabilities tending to be generally slightly higher than the frame loss probabilities. This is because with the bit-based prefetch algorithms, the larger frames have generally the same chance to be scheduled as smaller frames, however, larger frames require more transmission time and therefore tend to be dropped more often when the playout deadline is close.

Overall, we observe that counting the actual number of transmitted (or lost) bits (as done in the RTB, WRTB, RLB, and WRLB policies) gives smaller loss probabilities than relying on the long run average (as done in the NTB, WNTB, and NLB policies). This is because the actual number of transmitted (or lost) bits more closely reflects the current priority level of a stream. Importantly, we observe that taking the number of currently prefetched frames (as done in the WNTB, WRLB, and WRLB policies) into consideration results in significant reductions in loss probability compared to the corresponding policies without the weighing. The number of prefetched frames gives an immediate indication of how close or far—in terms of frame periods—a client is to starvation and thus helps significantly in selecting the client that is most in need of a frame. Combining the counting of the actual number of transmitted (or lost) bits with the weighing by the number of current prefetched frames (as done in the WRTB and WRLB policies) gives the smallest loss probabilities. We also observe that there is generally little difference between considering the transmitted bits or the lost bits.

B. Comparison Between Discrete-Time and Continuous-Time Prefetch Algorithms

In this section we compare three representative discrete-time prefetch algorithms, namely deadline credit (DC) [21], join-the-shortest-queue (JSQ) [28], and bin packing (BP) [27], with the continuous-time prefetch algorithms proposed in this paper. We employ start-up simulations for this comparison since the DC scheme is formulated for the start-up scenario in [21]. In the start-up simulations, each of the J streams starts at time zero with an empty prefetch buffer and uniformly randomly selects a full-length video trace and independently uniformly randomly a starting phase into the selected trace. The full-length trace is then streamed once and the frame and information loss probabilities for each stream are recorded. We run many independent replications of this start-up simulation until the 90% confidence intervals of the loss probabilities are less than 10% of the corresponding means. In each replication, all J streams start with an empty prefetch buffer at time zero and select a random full-length trace and starting phase.

For the DC scheme, we use a slot length of 1/2000th of the frame period, i.e., $\Delta = 2000$ slots. To convert the buffer occupation in bytes to the maximum deadline credit, we use the actual sizes of the frames in the buffer. In the BP scheme, we use the window size of 128 frames, which is a reasonable window size for a prefetch buffer of $B = 64\ \mathrm{KByte}$ [27].

TABLE V

AVERAGE LOSS PROBABILITIES FOR DISCRETE- AND CONTINUOUS-TIME PREFETCH ALGORITHMS FROM START-UP SIMULATIONS FOR DIFFERENT PREFETCH BUFFERS $B = 32, 64, 128$, AND 256 KByte; $J = 64$ STREAMS OVER AN $R_s = 64$ LINK. (a) AVERAGE FRAME LOSS PROBABILITY; (b) AVERAGE INFORMATION LOSS PROBABILITY

(a)

	Low Variability Video Streams				High Variability Video Streams			
Algorithm	**32 KB**	**64 KB**	**128 KB**	**256 KB**	**32 KB**	**64 KB**	**128 KB**	**256 KB**
DC	0.02264	0.02052	0.01911	0.01793	0.02804	0.02547	0.02429	0.02347
BP	0.00386	0.00373	0.00362	0.00360	0.00421	0.00407	0.00395	0.00392
JSQ	0.00725	0.00684	0.00675	0.00668	0.00592	0.00561	0.00551	0.00547
TF	0.00185	0.00120	0.00093	0.00048	0.00170	0.00106	0.00090	0.00055
NTB	0.00651	0.00479	0.00268	0.00115	0.00509	0.00417	0.00289	0.00219
RTB	0.00534	0.00388	0.00226	0.00089	0.00532	0.00402	0.00245	0.00101
WNTB	0.00480	0.00340	0.00202	0.00076	0.00500	0.00359	0.00212	0.00087
WRTB	0.00351	0.00232	0.00108	0.00064	0.00379	0.00253	0.00130	0.00069
NLB	0.00565	0.00429	0.00270	0.00111	0.00533	0.00415	0.00272	0.00129
RLB	0.00538	0.00387	0.00225	0.00086	0.00534	0.00389	0.00246	0.00107
WRLB	0.00352	0.00207	0.00102	0.00060	0.00359	0.00216	0.00113	0.00060

(b)

	Low Variability Video Streams				High Variability Video Streams			
Algorithm	**32 KB**	**64 KB**	**128 KB**	**256 KB**	**32 KB**	**64 KB**	**128 KB**	**256 KB**
DC	0.02733	0.02485	0.02311	0.02178	0.03945	0.03585	0.03423	0.03308
BP	0.01042	0.00999	0.00972	0.00965	0.01521	0.01489	0.01453	0.01447
JSQ	0.00954	0.00900	0.00887	0.00873	0.00949	0.00896	0.00882	0.00869
TF	0.00489	0.00312	0.00213	0.00154	0.00492	0.00315	0.00220	0.00142
NTB	0.00686	0.00515	0.00286	0.00123	0.00514	0.00432	0.00304	0.00220
RTB	0.00584	0.00402	0.00232	0.00090	0.00631	0.00446	0.00265	0.00109
WNTB	0.00546	0.00352	0.00204	0.00086	0.00516	0.00409	0.00215	0.00103
WRTB	0.00317	0.00198	0.00184	0.00149	0.00325	0.00210	0.00189	0.00120
NLB	0.00626	0.00458	0.00274	0.00115	0.00629	0.00502	0.00309	0.00142
RLB	0.00588	0.00408	0.00230	0.00089	0.00627	0.00433	0.00267	0.00114
WRLB	0.00309	0.00181	0.00165	0.00120	0.00296	0.00183	0.00163	0.00113

1) Impact of Buffer Size and Traffic Variability: In Table V we report the average loss probabilities for the discrete-time and continuous-time prefetch algorithms for different prefetch buffer sizes for both the low and high variability streams. In Fig. 7 we plot the frame and information loss probabilities for the individual clients for a prefetch buffer of $B = 64$ KByte. We first note that for this start-up scenario, the comparison tendencies among the different continuous-time prefetch algorithms are generally unchanged from the steady-state scenario considered in Section VII-A.

Turning to the comparison of continuous- and discrete-time prefetch algorithms, we observe that for both the low and high variability streams, the discrete-time prefetch algorithms give significantly larger frame and information loss probabilities than the continuous-time algorithms. Typically, the continuous-time prefetch algorithms reduce the starvation probabilities by a factor of four or more compared to the discrete-time algorithms. This improvement is primarily due to utilizing the full bandwidth by scheduling video frames across frame periods with the continuous-time prefetch algorithms. The frame loss probabilities achieved by BP for small prefetch buffers are an exception to this general trend in that BP achieves frame loss probabilities as low as the continuous-time algorithms, but BP has information loss probabilities much larger than the continuous-time algorithms. This behavior is caused by BP's strategy to look for small frames by temporarily skipping large frames that do not fit into the remaining transmission capacity in a frame period and thus prefetching more future small frames. In contrast, the JSQ algorithm does not skip frames, but rather scans the next frame to be transmitted for all ongoing streams to find frames that fit into the remaining transmission capacity in a frame period. As observed from Table V, this JSQ strategy results in larger frame loss probabilities, but smaller information loss probabilities compared to BP.

Regarding fairness, we observe from Fig. 7 for both low and high variability streams that the discrete-time prefetching algorithms generally provide the individual clients with approximately equal frame loss probabilities, but that the information loss probabilities can vary significantly between clients. Indeed, the discrete-time prefetch algorithms have primarily been designed to be fair in terms of the frame loss probability, while the information loss probability has been largely ignored in the development of the discrete-time algorithms. In contrast, the continuous-time prefetch algorithms can provide fairness in terms of the frame or information loss probability corresponding to the adopted priority function. More specifically, the frame-based TF prefetch policy meets the very tight fairness criterion of Lemma 1, and gives consequently essentially identical frame loss probabilities, as observed in Fig. 7. The individual information loss probabilities achieved with the TF algorithm vary only slightly. Also, the bit-based prefetch algorithms give good fairness both in terms of frame and information loss probability. Overall, the continuous-time prefetch algorithms avoid the pronounced differences between the frame loss probability and the information loss probability behaviors observed for BP by not selectively prefetching frames with specific properties (such as

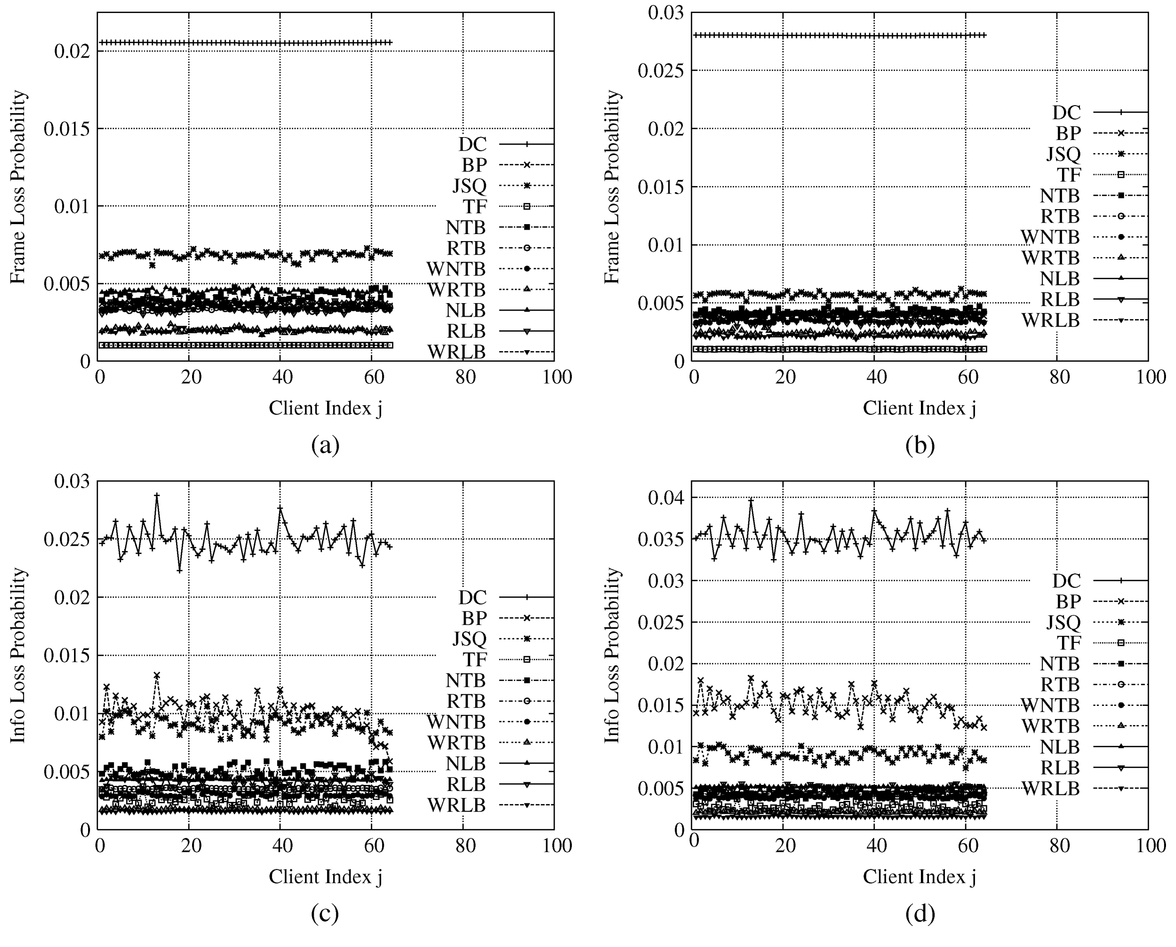


Fig. 7. Loss probabilities for individual clients for $J = 64$ streams over a link supporting $R_s = 64$ streams with $B = 64$ KByte buffer in each client. (a) Frame loss probability for low variability streams; (b) frame loss probability for high variability streams; (c) information loss probability for low variability streams; (d) information loss probability for high variability streams.

TABLE VI
AVERAGE LOSS PROBABILITIES FOR $J = 64$ LOW VARIABILITY STREAMS OVER LINK WITH DIFFERENT NORMALIZED CAPACITY R_s; START-UP SIMULATION WITH FIXED $B = 64$ KByte CLIENT BUFFERS

	Frame Loss Probability					Information Loss Probability				
Alg.	$R_s = 32$	$R_s = 63$	$R_s = 63.5$	$R_s = 64$	$R_s = 64.5$	$R_s = 32$	$R_s = 63$	$R_s = 63.5$	$R_s = 64$	$R_s = 64.5$
DC	0.5257	0.03548	0.02932	0.02052	0.02005	0.5509	0.04303	0.03383	0.02485	0.02063
BP	0.5063	0.02184	0.01268	0.00373	0.00347	0.53482	0.03009	0.01875	0.00999	0.00537
JSQ	0.5099	0.02364	0.01450	0.00684	0.00368	0.5338	0.02795	0.01724	0.00900	0.00466
TF	0.5034	0.02160	0.01074	0.00120	0.00082	0.5274	0.02292	0.01223	0.00312	0.00152
NTB	0.5075	0.02250	0.01291	0.00479	0.00247	0.5296	0.02444	0.01346	0.00515	0.00298
RTB	0.5065	0.02213	0.01225	0.00388	0.00216	0.5284	0.02244	0.01286	0.00402	0.00245
WNTB	0.5059	0.02040	0.01132	0.00340	0.00127	0.5278	0.02158	0.01194	0.00352	0.00165
WRTB	0.5047	0.02091	0.01079	0.00232	0.00095	0.5261	0.02046	0.01055	0.00198	0.00096
NLB	0.5070	0.02170	0.01285	0.00429	0.00260	0.5290	0.02307	0.01319	0.00458	0.00288
RLB	0.5065	0.02137	0.01223	0.00387	0.00216	0.5284	0.02352	0.01249	0.00408	0.00240
WRLB	0.5044	0.02085	0.01038	0.00207	0.00121	0.5260	0.02059	0.01009	0.00181	0.00093

small number of bits). Instead, the continuous-time prefetch algorithms transmit the frames of a given stream in order of their playout deadlines.

2) Impact of Utilization Level and System Size: In this section we analyze the impact of the utilization level, i.e., the ratio of number of supported streams J to normalized link capacity R_s, and the system size, i.e., the number of multiplexed streams J for a fixed utilization level, on frame and information loss. We vary the utilization level by fixing $J = 64$ and considering a range of normalized link capacities R_s.

Considering first the overload scenarios with $J > R_s$ in Table VI, we observe that the continuous-time prefetching algorithms achieve information loss probabilities closer to the theoretical minimum of $(J - R_s)/J$ than the discrete-time algorithms. Consistent with our earlier observations, the WRTB and WRLB algorithms achieve information loss probabilities closest to the theoretical minimum. We also observe that even slight

TABLE VII
AVERAGE LOSS PROBABILITIES FOR STREAMING $J = R_s$ LOW VARIABILITY STREAMS TO $B = 64$ KByte CLIENTS

	Frame Loss Probability			Information Loss Probability		
Algorithm	$J = Rs = 32$	$J = Rs = 64$	$J = R_s = 128$	$J = R_s = 32$	$J = R_s = 64$	$J = R_s = 128$
DC	0.02061	0.02052	0.02049	0.02498	0.02485	0.02481
BP	0.00375	0.00373	0.00371	0.01010	0.00999	0.00994
JSQ	0.00690	0.00684	0.00675	0.00905	0.00900	0.00888
TF	0.00122	0.00120	0.00115	0.00316	0.00312	0.00298
NTB	0.00483	0.00479	0.00475	0.00530	0.00515	0.00510
RTB	0.00392	0.00388	0.00379	0.00408	0.00402	0.00392
WNTB	0.00348	0.00340	0.00333	0.00361	0.00352	0.00344
WRTB	0.00241	0.00232	0.00227	0.00202	0.00198	0.00194
NLB	0.00439	0.00429	0.00421	0.00468	0.00458	0.00450
RLB	0.00395	0.00387	0.00382	0.00417	0.00408	0.00403
WRLB	0.00216	0.00207	0.00206	0.00185	0.00181	0.00180

TABLE VIII
AVERAGE LOSS PROBABILITIES FOR 32 LOW VARIABILITY STREAMS AND 32 HIGH VARIABILITY STREAMS OVER AN $R_s = 64$ LINK

	Frame Loss Probability		Information Loss Probability	
Algorithm	**low variab. str.**	**high variab. str.**	**low variab. str.**	**high variab. str.**
DC	0.0272	0.0275	0.0331	0.0336
BP	0.0033	0.0037	0.0086	0.0104
JSQ	0.0071	0.0067	0.0092	0.0091
TF	0.0009	0.0009	0.0026	0.0033
NTB	0.0050	0.0048	0.0060	0.0059
RTB	0.0040	0.0042	0.0041	0.0041
WNTB	0.0036	0.0038	0.0035	0.0036
WRTB	0.0023	0.0025	0.0021	0.0022
NLB	0.0043	0.0051	0.0045	0.0055
RLB	0.0040	0.0048	0.0043	0.0046
WRLB	0.0021	0.0020	0.0019	0.0020

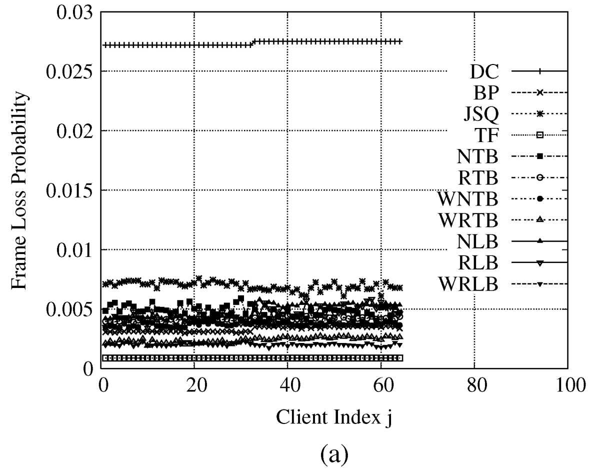
overload conditions with $R_s = 63.5$ result in fairly large increases in the loss probabilities from the $R_s = J = 64$ stability limit scenario. The loss probabilities of the continuous-time algorithms increase by factors of around three to five, whereas the discrete time algorithms, which have already larger loss probabilities at the $R_s = J$ stability limit, experience relatively smaller increases of the information loss probabilities by factors less than two. Further increasing the utilization by reducing the link capacity to $R_s = 63$ results roughly in a doubling of the loss probabilities compared to the $R_s = 63.5$ case. For the extreme overload case of $J/R_s = 2$, we observe information loss probabilities close to the theoretical minimum of 1/2 and correspondingly large frame loss probabilities.

Turning to the reduction in the utilization level by increasing R_s from 64 to 64.5, we observe that all algorithms, except DC and BP, achieve reductions in the loss probabilities by roughly a factor of two. The BP algorithm achieves a reduction by a factor of close to two for the information loss probability; but the frame loss probability, which is already very small for $R_s = 64$, is only very slightly further reduced. The DC algorithm can extract only small reductions of the loss probabilities for the lower utilization. Overall, we conclude that the proposed continuous-time prefetch algorithms achieve information loss probabilities close to the theoretical minimum for overload conditions, and provide significant reductions in the loss probability for even slight reductions on the utilization below the stability limit.

From Table VII we observe that the system size has a relatively minor impact on the performance of the prefetching algorithms. Larger systems that multiplex more streams J for the same utilization level J/R_s have somewhat increased opportunities for statistical multiplexing and therefore achieve very slightly smaller loss probabilities.

3) Impact of Heterogeneous Streams: In this section we examine the impact of heterogenous streams on the performance of the prefetching algorithms. We first consider concurrently streaming 32 low variability streams and 32 high variability streams over an $R_s = 64$ link to clients with a $B = 64$ KByte prefetch buffer using start-up simulations. We report the mean frame and information loss probabilities experienced by the group of 32 clients receiving low variability streams and the group of 32 clients receiving high variability streams in Table VIII.

We observe from the table that generally the prefetch algorithms provide relatively good fairness to the two groups of video clients with both groups experiencing roughly equivalent frame and information loss probabilities. In a few instances, the high variability clients experience slightly higher loss probabilities, reflecting that the higher variability traffic is more challenging to schedule. The exception to this relatively good fairness performance for both loss probability measures is the TF algorithm, which gives both groups of clients equivalent frame loss probabilities, but significantly larger information loss probabilities to the high variability clients. This behavior is due to the frame-based nature of the TF algorithm, which enforces the same frame loss probability for all clients, but does not consider the sizes of the video frames. The high variability streams contain relatively more large-sized frames, which are more difficult



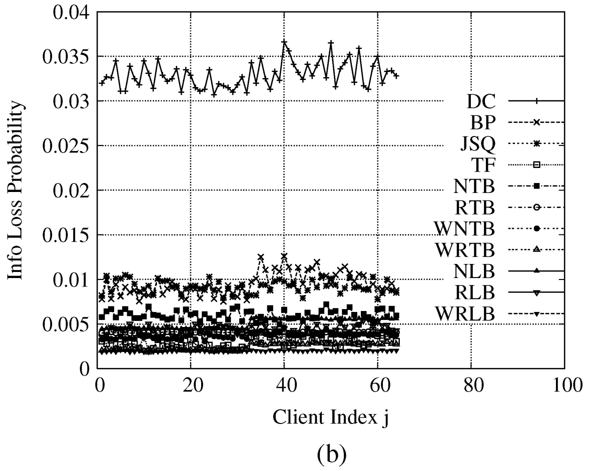


Fig. 8. Loss probabilities for individual clients with clients 1 through 32 receiving low variability streams and clients 33 through 64 receiving high variability streams over a link supporting $R_s = 64$ streams with $B = 64$ KByte buffer in each client. (a) Frame loss probability. (b) Information loss probability.

TABLE IX
AVERAGE LOSS PROBABILITIES FOR 32 STREAMS WITH 64 kbps AVERAGE BIT RATE AND 16 STREAMS WITH AND 128 kbps AVERAGE BIT RATE OVER AN $R_s = 64$ LINK

	Frame Loss Probability		Information Loss Probability	
Algorithm	**64 kbps str.**	**128 kbps str.**	**64 kbps str.**	**128 kbps str.**
DC	0.0275	0.0275	0.0323	0.0362
BP	0.0039	0.00412	0.0094	0.0142
JSQ	0.0069	0.0071	0.0091	0.0094
TF	0.0019	0.0019	0.0053	0.0062
NTB	0.0051	0.0049	0.0052	0.0059
RTB	0.0039	0.0038	0.0040	0.0040
WNTB	0.0034	0.0045	0.0035	0.0047
WRTB	0.0023	0.0023	0.0021	0.0022
NLB	0.0044	0.0042	0.0046	0.0043
RLB	0.0038	0.0033	0.0041	0.0041
WRLB	0.0021	0.0032	0.0018	0.0017

to schedule, resulting in larger information loss for the high variability stream group.

For further insight into the loss probabilities experienced by the individual clients we plot in Fig. 8 the frame and information loss probabilities for the individual clients. We observe that the algorithms that provide good fairness among the groups also provide good fairness among the individual clients within a given group. Especially the continuous-time prefetch algorithms counting the actual numbers of bits and incorporating weighing according to the number of prefetched frames (done in WRTB and WRLB) achieve uniform information loss probabilities across the individual clients.

In a second heterogenous streaming scenario we consider 32 clients that receive video with an average bit rate of 64 kbps and 16 clients that receive video with an average bit rate of 128 kbps. Note that the total average bit rate is equivalent to 64 clients concurrently receiving 64 kbps steams. We set $R_s = 64$ as in the preceding start-up simulations. Table IX shows the average frame and information loss probabilities for the two groups of clients. We observe that the frame-based algorithms provide similar frame loss probabilities to the two groups, with the DC and TF algorithms providing particularly strict fairness. On the other hand, the bit-based algorithms provide similar information loss probabilities to the two groups. The algorithms incorporating the long run average rates show slight differences, while the algorithms considering the actual number of transmitted bits demonstrate very good fairness performance.

Lastly, comparing the loss probabilities for the heterogenous scenarios in Tables VIII and IX with the corresponding loss probabilities for the homogeneous scenario, e.g., the low variability streams with $B = 64$ KByte scenario in Table V, we observe that the DC algorithm gives overall somewhat higher loss probabilities for the heterogeneous scenarios. The other algorithms give overall roughly equivalent loss probability levels for the homogeneous and heterogeneous scenarios.

VIII. CONCLUSION

We have developed and evaluated continuous-time algorithms for the collaborative prefetching of continuous media with discrete playout deadlines, such as encoded video. The continuous-time algorithms allow for the transmission of video frames across frame period boundaries. In contrast, previously studied collaborative prefetching schemes typically scheduled video frames for transmission during a given frame period and did not permit transmissions across the frame period boundaries. This led to inefficiencies in the form of unused transmission capacity at the end of a frame period, when no frame could fit into the remaining capacity. The continuous-time prefetching algorithms developed in this paper overcome these inefficiencies and typically reduce the starvation probabilities by a factor of four or more. We have formally analyzed the fairness and efficiency characteristics of the proposed continuous-time prefetching algorithms.

There are many exciting avenues for future work on continuous-time collaborative prefetching. One direction is to consider the streaming from several distributed servers over a common bottleneck to clients. In the present study the video was streamed from a single server and all the scheduling decisions were made at that single server. In a multi-server scenario, which could arise when streaming in parallel from several peers in a peer-to-peer network, the different servers would need coordinate their transmissions so as to achieve overall low starvation probabilities and good fairness. Another direction is to explore the continuous-time prefetching over wireless links. In the present study, the bottleneck link was reliable, i.e., a video frame scheduled on the link was guaranteed to arrive at the client. In contrast, wireless links are unreliable, i.e., a transmitted frame may be dropped on the wireless link, and possibly require retransmission.

References

[1] F. H. Fitzek and M. 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] C. Bewick, R. Pereira, and M. 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.

[3] H.-C. 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.

[4] W.-C. Feng and J. Rexford, "Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video," *IEEE Trans. Multimedia*, vol. 1, no. 3, pp. 302–313, Sept. 1999.

[5] T. Gan, K.-K. Ma, and L. Zhang, "Dual-plan bandwidth smoothing for layer-encoded video," *IEEE Trans. Multimedia*, vol. 7, no. 2, pp. 379–392, Apr. 2005.

[6] M. Grossglauser, S. Keshav, and D. Tse, "RCBR: A simple and efficient service for multiple time-scale traffic," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 741–755, Dec. 1997.

[7] Z. Gu and K. 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] C.-D. Iskander and R. T. Mathiopoulos, "Online smoothing of VBR H.263 video for the CDMA2000 and IS-95B uplinks," *IEEE Trans. Multimedia*, vol. 6, no. 4, pp. 647–658, Aug. 2004.

[9] M. Krunz, W. Zhao, and I. 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.

[10] M. Krunz, "Bandwidth allocation strategies for transporting variable-bit-rate video traffic," *IEEE Communications Magazine*, vol. 37, no. 1, pp. 40–46, Jan. 1999.

[11] H. Lai, J. Y. Lee, and L.-K. Chen, "A monotonic-decreasing rate scheduler for variable-bit-rate video streaming," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, no. 2, pp. 221–231, Feb. 2005.

[12] S. S. Lam, S. Chow, and D. K. Y. Yau, "A lossless smoothing algorithm for compressed video," *IEEE/ACM Trans. Networking*, vol. 4, no. 5, pp. 697–708, Oct. 1996.

[13] R. Sabat and C. 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.

[14] J. Salehi, Z.-L. Zhang, J. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, pp. 397–410, Aug. 1998.

[15] A. Solleti and K. J. Christensen, "Efficient transmission of stored video for improved management of network bandwidth," *International Journal of Network Management*, vol. 10, pp. 277–288, 2000.

[16] B. Vandalore, W.-C. Feng, R. Jain, and S. Fahmy, "A survey of application layer techniques for adaptive streaming of multimedia," *Real-Time Imaging Journal*, vol. 7, no. 3, pp. 221–235, 2001.

[17] D. Ye, J. C. Barker, Z. Xiong, and W. Zhu, "Wavelet-based VBR video traffic smoothing," *IEEE Trans. Multimedia*, vol. 6, no. 4, pp. 611–623, Aug. 2004.

[18] D. Ye, Z. Xiong, H.-R. Shao, Q. Wu, and W. Zhu, "Wavelet-based smoothing and multiplexing of VBR video traffic," in *Proceedings of IEEE Globecom*, San Antonio, TX, Nov. 2001, pp. 2060–2064.

[19] Z.-L. Zhang, J. Kurose, J. Salehi, and D. 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.

[20] 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.

[21] Z. Antoniou and I. Stavrakakis, "An efficient deadline-credit-based transport scheme for prerecorded semisoft continuous media applications," *IEEE/ACM Trans. Networking*, vol. 10, no. 5, pp. 630–643, Oct. 2002.

[22] S. Bakiras and V. O. Li, "Maximizing the number of users in an interactive video-on-demand system," *IEEE Trans. Broadcasting*, vol. 48, no. 4, pp. 281–292, Dec. 2002.

[23] J. C. H. Yuen, E. Chan, and K.-Y. Lam, "Real time video frames allocation in mobile networks using cooperative pre-fetching," *Multimedia Tools and Applications*, vol. 32, no. 3, pp. 329–352, Mar. 2007.

[24] Y.-W. Leung and T. K. C. Chan, "Design of an interactive video-on-demand system," *IEEE Trans. Multimedia*, vol. 5, no. 1, pp. 130–140, Mar. 2003.

[25] F. Li and I. 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.

[26] C.-S. Lin, M.-Y. Wu, and W. 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, pp. 1461–1464.

[27] S. Oh, Y. Huh, B. Kulapala, G. Konjevod, A. W. Richa, and M. Reisslein, "A modular algorithm-theoretic framework for the fair and efficient collaborative prefetching of continuous media," *IEEE Trans. Broadcasting*, vol. 51, no. 2, pp. 200–215, June 2005.

[28] M. Reisslein and K. W. Ross, "High-performance prefetching protocols for VBR prerecorded video," *IEEE Network*, vol. 12, no. 6, pp. 46–55, Nov/Dec 1998.

[29] H. Zhu and G. Cao, "A power-aware and QoS-aware service model on wireless networks," in *Proceedings of IEEE Infocom*, Hong Kong, Hong Kong, Mar. 2004, pp. 1393–1403.

[30] M. Krunz and S. 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.

[31] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *Proceedings of the 32nd Annual Symposium on Foundation of Computer Science*, 1991, pp. 100–110.

[32] T.-W. Lam and K.-K. To, "Performance guarantee for online deadline scheduling in the presence of overload," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001, pp. 755–764.

[33] L. Sha, J. P. Lehoczky, and R. Rajkuma, "Solutions for some practical problems in prioritized preemptive scheduling," in *Proceedings of the Real-Time Systems Symposium*, 1986, pp. 181–191.

[34] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings of the IFIP Congress*, 1974, pp. 807–813.

Soohyun Oh is a lecturer in the Department of Computer Engineering at Hansung University, Seoul, Korea. She received M.S. and Ph.D. degrees in Computer Science from Arizona State University, in 2001 and 2005, respectively. From July 2006 through February 2007 she joined Mobile Computing Laboratory at Sungkyunkwan University as a researcher. Her research interests are continuous media streaming, QoS, and packet routing.

Beshan Kulapala received the B.S. degree in electrical engineering from the University of Kentucky, Lexington, in 2001, and received the M.S. degree in electrical engineering from Arizona State University, Tempe, in 2003. Since 2003 he has been a Ph.D. student in the Department of Electrical Engineering at Arizona State University. His research interests are in the area of video transmission over wired and wireless networks. He is a student member of the IEEE.

Andréa W. Richa is an Associate Professor at the Department of Computer Science and Engineering at Arizona State University, Tempe, since August 2004. She joined this department as an Assistant Professor in August 1998. Prof. Richa received her M.S. and Ph.D. degrees from the School of Computer Science at Carnegie Mellon University, in 1995 and 1998, respectively. She also earned an M.S. degree in Computer Systems from the Graduate School in Engineering (COPPE), and a B.S. degree in Computer Science, both at the Federal University of Rio de Janeiro, Brazil, in 1992 and 1990, respectively. Prof. Richa's main area of research is in network algorithms. Some of the topics Dr. Richa has worked on include packet scheduling, distributed load balancing, packet routing, mobile network clustering and routing protocols, and distributed data tracking. Prof. Richa's data tracking (or name lookup) algorithm has been widely recognized as the first benchmark algorithm for the development of distributed databases in peer-to-peer networking, having being references by over 130 academic journal or conference publications to date, and being implemented as part of two of the current leading projects in peer-to-peer networking. Dr. Richa's was the recipient of an NSF CAREER Award in 1999. For a selected list of her publications, CV, and current research projects, please visit http://www.public.asu.edu/aricha.

Martin Reisslein is an Associate Professor in the Department of Electrical Engineering at Arizona State University (ASU), Tempe. He received the Dipl.-Ing. (FH) degree from the Fachhochschule Dieburg, Germany, in 1994, and the M.S.E. degree from the University of Pennsylvania, Philadelphia, in 1996. Both in electrical engineering. He received his Ph.D. in systems engineering from the University of Pennsylvania in 1998. During the academic year 1994–1995 he visited the University of Pennsylvania as a Fulbright scholar. From July 1998 through October 2000 he was a scientist with the German National Research Center for Information Technology (GMD FOKUS), Berlin and lecturer at the Technical University Berlin. From October 2000 through August 2005 he was an Assistant Professor at ASU. He served as editor-in-chief of the *IEEE Communications Surveys and Tutorials* from January 2003 through February 2007 and has served on the Technical Program Committees of *IEEE Infocom*, *IEEE Globecom*, and the *IEEE International Symposium on Computer and Communications*. He has organized sessions at the *IEEE Computer Communications Workshop (CCW)*. He maintains an extensive library of video traces for network performance evaluation, including frame size traces of MPEG–4 and H.264 encoded video, at http://trace.eas.asu.edu. His research interests are in the areas of Internet Quality of Service, video traffic characterization, wireless networking, optical networking, and engineering education. For a selected list of his publications, CV, and current research projects, please visit http://www.fulton.asu.edu/~mre.