Technical University Berlin

Telecommunication Networks Group

A Prefetching Protocol for Continuous Media Streaming in Wireless Environments

Frank H.P. Fitzek and Martin Reisslein

fitzek@ee.tu-berlin.de, reisslein@fokus.gmd.de

Berlin, Juli 2000

TKN Technical Report TKN-00-05

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

A Prefetching Protocol for Continuous Media Streaming in Wireless Environments

Frank H.P. Fitzek
TU Berlin
Einsteinufer 25
10585 Berlin, Germany
fitzek@ee.tu-berlin.de
www-tkn.ee.tu-berlin.de/~fitzek

Martin Reisslein
GMD FOKUS
Kaiserin–Augusta–Allee 31
10589 Berlin, Germany
reisslein@fokus.gmd.de
www.fokus.gmd.de/usr/reisslein

Abstract

Streaming of continuous media over wireless links is a notoriously difficult problem. This is due to the stringent Quality of Service requirements of continuous media and the unreliability of wireless links. We develop a streaming protocol for the real–time delivery of gprerecorded continuous media from a central base station to multiple mobile clients within a wireless cell. Our protocol prefetches parts of the ongoing continuous media streams into prefetch buffers in the clients. Our protocol prefetches according to a Join–the–Shortest–Queue policy. By exploiting rate adaptation techniques of wireless data packet protocols, the Join–the–Shortest–Queue policy dynamically allocates more transmission capacity to clients with small prefetched reserves. Our protocol uses channel probing to handle the location–dependent, time–varying, and bursty errors of wireless links. We evaluate our prefetching protocol through extensive simulations with VBR MPEG encoded video traces. Our simulations indicate that for bursty VBR video with an average rate of 64 kbit/sec and typical wireless communication conditions our prefetching protocol achieves client starvation probabilities on the order of 10^{-4} and a bandwidth efficiency of 90 % with client buffers of 128 kBytes.

Keywords

CDMA, Channel Probing, Multimedia, Prefetching, Prerecorded Continuous Media, Rate Adaptation, Real–Time Streaming, Wireless Communication.

I. INTRODUCTION

Due to the popularity of the World Wide Web retrievals from web servers are dominating today's Internet. While most of the retrieved objects today are textual and image objects, web–based streaming of continuous media, such as video and audio, becomes increasingly popular. It is expected that by 2003, continuous media will account for more than 50 % of the data available on the web servers [1]. This trend is reflected in a recent study [2], which found that the number of continuous media objects stored on web servers has tripled in the first nine months of 1998. At the same time there is increasingly the trend towards accessing the Internet and Web from wireless mobile devices. Analysts predict that there will be over one billion mobile phone users by 2003 and more people will access the Internet from wireless than wireline devices [3].

The stringent Quality of Service (QoS) requirements of continuous media and the unreliability of wireless links combine to make streaming over wireless links a notoriously difficult problem. In this paper we develop a high performance streaming protocol for the real–time delivery of prerecorded continuous media over wireless links. We consider the streaming from a central base station to multiple mobile clients within a wireless cell. We note that to our knowledge our study is the *first to develop and evaluate a protocol for the streaming of* prerecorded *continuous media over wireless links.* Our protocol allows for immediate commencement of playback as well as near instantaneous client interactions, such as pause/resume and temporal jumps. Our protocol gives a constant perceptual media quality at the clients while achieving a very high bandwidth efficiency. Our protocol achieves this high performance by exploiting two special properties of *prerecorded* continuous media: (1) the client consumption rates over the duration of the playback are known before the streaming commences, and (2) while the continuous media stream is being played out at the client, parts of the stream can be prefetched into the client's memory. The prefetched reserves allow the clients to continue playback during periods of adverse transmission conditions on the wireless links. In addition, the prefetched reserves allow the clients to maintain a high perceptual media quality when retrieving bursty Variable Bit Rate (VBR) encoded streams.

The prerecorded continuous media streams are prefetched according to a specific Join–the–Shortest–Queue (JSQ) policy, which strives to balance the prefetched reserves in the mobile clients within a wireless cell. The JSQ prefetch policy exploits rate adaptation techniques of wireless data packet protocols [4]. The rate adaptation techniques allow for the dynamic allocation of transmission capacities to the ongoing wireless connections. In the Code Division Multiple Access (CDMA) IS–95 (Revision B) standard, for instance, the rate adaptation is achieved by varying the number of codes (i.e., the number of parallel channels) used for the transmissions to the individual clients. (The total number of code channels used for continuous media streaming in the wireless cell is typically constant.) Roughly speaking, the JSQ prefetch policy dynamically allocates more transmission capacity to mobile clients with small prefetched reserves while allocating less transmission capacity to the clients with large reserves. The ongoing streams within a wireless cell collaborate through this lending and borrowing of transmission capacities. Channel probing is used to judiciously utilize the transmission capacities of the wireless links, which typically experience location–dependent, time–varying, and bursty errors. Our extensive numerical studies indicate that this collaboration is highly effective in reducing playback

starvation at the clients while achieving a high bandwidth efficiency. For bursty VBR video with an average rate of 64 kbit/sec and typical wireless communication conditions our prefetching protocol achieves client starvation probabilities on the order of 10^{-4} and a bandwidth efficiency of 90 % with client buffers of 128 kBytes. Introducing a start–up latency of 0.4 sec reduces the client starvation probability to roughly 10^{-5}.

This paper is organized as follows. In Section II we describe our wireless streaming architecture. In Section III we develop the JSQ prefetching protocol. We outline how to employ our prefetching protocol in third generation CDMA systems and TDMA systems. We also describe our simulation set–up. In Section IV we introduce channel probing; a mechanism that handles the location–dependent, time–varying, and bursty errors of wireless environments. We also evaluate our prefetching protocol through extensive simulations. In Section V we discuss how our prefetching protocol allows for client interactions, such as pause/resume and temporal jumps, with minimal delays; numerical results for client interactions are presented. We discuss the related work in Section VI and conclude in Section VII.

II. ARCHITECTURE

Figure 1 illustrates our architecture for continuous media streaming in wireless environments. A central base station provides streaming services to multiple wireless (and possibly mobile) clients within a wireless cell. Let J denote the number of clients serviced by the base station. We assume for the purpose of this study that each client receives one stream; thus there are J streams in process. (Although some clients might receive the same stream, the phases (i.e., starting times) are typically different.) The basic principle of our streaming protocol — exploiting rate adaptation techniques for prefetching — can be applied to any type of wireless communication system with a slotted Time Division Duplex (TDD) structure. The TDD structure provides alternating forward (base station to clients) and backward (clients to base station) transmission slots.

To fix ideas we initially consider a Code Division Multiple Access (CDMA) system that adapts rates in the forward direction by aggregating orthogonal code channels, that is, by varying the number of code channels used for transmissions to individual clients. The second generation CDMA IS–95 (Rev. B) system [5] is an example of such a system; as is the third generation UMTS system in TDD mode. (We consider continuous media streaming in third generation CDMA systems and TDMA systems in Section III-C.) Let S denote the number of orthogonal codes used by the base station for transmitting the continuous media streams to the clients. Let $R(j),\ j = 1, \ldots, J$, denote the number of parallel

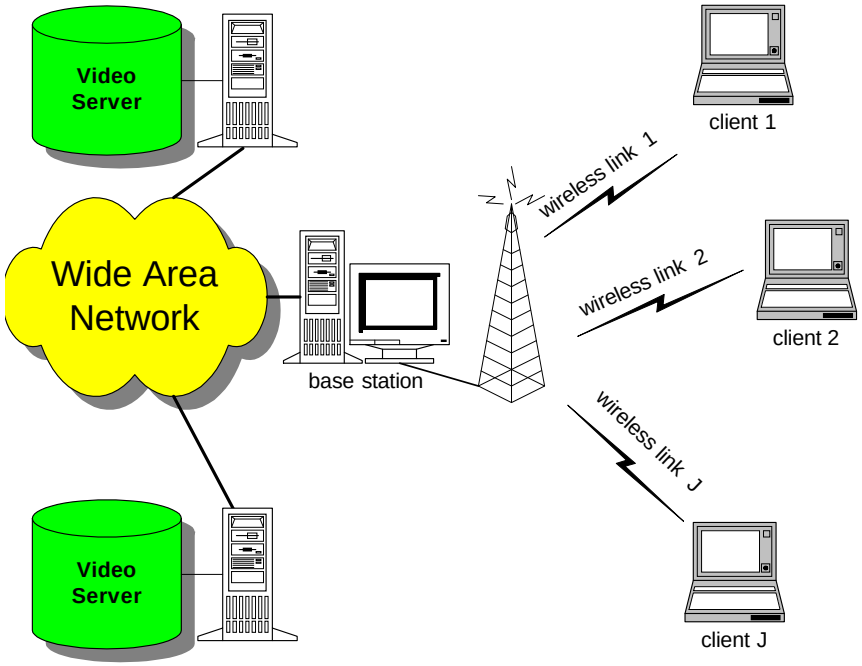

Fig. 1. Architecture: A central base station streams prerecorded continuous media to wireless (and possibly mobile) clients within a wireless cell.

channels supported by the radio front–end of client j. The CDMA IS–95 (Rev. B) standard provides up to eight parallel channels per client; in the TDD mode of UMTS up to 15 parallel code channels can be assigned to an individual client. Let C denote the data rate (in bit/sec) provided by one CDMA code channel in the forward direction.

Our streaming protocol is suitable for any type of prerecorded continuous media. To fix ideas we focus on prerecorded video streams in this paper. A key feature of our protocol is that it accommodates any type of encoding; it accommodates Constant Bit Rate (CBR) and bursty Variable Bit Rate (VBR) encodings as well as encodings with a fixed frame rate (such as MPEG) and a variable frame rate (such as H.263). For the transmission over the wireless links the video frames are packetized into fixed length packets. The packet size is set such that one CDMA code channel accommodates exactly one packet in one forward slot; thus the base station can transmit S packets on the orthogonal code channels in a forward slot.

Let $N(j),\ j = 1, \ldots, J$, denote the length of the video streams in frames. Let $x_n(j)$ denote the number of packets in the nth frame of video stream j. Note that for a CBR encoded video stream j the $x_n(j)$'s are identical. Let $t_n(j)$ denote the interarrival time between the nth frame and the $(n+1)$th

frame of video stream j in seconds. Frame n is displayed for a frame period of $t_n(j)$ seconds on the client's screen. For a constant frame rate encoding the frame periods $t_n(j)$ are identical. Because the video streams are prerecorded the sequence of integers $(x_1(j),\ x_2(j), \ldots, x_{N(j)}(j))$ and the sequence of real numbers $(t_1(j),\ t_2(j), \ldots, t_{N(j)}(j))$ are fully known when the streaming commences.

When a client requests a specific video the base station relays the request to the appropriate origin server or proxy server. If the request passes the admission tests the origin/proxy server immediately begins to stream the video via the base station to the client. Our focus in this paper is on the streaming from the base station over the wireless link to the client. The streaming from the origin/proxy server to the base station is beyond the scope of this paper. We assume for the purpose of this study that the video is delivered to the base station in a timely fashion. Upon granting the client's request the base station immediately commences streaming the video to the client. The packets arriving at the client are placed in the client's prefetch buffer. The video is displayed on the client's monitor as soon as a few frames have arrived at the client. Under normal circumstances the client displays frame n of video stream j for $t_n(j)$ seconds, then removes frame $n+1$ from its prefetch buffer, decodes it, and displays it for $t_{n+1}(j)$ seconds. If at one of these epochs there is no complete frame in the prefetch buffer the client suffers playback starvation and loses the current frame. The client will try to conceal the missing encoding information by applying error concealment techniques [6]. At the subsequent epoch the client will attempt to display the next frame of the video.

In our protocol the base station keeps track of the contents of the prefetch buffers in the clients. Towards this end, let $b(j),\ j = 1, \ldots, J$, denote the number of packets in the prefetch buffer of client j. Furthermore, let $p(j),\ j = 1, \ldots, J$, denote the length of the prefetched video segment in the prefetch buffer of client j in seconds. The counters $b(j)$ and $p(j)$ are updated (1) when the client j acknowledges the reception of sent packets, and (2) when a frame is removed, decoded, and displayed at client j.

First, consider the update when packets are acknowledged. For the sake of illustration suppose that the $x_n(j)$ packets of frame n of stream j have been sent to client j during the just expired forward slot. Suppose that all $x_n(j)$ packets are acknowledged during the subsequent backward slot. When the last of the $x_n(j)$ acknowledgments arrives at the base station, the counters are updated by setting $b(j) \leftarrow b(j) + x_n(j)$, and $p(j) \leftarrow p(j) + t_n(j)$.

Next, consider the update of the counters when a frame is removed from the prefetch buffer, decoded,

and displayed at the client. Given the sequence $t_n(j),\ n = 1, \ldots, N$, and the starting time of the video playback at the client the base station keeps track of the removal of frames from the prefetch buffer of client j. Suppose that at a particular instant frame $\theta(j)$ is to be removed from the prefetch buffer of client j. The base station tracks the prefetch buffer contents by updating $b(j) \leftarrow [b(j) - x_{\theta(j)}(j)]^+$ and $p(j) \leftarrow [p(j) - t_{\theta(j)}(j)]^+$, where $[x]^+ = \max(0,\ x)$. Note that the client suffers playback starvation when $b(j) - x_{\theta(j)}(j) < 0$, that is, when the frame that is supposed to be removed is not in the prefetch buffer.

III. JSQ PREFETCH POLICY

For each forward slot the base station must decide which packets to transmit from the J ongoing streams. The prefetch policy is the rule that determines which packets are transmitted. The maximum number of packets that can be transmitted in a forward slot is S. The Join–the–Shortest–Queue (JSQ) prefetch policy strives to balance the lengths of the prefetched video segments across all of the clients serviced by the base station. The basic idea is to dynamically assign more codes (and thus transmit more packets in parallel) to clients that have only a small reserve of prefetched video in their prefetch buffers. The JSQ prefetch policy is inspired by the Earliest Deadline First (EDF) scheduling policy. The EDF scheduling policy is known to be optimal among the class of non–preemptive scheduling policies for a single wireline link [7]. It is therefore natural to base the prefetch policy on the EDF scheduling policy. With JSQ prefetching the base station selects the packet with the earliest playback deadline for transmission. In other words, the base station transmits the next packet to the playout queues with the shortest segments of prefetched video (i.e., the shortest queues).

In order to simplify the discussion and highlight the main points of our approach we first introduce a basic prefetch policy. This basic prefetch policy assumes that all clients (1) support S parallel channels, and (2) have infinite prefetch buffer space. We shall address these two restrictions in a refined prefetch policy. Also, we initially exclude client interactions, such as pause/resume and temporal jumps; these are discussed in Section V.

A. Basic JSQ Prefetch Policy

Let $z(j),\ j = 1, \ldots, J$, denote the length of the video segment (in seconds of video run time) that is scheduled for transmission to client j in the current forward slot. The following scheduling procedure is executed for every forward slot. At the beginning of the scheduling procedure all $z(j)$'s

are initialized to zero. The base station determines the client j^* with the smallest $p(j) + z(j)$. The base station schedules one packet for transmission (by assigning a code to it) and increments $z(j^*)$: $z(j^*) \leftarrow z(j^*) + t_{\sigma(j^*)}(j^*)/x_{\sigma(j^*)}(j^*)$, where $\sigma(j^*)$ is the frame (number) of stream j^* that is carried (partially) by the scheduled packet. (Although the length of the prefetched video segment grows in increments of $t_n(j)$ seconds whenever the transmission of the $x_n(j)$ packets carrying frame n of stream j is completed; for simplicity we account for partially transmitted frames by incrementing the prefetched segment by $t_n(j)/x_n(j)$ for each transmitted packet. This approach is further justified by error concealment techniques that can decode partial frames [6].) The base station repeats this procedure S times, that is, until the S available codes are used up. At each iteration the base station determines the j^* with the smallest $p(j) + z(j)$, schedules one packet for client j^* and increments $z(j^*)$.

Throughout this scheduling procedure the base station skips packets from a frame that would miss its playback deadline at the client. (Specifically, if frame $\theta(j)$ is to be removed before the end of the upcoming forward slot and if $p(j) < t_{\theta(j)}(j)$ the base station skips frame $\theta(j)$ and prefetches for frame $\theta(j+1)$. Moreover, frame $\theta(j)$ is skipped if $b(j) + k \cdot R(j) < x_{\theta(j)}(j)$, where k is the number of forward slots before frame $\theta(j)$ is removed.)

During the subsequent backward slot the base station waits for the acknowledgments from the clients. (Typical hardware configurations of wireless communication systems allow the clients to acknowledge the packets received in a forward slot of the TDD timing structure, immediately in the following backward slot [8].) If all packets sent to client j are acknowledged by the end of the backward slot we set $p(j) \leftarrow p(j) + z(j)$. If some of the acknowledgments for a stream j are missing at the end of the backward slot, $p(j)$ is left unchanged. (This approach is conservative in that it considers all of the packets sent to a client in a forward slot as lost when one (or more) of the packets (or acknowledgments) is lost. An alternative approach would be to selectively account for the acknowledged packets, however, this would lead to "gaps" in the prefetched videos that are difficult to keep track of.) At the end of the backward slot the scheduling procedure starts over. The $z(j)$'s are re–initialized to zero and the base station schedules packets for the clients with the smallest $p(j) + z(j)$.

With prefetching it is possible that all $N(j)$ frames of video stream j have been prefetched into the client's prefetch buffer but not all frames have been displayed. When a stream reaches this state we no longer consider it in the above JSQ policy. From the base station's perspective it is as if the stream has terminated.

B. Refined JSQ Prefetch Policy

In this section we discuss important refinements of the JSQ prefetch policy. These refinements limit (1) the number of packets, that are sent (in parallel) to a client in a forward slot, and (2) the number of packets that a client may have in its prefetch buffer. Suppose that the clients $j,\ j = 1, \ldots, J$, support at most $R(j)$ parallel channels, and have limited prefetch buffer capacities of $B(j)$ packets. Let $r(j),\ j = 1, .., J$, denote the number of packets scheduled for client j in the upcoming forward slot. Recall that $b(j)$ is the current number of packets in the prefetch buffer of client j. The refinements work as follows. Suppose that the base station is considering scheduling a packet for transmission to client j^*. The base station schedules the packet only if

$$r(j^*) \leq R(j^*) - 1, \tag{1}$$

and

$$b(j^*) \leq B(j^*) - 1. \tag{2}$$

If one of these conditions is violated, that is, if the packet would exceed the number of parallel channels of client j^* or the packet would overflow the prefetch buffer of client j^*, the base station removes connection j^* from consideration. The base station next finds a new j^* that minimizes $p(j) + z(j)$. If conditions (1) and (2) hold for the new client j^*, we schedule the packet, update $z(j^*)$ and $r(j^*)$, and continue the procedure of transmitting packets to the clients that minimize $p(j) + z(j)$. Whenever one of the conditions (1) or (2) (or both) is violated we skip the corresponding client and find a new j^*. This procedure stops when we have either (1) scheduled S packets, or (2) skipped over all J streams. The JSQ scheduling algorithm can be efficiently implemented with a sorted list (using $p(j)$ as the sorting key).

C. JSQ Prefetching in third generation CDMA systems and TDMA systems

We emphasize at this juncture that our streaming protocol based on the JSQ prefetch policy can be employed with any of the rate adaptation techniques for wireless communication. We have illustrated the JSQ prefetch policy in the context of a second generation CDMA IS–95 (Rev. B) system, where rate adaptation is achieved through code aggregation, that is, by dynamically assigning multiple code channels to one particular client (stream). We have considered a scenario where the base station uses S

orthogonal codes for the streaming service and client j can receive (and process) $R(j)$ code channels in parallel.

We note that JSQ prefetching is equally well suited for the rate adaptation techniques used in third generation CDMA systems. The third generation North American CDMA standard, cdma 2000, adapts data rates by employing code aggregation and variable spreading gains [9]. With a 5 MHz carrier, for instance, the data rate of one CDMA code channel can be adjusted from 9.6 kbit/sec to 614.4 kbit/sec by varying the spreading factor. With code aggregation a maximum data rate of 2,048 kbit/sec can be achieved. Similarly, the Universal Mobile Telecommunications System (UMTS) Wideband CDMA (WCDMA) standard for Europe and Japan adapts data rates by employing code aggregation in conjunction with variable spreading gains and code puncturing [10]. UMTS may run in the Frequency Division Duplex (FDD) or the Time Division Duplex (TDD) mode. In the TDD mode, which we assume throughout this paper, time is divided into 10 msec frames, which are subdivided into 16 minislots of 625 μsec each. A minislot is spread with a unique code and may carry either forward or backward traffic. Each minislot has a total of 15 code channels, which may be dynamically assigned to the individual clients. To illustrate JSQ prefetching in these third generation CDMA systems, suppose that the forward (base station to clients) transmission capacity allocated to streaming services allows for the transmission of S packets in one forward slot of the TDD. The base station executes the JSQ scheduling algorithm to determine the number of packets $z(j)$ scheduled for client $j,\ j = 1, \ldots, J$, in the upcoming forward slot. The spreading factors and code channels for the forward slot are assigned according to the transmission schedule $z(j),\ j = 1, \ldots, J$.

The JSQ prefetch policy is also suited for the rate adaptation techniques of wireless Time Division Multiple Access (TDMA) systems. In Generalized Packet Radio Service (GPRS) and Enhanced GPRS (EGPRS), the GSM standards for packet data services, rate adaptation is achieved through adaptive coding and time slot aggregation, that is, by assigning multiple time slots within a GSM frame to a particular client (stream) [11]. Up to eight time slots per GSM frame can be allocated to a client, giving maximum data rates of 160 kbit/sec in GPRS and 473 kbit/sec in EGPRS. Similarly, in GPRS–136, the IS–136 TDMA standard for packet data services, rate adaptation is achieved through adaptive modulation and time slot aggregation [12]. Up to three time slots per 20 msec TDMA frame can be allocated to a client, giving a maximum data rate of 44.4. kbit/sec. Suppose that the base station allocates S time slots of the forward portion of the TDMA frame to continuous media streaming. The

base station executes the JSQ scheduling algorithm to determine the number of time slots (packets) $z(j)$ assigned to client j, $j = 1, \ldots, J$, in the upcoming TDMA frame. The base station then assigns $z(j)$ time slots to client j in the forward portion of the TDMA frame (using, for instance, the Dynamic Slot Assignment (DSA++) protocol [13]).

D. Simulation of Prefetch Protocol

In this section we describe the simulations of our protocol for continuous media streaming in wireless environments. In our simulations we consider a generic wireless communication system with Time Division Duplex. We assume throughout that the base station allocates S = 15 channels (e.g., orthogonal codes in CDMA or time slots in TDMA) to continuous media streaming. We assume that each channel provides a data rate of C = 64 kbit/sec in the forward (downlink) direction. Throughout we consider scenarios where all J clients have the same buffer capacity of B packets and support the same number of parallel channels R, i.e., $B(j) = B$ and $R(j) = R$ for all $j = 1, \ldots, J$.

We evaluate our streaming protocol for two different encodings of video streams: (i) video streams encoded at a Constant Bit Rate (CBR) and a constant frame rate, and (ii) video streams encoded at a Variable Bit Rate (VBR) and a constant frame rate (e.g., MPEG–1 encodings). In the CBR scenario we assume a video stream with a frame rate of 25 frames/sec and a bit rate (including packet headers and padding) of $\bar{r}$ = 64 kbit/sec.

For the VBR MPEG–1 scenario we obtained 7 MPEG–1 traces, which give the number of bits in each encoded video frame, from the public domain [14], [15], [16]. The 7 video traces were used to create 10 pseudo traces, each 40,000 frames long. The statistics of the pseudo traces are summarized in Table I. The *bean*, *bond*, *soccer*, and *terminator* traces were created by multiplying the frame sizes of the video traces from [16] by a constant to bring the average bit rate of the packetized video stream (including headers and padding) to $\bar{r}$ = 64 kbit/sec. The *oz* trace was created by taking the first 40,000 frames of the MPEG encoding from [15] and scaling the frame sizes to bring the average bit rate to $\bar{r}$ = 64 kbit/sec. Finally, the four *star wars* traces were obtained by dividing the MPEG encoding from [14] into four segments of 40,000 frames each and then scaling the average bit rate of the segments to $\bar{r}$ = 64 kbit/sec. Although the 10 pseudo traces are not traces of actual video objects, we believe that they reflect the characteristics of VBR video streams (e.g., basic quality layers of hierarchical MPEG encodings [17]) that will be delivered in wireless environments. In summary, we have 10 VBR video streams with 40,000 frames and a constant frame rate of 25 frame/sec.

TABLE I
MPEG–1 TRACE STATISTICS; THE AVERAGE BIT RATE IS 64 KBIT/SEC FOR ALL VIDEOS.

	Frames		
Trace	Peak in kbit/sec	Peak/Mean	Std. Dev. in bits
bean	646	10.1	752.2
bond	832	13.0	1919.8
lambs	1178	18.4	551.9
oz	538	8.4	636.1
soccer	441	6.9	377.3
star wars 1	698	10.9	616.5
star wars 2	845	13.2	1362.4
star wars 3	768	12.0	522.0
star wars 4	544	8.5	669.2
terminator	467	7.3	479.6

For each of the J ongoing streams in the wireless cell we select randomly one of the MPEG traces. We generate random starting phases $\theta(j),\ j = 1, \ldots, J$, into the selected traces. The $\theta(j)$'s are independent and uniformly distributed over the lengths of the selected traces. The frame $\theta(j)$ is removed from the prefetch buffer of client j at the end of the first frame period. All clients start with empty prefetch buffers. Furthermore, we generate random stream lengths $N(j),\ j = 1, \ldots, J$. The $N(j)$'s are independent and are drawn from an exponential distribution with mean N_T frames (corresponding to a video run time of T seconds). We initially assume that the client consumes without interruption $N(j)$ frames starting at frame number $\theta(j)$ of the selected trace. The trace is wrapped around if $\theta(j) + N(j)$ extends beyond the end of the trace. When the $N(j)$th frame is removed from the prefetch buffer of client j, we assume that the client immediately requests a new video stream. For the new video stream we again select randomly one of the traces, a new independent random starting phase $\theta(j)$ into the trace, and a new independent random stream lifetime $N(j)$. Thus there are always J streams in progress.

In simulating the wireless links we follow the well–known Gilbert–Elliot model [18], [19]. We simulate each wireless link (consisting of up to $R(j)$ parallel code channels) as an independent discrete–time Markov Chain with two states: "good" and "bad". (We assume that all parallel code channels of a wireless link (to a particular client) are either in the good state or bad state.) The transition probabilities of the Markov Chains are set to typical values such that the steady state probability of being in the "good" state is $\pi_g = 0.99$ (and $\pi_b = 0.01$ accordingly); the average sojourn times are 9 seconds for the

”good” state and 1 second for the ”bad” state. We set the channel error probabilities such that a packet is lost with probability $P_l^g = 0.05$ in the good channel state, and with probability $P_l^b = 1$ in the bad channel state. We assume that acknowledgments are never lost in the simulations.

Figures 2 and 3 shows typical sample path plots from the simulations. In this experiment we

PSfrag replacements

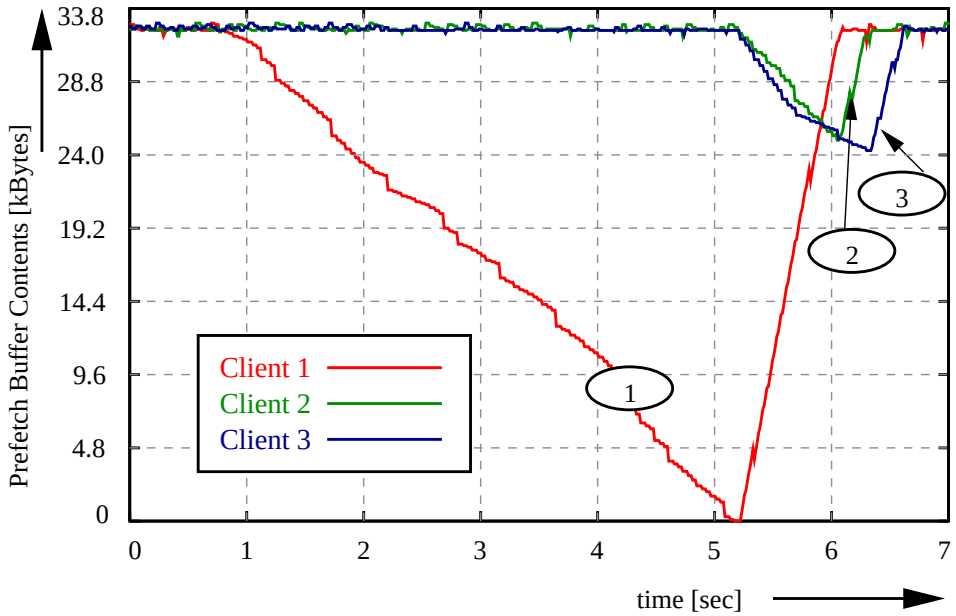

Fig. 2. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 starts over

simulate the streaming of VBR MPEG–1 videos to clients with a buffer capacity of $B = 32$ kBytes. The figure shows the prefetch buffer contents of three clients in kBytes. The plots illustrate the collaborative nature of the JSQ prefetch policy in conjunction with the rate adaptation of the wireless communication system. We observe from Figure 2 that at time $t = 5.1$ sec the buffer content of client 1 drops to zero. This is because the video stream of client 1 ends at this time; the client selects a new video stream and starts over with an empty prefetch buffer. Note that already at time $t = 1.0$ second all frames of the ”old” video stream have been prefetched into the client’s buffer and the client continued to consume frames without receiving any transmissions. When the client starts over with an empty prefetch buffer, the JSQ prefetch policy gives priority to this client and quickly fills its prefetch buffer. While the prefetch buffer of client 1 is being filled the JSQ prefetch policy reduces the transmissions to the other clients; they ”live off” their prefetched reserves until client 1 catches up with them. Notice that the buffer of client 1 is then filled faster than the buffers of clients 2 and 3. This is because the JSQ prefetch policy

PSfrag replacements

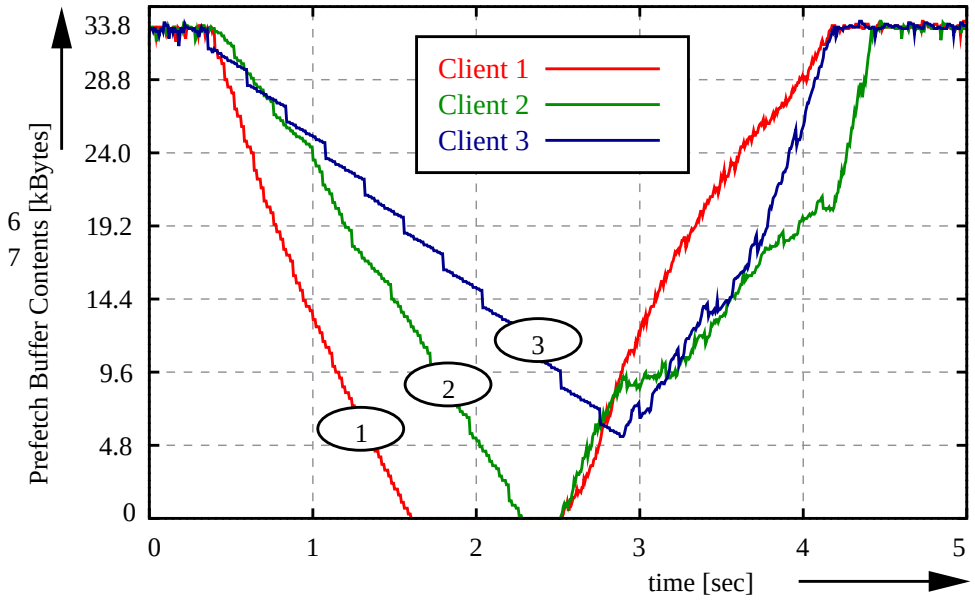

Fig. 3. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 experiences a bad channel.

strives to balance the lengths of the prefetched video segments (in seconds of video run time) in the clients' buffers; clients 2 and 3 just happen to have video segments with lower bit rates in their buffers in the time period from 5.8 sec to 6.4 sec.

Notice from Figure 3 that at time t = 0.4 sec the buffer occupancy of client 1 drops. This is because this client experiences a bad channel that persists for 2.1 sec (a rather long period chosen for illustration, in our numerical work we set the average sojourn time in the bad channel state to 1 sec), that is, the client is temporarily cut off from the base station. The prefetched reserves allow the client to continue playback during this period. As the prefetched reserves of client 1 dwindle the JSQ prefetch policy allocates larger transmission capacities to it. This, however, cuts down on the transmissions to the other clients, causing their prefetched reserves to dwindle as well. This degrades the performance of the streaming protocol as smaller prefetched reserves make client starvation more likely. Loosely speaking, the JSQ prefetch policy tends to waste transmission resources on clients that experience a bad channel. Adverse transmission conditions to just one client can decrease the prefetched reserves of all clients in the wireless cell. We address this shortcoming of the purely JSQ based streaming protocol in the next section, where we introduce channel probing. In the next section we also conduct a detailed quantitative evaluation of the JSQ prefetch protocol.

IV. CHANNEL PROBING

In this section we introduce a channel probing refinement designed to improve the performance of the purely JSQ based prefetch protocol. Note that the prefetch protocol introduced in Section III does not directly take the physical characteristics of the wireless channels into consideration. The JSQ transmission schedule is based exclusively on the prefetch buffer contents at the clients (and the consumption rates of the video streams). Wireless channels, however, typically experience location–dependent, time–varying, and bursty errors, that is, periods of adverse transmission conditions during which all packets sent to a particular client are lost. These periods are modeled by the ”bad” channel state in the Gilbert–Elliot model. Especially detrimental to the prefetch protocol's performance are the persistent bad channels of long–term shadowing that is caused by terrain configuration or obstacles. Long–term shadowing typically persists for hundreds of milliseconds, even up to seconds [20]. To see how long–term shadowing degrades the performance of the prefetch protocol, consider a client that is just experiencing the onset of a bad channel. Suppose that the JSQ algorithm schedules a packet for this client in the upcoming slot. Due to the bad channel the packet is lost. At the same time, the ongoing consumption reduces the client's prefetched reserve. It is therefore likely that the JSQ algorithm again schedules a packet (or multiple packets) for the client in the next slot. As the bad channel persists the client's prefetched reserve dwindles and (provided the other clients are not experiencing a bad channel or the consumption of extraordinarily large frames at the same time) the JSQ algorithm tends to schedule more and more (up to $R(j)$) packets for the client in the following slots. All these packets are lost, however, as long as the bad channel persists. Thus, by scheduling multiple packets per slot for a client experiencing a persistent bad channel, the JSQ algorithm tends to waste transmission resources. As illustrated in Figure 3(b), the excessive transmission resources expended on the client that experiences the bad channel, tend to reduce the prefetched reserves of all the other clients in the wireless cell. This makes playback starvation — not only for the client experiencing the bad channel, but all the other clients as well — more likely.

To fix this shortcoming we introduce the channel probing refinement, which is inspired by recent work on channel probing for power saving [21]. The basic idea is to start *probing* the channel (client) when acknowledgment(s) are missing at the end of a backward slot. While probing the channel the base station sends at most one packet (probing packet) per forward slot to the affected client. (A more complex strategy might send a probing packet every $k,\ k \geq 1$, forward slots.) The probing continues

until an acknowledgment for a probing packet is received. More specifically, if the acknowledgment for at least one packet sent to client j in a forward slot is missing at the end of the subsequent backward slot, we set $R(j) = 1$. This allows the JSQ algorithm to schedule at most one packet (probing packet) for client j in the next forward slot. If the acknowledgment for the probing packet is returned by the end of the next backward slot, we set $R(j)$ back to its original value; otherwise we continue probing with $R(j)$ = 1.

A. Simulation Results

We now present a detailed quantitative evaluation of the JSQ prefetch protocol. We first define two key measures of the performance of a streaming protocol. We define the *bandwidth efficiency* ξ of a wireless streaming protocol as the sum of the average rates of the streams supported by the base station divided by the total available effective transmission capacity of the base station, i.e.,

$$\xi = \frac{J \cdot \bar{r}}{[\pi_g \cdot (1 - P_l^g) + \pi_b \cdot (1 - P_l^b)] \cdot C \cdot \min(RJ,\ S)}.$$

We define the *client starvation probability* P_{loss} as the long run fraction of encoding information (packets) that misses its playback deadline at the clients. We conservatively consider all $x_n(\cdot)$ packets of frame n as deadline misses when at least one of the frame's packets misses its playback deadline. We warm up each simulation for a period determined with the Schruben's test and obtain confidence intervals on the client starvation probability P_{loss} using the method of batch means. We run the simulations until the 90 % confidence interval of P_{loss} is less than 10 % of its point estimate.

Unless stated otherwise, all the following experiments are conducted for the streaming of VBR MPEG video to clients with a buffer capacity of B = 32 kBytes and support for R = 15 parallel channels. The average lifetime of the video stream is set to T = 10 minutes unless stated otherwise. Throughout, the base station has a total of S = 15 channels available for video streaming.

Figure 4 shows the client starvation probability P_{loss} without and with channel probing as a function of the average sojourn time in the "bad" channel state. For this experiment the number of clients is fixed at J = 13 (and 12 respectively). We observe from the figure that over a wide range of channel conditions, channel probing is highly effective in reducing the probability of client starvation. In scenarios with persistent bad channel conditions, probing reduces P_{loss} by over one order of magnitude. We set the average sojourn time in the "bad" channel state to 1 sec for all the following experiments.

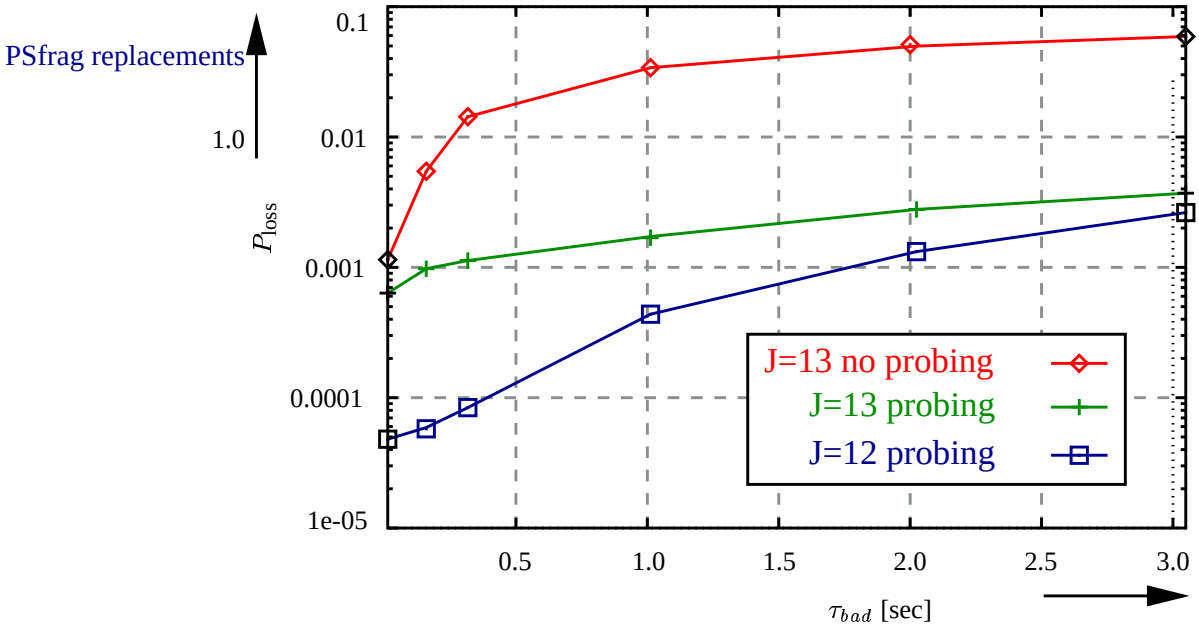

Fig. 4. Client starvation probability P_{loss} as a function of the average sojourn time in the "bad" channel state for JSQ prefetching without and with channel probing for VBR video.

Figure 5 shows the client starvation probability P_{loss} as a function of the maximum number of parallel channels R that can be assigned to an individual client. The figure gives results for JSQ prefetching without and with channel probing. We observe from Figure 5 that both for prefetching without and with channel probing, P_{loss} drops by over one order of magnitude as R increases from one to two, allowing for collaborative prefetching through the lending and borrowing of channels. Now consider prefetching with J = 13 clients. For prefetching with channel probing, P_{loss} drops steadily as R increases. For prefetching without channel probing, however, P_{loss} increases as R grows larger than two, that is, allowing for more extensive lending and borrowing of channels is detrimental to performance in this scenario.

This is because JSQ prefetching without channel probing tends to waste transmission channels on a client experiencing a persistent bad channel. This reduces the prefetched reserves of all clients in the cell, thus increasing the likelihood of client starvation. The larger the number of parallel channels R that can be assigned to an individual client the larger is this waste of transmission channels (resulting in a larger P_{loss}). Now consider prefetching with J = 7 clients. P_{loss} drops both for prefetching without and with channel probing as R increases to nine. As R grows larger than nine, however, P_{loss} increases for prefetching without channel probing. This is because in this low load scenario a client experiencing

PSfrag replacements

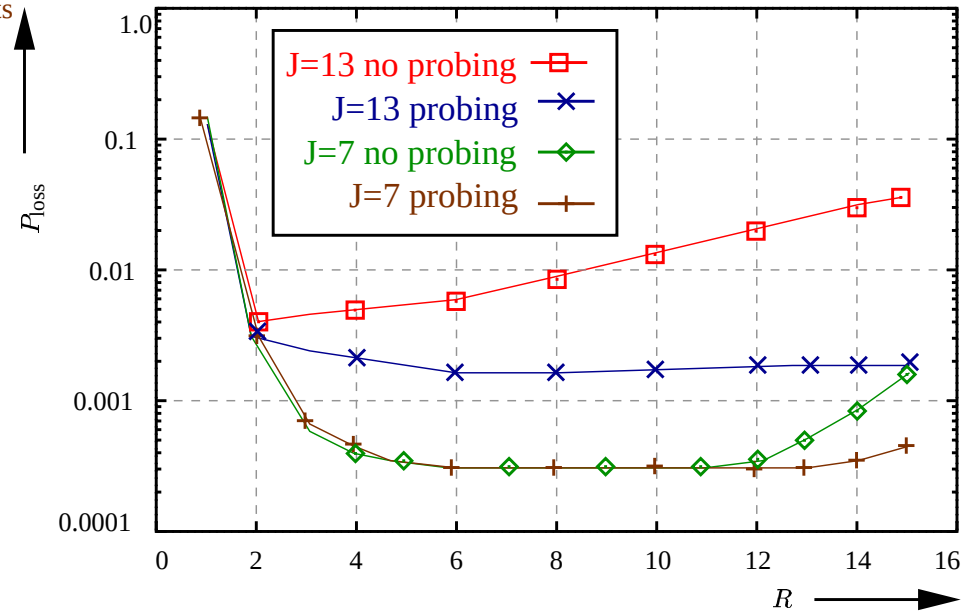

Fig. 5. Client starvation probability P_{loss} as a function of the maximum number of channels R per client for JSQ prefetching without and with channel probing for VBR video.

a bad channel may occupy nine out of the 15 available channels without doing much harm to the prefetched reserves of the other six clients. (The noise level, however, is unnecessarily increased.) As R grows larger than nine, however, a client experiencing a persistent bad channel tends to reduce the prefetched reserves of the other clients.

Another important observation from Figure 5 is that already a low–cost client with support for a few parallel channels allows for effective prefetching.

Figure 6 shows the client starvation probability P_{loss} as a function of the bandwidth efficiency ξ. The plots are obtained by varying the number of clients J. Noteworthy is again the effectiveness of channel probing: P_{loss} is generally over one order smaller with channel probing. Importantly, the results in Figure 6 indicate that a crude admission control criterion that limits the bandwidth efficiency to less than 0.9, say, is highly effective in ensuring small client starvation probabilities. We note, however, that more research is needed on admission control for streaming in wireless environments.

Figure 7 shows the client starvation probability P_{loss} as a function of the client buffer capacity B. The results demonstrate the dramatic improvement in performance that comes from prefetching. For J = 13 ongoing VBR streams the client starvation probability drops by over two orders of magnitude as the clients' buffers increase from 8 kBytes to 128 kBytes. (A buffer of 128 kBytes can hold on average 16

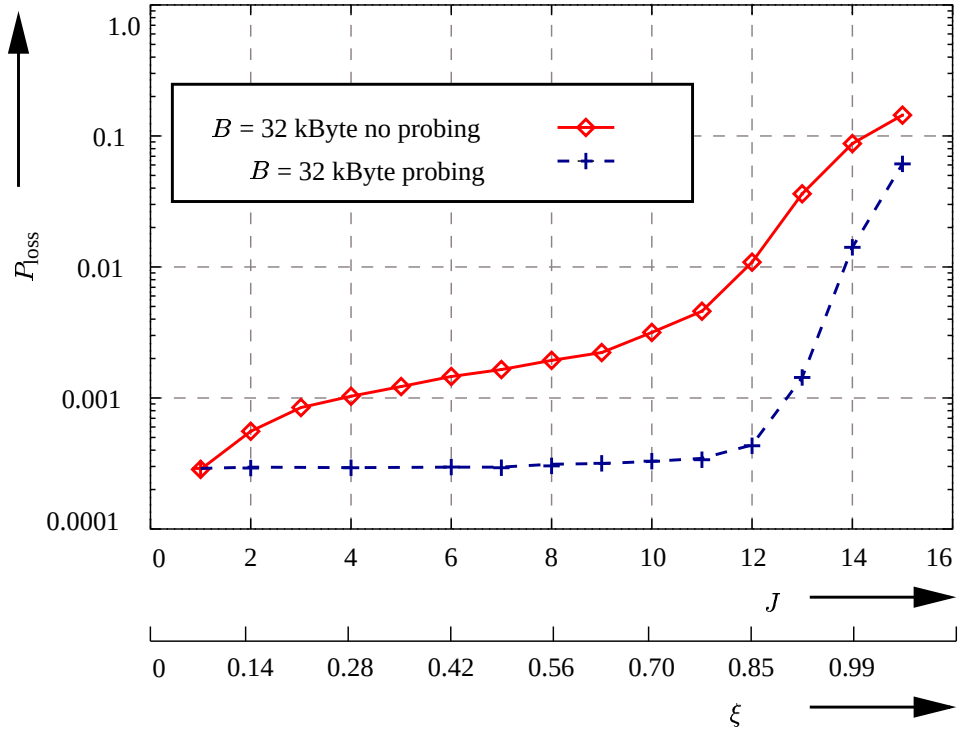

Fig. 6. Client starvation probability P_{loss} as a function of the bandwidth efficiency ξ (obtained by varying the number of clients J) for VBR video.

PSfrag replacements

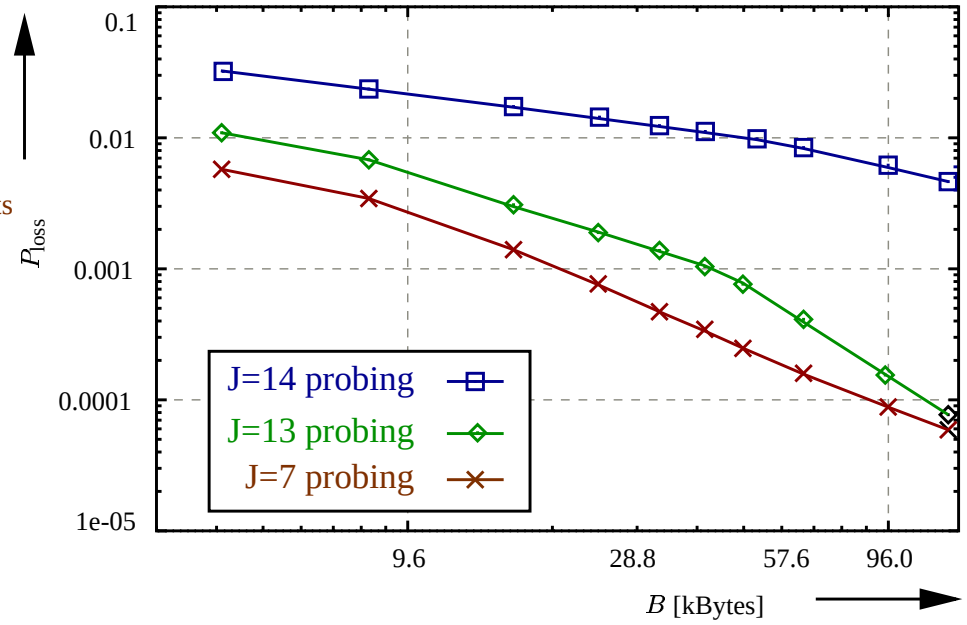

Fig. 7. Client starvation probability P_{loss} as a function of the client buffer capacity B for VBR video.

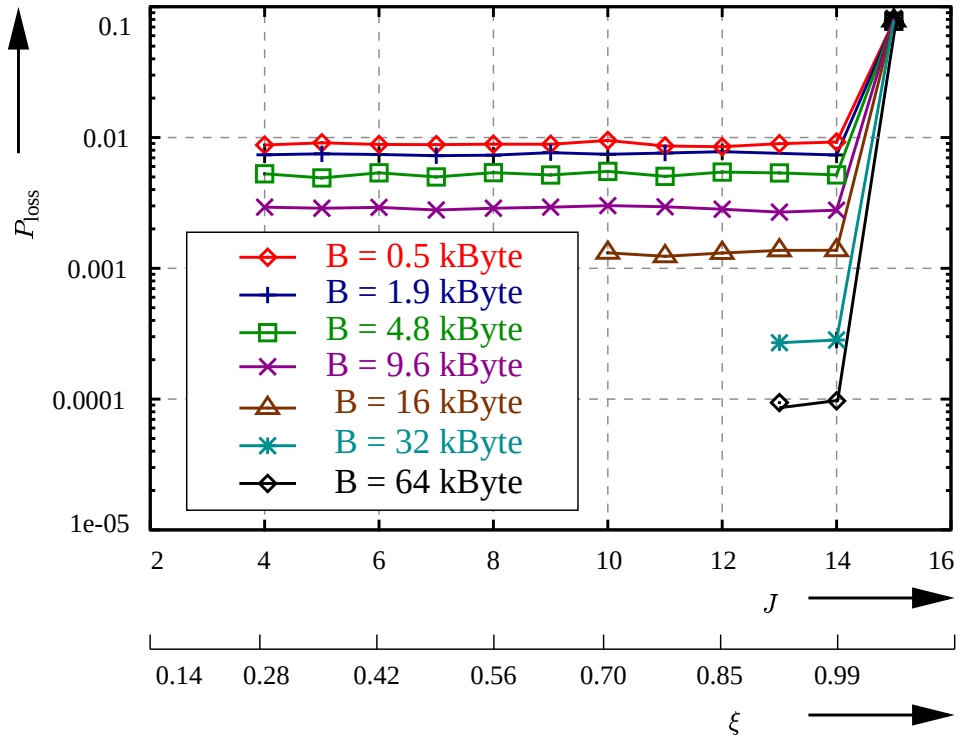

Fig. 8. Client starvation probability P_{loss} as a function of the bandwidth efficiency ξ (obtained by varying the number of clients J) for CBR video.

second segments of the VBR videos with an average rate of $\bar{r}$ = 64 kbit/sec.) With client buffers of B = 128 kBytes and J = 13 ongoing streams our prefetch protocol achieves a client starvation probability of less than 10^{-4} and a bandwidth efficiency of 90%! Our protocol achieves this remarkable performance for the streaming of bursty VBR videos with a typical peak–to–mean ratio of the frame sizes of ten. In the long run each wireless link is in the ”bad” state (where all packets are lost) for one percent of the time; the average sojourn time in the ”bad” state is one second.

Figure 8 shows the client starvation probability P_{loss} as a function of the bandwidth efficiency ξ for the streaming of CBR videos. Figure 9 shows the client starvation probability P_{loss} as a function of the client buffer capacity B for the streaming of CBR videos. Comparing the CBR plots with the corresponding VBR plots we observe that there are significant differences only at very high bandwidth efficiencies. For J = 14 ongoing streams (i.e., a bandwidth efficiency of ξ = 0.99) and client buffers of B = 32 kBytes we achieve a client starvation probability of roughly $3 \cdot 10^{-4}$ for CBR video whereas the client starvation probability is roughly $1.5 \cdot 10^{-2}$ for VBR video. It is also noteworthy that for CBR video for a fixed buffer capacity B the client starvation probability is roughly constant as the number

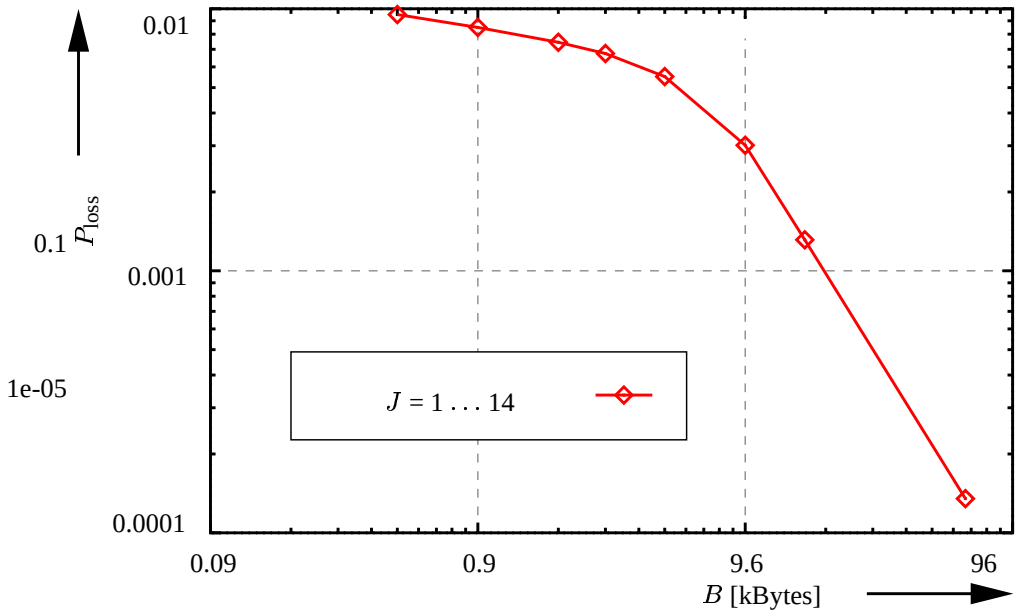

Fig. 9. Client starvation probability P_{loss} as a function of the client buffer capacity B for CBR video.

of ongoing streams increases from 1 to 14. This is because in the the CBR scenario client starvation is only caused by channel outages (whereas in the VBR scenario the playout of high rate video scenes may also cause starvation). We observed in our simulations that with the adopted typical channel parameters, channel outages for a particular client are sufficiently far apart to typically allow the client to refill its buffer completely between outages. When an outage occurs the client can thus typically continue playback for a duration of $B/\bar{r}$. Since it is rather rare that two or more clients experience the end of a bad channel roughly at the same time the buffer of the affected client is quickly refilled as the other ongoing streams lend their transmission channels. In the VBR scenario the situation is slightly different as the other ongoing streams may play out high rate video scenes at that time and therefore not as readily lend their channels as in the CBR scenario.

Table II gives the client starvation probability P_{loss} as a function of the average stream lifetime T for the streaming of VBR video to J = 13 clients each with a buffer capacity of B = 64 kBytes. Our prefetching protocol performs very well for stream lifetimes on the order of minutes or longer. For stream lifetimes shorter than one minute P_{loss} increases considerably as the lifetime decreases. This is because stream lifetimes this short allow for very little time to build up prefetched reserves. Even for an average stream lifetime of T = 10 sec, however, prefetching reduces the client's starvation probability from $2.8{\cdot}10^{-2}$ without any prefetching to $4.5{\cdot}10^{-3}$ with prefetching. We observed in our simulations

TABLE II
CLIENT STARVATION PROBABILITY P_{LOSS} AS A FUNCTION OF THE AVERAGE STREAM LIFETIME T FOR VBR VIDEO.

T [sec]	10	50	100	600	1200
P_{loss} [10^{-4}]	45	7.3	4.6	4.4	2.1

TABLE III
CLIENT STARVATION PROBABILITY P_{LOSS} AS A FUNCTION OF THE START–UP LATENCY FOR VBR VIDEO.

start–up lat. [msec]	0	40	80	120	400
P_{loss} [10^{-5}]	7.7	5.0	3.6	2.2	0.7

that losses occur typically right at the beginning of the video playback when the client has no prefetched reserves. This motivates us to introduce a short start–up latency allowing the client to prefetch into its prefetched buffer for a short period of time without removing and displaying video frames. Table III gives the client starvation probability P_{loss} as a function of the start–up latency for J = 13 ongoing VBR streams and client buffers of B = 128 kBytes. We observe from Table III that very short start–up latencies reduce the client starvation probability significantly; a start–up latency of 400 msec, for instance, reduces the client starvation probability by roughly one order of magnitude. With a start–up latency of 400 msec the client prefetches for 400 msec without removing video frames from its prefetch buffer; the first frame is removed and displayed at time 400 msec + $t_{\theta(j)}(j)$, where $t_{\theta(j)}(j)$ denotes the frame period of the first frame of the video stream.

V. CLIENT INTERACTIONS

In this section we adapt our streaming protocol to allow for client interactions, such as pause/resume and temporal jumps. With JSQ prefetching these client interactions can be performed with minimal delay. Whenever a user invokes an interactive request, the client sends a message indicating the interaction to the base station. The base station forwards the message to the origin/proxy server providing the video stream. The support of the client interactions by the origin/proxy server and the network connecting the server to the base station is beyond the scope of this study. We assume that the appropriate portions of the video are delivered to the base station in a timely fashion.

Suppose that the user for stream j pauses the video. Upon receiving notification of the action, the base station can simply remove stream j from consideration until it receives a resume message from the client. While the client is in the paused state, it's prefetch buffer contents remain unchanged; a slightly

TABLE IV
CLIENT STARVATION PROBABILITY P_{LOSS} AS A FUNCTION OF THE AVERAGE SPACING D BETWEEN CLIENT INTERACTIONS FOR VBR VIDEO.

D [sec]	10	50	100	250	500	1000	5000
P_{loss} [10^{-4}]	586	214	67	9.0	7.5	3.6	3.2

more complex policy would be to fill the corresponding buffer once all the other ("unpaused") client buffers are full or reach a prespecified level.

Suppose that the user for stream j makes a temporal jump of δ frames (corresponding to t_δ seconds of video runtime) into the future. If $t_\delta < p(j)$ we discard δ frames from the head of the prefetch buffer and set $p(j) \leftarrow p(j) - t_\delta$; $b(j)$ is adjusted accordingly. If $t_\delta \geq p(j)$ we set $p(j) \leftarrow 0$ and $b(j) \leftarrow 0$ and discard the prefetch buffer contents. Finally, suppose that the user for stream j makes a backward temporal jump. In this case we set $p(j) \leftarrow 0$ and $b(j) \leftarrow 0$ and discard the prefetch buffer contents.

In terms of performance, pauses actually improve performance because the video streams that remain active have more transmission capacity to share. Frequent temporal jumps, however, can degrade performance because prefetch buffers would be frequently set to zero. We now give some simulation results for client interactions. In our simulations we consider only forward and backward temporal jumps and ignore pauses because pauses can only improve performance. We furthermore assume that $t_\delta > p(j)$ for all forward temporal jumps. Thus, the prefetch buffer is set to zero whenever the corresponding user invokes such an interaction. Our results give therefore a conservative estimate of the actual performance. In our simulations, we assume that each user performs temporal jumps repeatedly, with the time between two successive jumps being exponentially distributed with mean D seconds. Table IV gives the client starvation probability P_{loss} as a function of the average spacing D between temporal jumps. This experiment was conducted for the streaming of VBR MPEG video to clients with buffers of B = 64 kbyte and support for R = 15 parallel channels. The bandwidth efficiency is fixed at 92 % (J = 13). The average stream lifetime is fixed at T = 1200 sec. As we would expect, the loss probability increases as the rate of temporal jumps increases, however, the increase is not significant for a sensible number of temporal jumps.

VI. RELATED WORK

There is a large body of literature on providing QoS in wireless environments. Much of the work in this area has focused on mechanisms for channel access; see Akyildiz *et al*. [22] for a survey. Choi

and Shin [23] have recently proposed a comprehensive channel access and scheduling scheme for supporting real–time traffic and non–real–time traffic on the uplinks and downlinks of a wireless LAN.

Recently, packet fair scheduling algorithms that guarantee clients a fair portion of the shared transmission capacity have received a great deal of attention [24], [25], [26], [27]. These works adapt fair scheduling algorithms originally developed for wireline packet–switched networks to wireless environments. They address the key difference between scheduling and resources allocation in wireline and wireless environments: wireline links have a fixed transmission capacity while wireless links experience location–dependent, time–varying, and bursty errors, which result in situations where the shared transmission capacity is temporarily available only to a subset of the clients. Another line of work addresses the efficiency of reliable data transfer over wireless links [28], [29].

We note that to our knowledge *none of the existing schemes for providing QoS in wireless environments takes advantage of the special properties (predictability and prefetchability) of prerecorded continuous media*, that are expected to account for a large portion of the future Internet traffic. There is an extensive literature on the streaming of prerecorded continuous media, in particular VBR video, over *wireline* packet–switched networks; see Krunz [30] for a survey. In this literature a wide variety of smoothing and prefetching schemes is explored to efficiently accommodate VBR video on fixed bandwidth wireline links. Among these schemes is a prefetching scheme based on the Join–the–Shortest–Queue (JSQ) principle developed by Reisslein and Ross [31]. Their scheme is designed for a Video on Demand service with VBR encoded fixed frame rate MPEG video over an ADSL network or the cable plant. The protocol proposed in this paper differs from the protocol in [31] in two major aspects. First, the protocol in [31] is designed for a shared wireline link of fixed capacity. It does not handle the location–dependent, time–varying, and bursty errors that are typical for wireless environments. Secondly, the protocol in [31] is designed for fixed frame rate video. It assumes that all ongoing video streams have the same frame rate. Furthermore, it requires the synchronization of the frame periods of the ongoing streams. Our protocol in this paper, on the other hand, does not require synchronization of the ongoing video streams. Our protocol accommodates video streams with different (and possibly time–varying) frame rates. It is thus well suited for H.263 encodings which are expected to play an important role in video streaming in wireless environments.

Elaoud and Ramanathan [32] propose a scheme for providing network level QoS to flows in a wireless CDMA system. Their scheme dynamically adjust the signal to interference and noise ratio require-

ments of flows based on MAC packet deadlines and channel conditions. The Simultaneous MAC Packet Transmission (SMPT) scheme of Fitzek *et al*. [33] provides transport level QoS by exploiting rate adaptation techniques of CDMA systems. The SMPT scheme delivers transport layer segments (e.g., UDP or TCP segments, which are divided into several MAC packets) with high probability within a permissible delay bound. Our work in this paper differs from the network/transport level schemes [32], [33] in several aspects. First, [32], [33] propose decentralized schemes for backward (uplink) transmissions, that is, the schemes are designed for uncoordinated transmissions from distributed clients to a central base station. Secondly, there is no prefetching in [32], [33]; the SMPT scheme resorts to rate adaptation (i.e., parallel packet transmissions) only to recover from gaps caused by errors on the wireless link within a given TCP or UDP segment. Moreover, [32], [33] do not take the characteristics of the application layer traffic into consideration; the scheme [32] operates on one MAC packet at a time and SMPT [33] operates on one TCP or UDP segment at a time. Our protocol in this paper, on the other hand, exploits two special properties of the *prerecorded* continuous media streaming traffic: (1) the client consumption rates over the duration of the playback are known before the streaming commences, and (2) while the continuous media stream is being played out at the client, parts of the stream can be prefetched into the client's memory.

VII. CONCLUSION

We have developed a high performance prefetching protocol for the streaming of prerecorded continuous media from a base station to wireless clients. Our prefetching protocol can be employed on top of any of the rate adaptation techniques of wireless communication systems. Our protocol accommodates CBR and VBR encodings as well as fixed frame rate and variable frame rate encodings. Channel probing is critical for the performance of our protocol. With channel probing the base station allocates transmission resources in a judicious manner avoiding the allocation of large portions of the available transmission capacity to clients experiencing adverse transmission conditions.

In our ongoing work we study more sophisticated channel probing schemes. One of the investigated approaches is to probe more aggressively (i.e., with multiple packets per slot) when the affected client has a small prefetched reserve. Clients with a large prefetched reserve, on the other hand, are probed less aggressively (i.e., with one packet every kth slot, with $k > 1$). Another avenue for future research is service differentiation among the ongoing streams. In a crude service differentiation scheme the base station prefetches for low priority clients only if the prefetched reserves of the high priority clients have

reached prespecified levels. In our ongoing work we also conduct more extensive evaluations of our prefetching protocol with traces of MPEG–4 and H.263 encoded videos.

REFERENCES

[1] G. A. Gibson, J. S. Vitter, and J. Wilkes, "Storage and I/O issues in large–scale computing," in *ACM Workshop on Strategic Directions in Computing Research, ACM Computing Surveys*, 1996.

[2] "Streaming media caching white paper," Technical Report, Inktomi Corporation, 1999, `http://www.inktomi.com/products/traffic/streaming.html`.

[3] L. Roberts and M. Tarsala, "Inktomi goes wireless; forms alliances," in *CBS MarketWatch*, March 14th 2000.

[4] S. Nanda, K. Balachandran, and S. Kumar, "Adaptation techniques in wireless packet data services," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 54–64, Jan. 2000.

[5] EIA/TIA-95 Rev. B, "Mobile station–base station compatibility standard for dual–mode wideband spread spectrum cellular systems," 1997.

[6] Y. Wang and Q. Zhu, "Error control and concealment for video communication: A review," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

[7] L. Georgiadis, R. Guerin, and A. K. Parekh, "Optimal multiplexing on a single link: Delay and buffer requirements," in *Proceedings of IEEE Infocom '94*, Toronto, Canada, June 1994.

[8] F. Wegner, "Personal Communication. Siemens AG, Mobile Radio Access Simulation Group, Berlin, Germany," May 2000.

[9] D. Kingsely *et al*, "Evolution of wireless data services: IS–95 to CDMA 2000," *IEEE Communications Magazine*, vol. 36, no. 10, Oct. 1998.

[10] UMTS 30.03, "Universal Mobile Telecommunications System (UMTS); Selection Procedures for the Choice of Radio Transmission Technologies of the UMTS," .

[11] ETSI GSM 03.60, "Digital Cellular Telecommunications System (Phase 2+); General Packet Radio Service (GPRS); Service Description; Stage 2," .

[12] K. Balachandran *et al*, "GPRS–136: High–rate packet data service for north american TDMA digital cellular systems," *IEEE Personal Communications*, June 1999.

[13] G. Anastasi, L. Lenzini, E. Mingozzi, A. Hettich, and A. Kramling, "MAC protocols for wideband wireless local access: Evolution towards wireless ATM," *IEEE Personal Communications*, pp. 53–64, Oct. 1998.

[14] M. W. Garret, *Contributions toward Real-Time Services on Packet Networks*, Ph.D. thesis, Columbia University, May 1993, ftp address and directory of the used video trace: bellcore.com /pub/vbr.video.trace/.

[15] M. Krunz, R. Sass, and H. Hughes, "Statistical characteristics and multiplexing of MPEG streams," in *Proceedings of IEEE Infocom '95*, April 1995, pp. 455–462.

[16] O. Rose, "Statistical properties of MPEG video traffic and their impact on traffic modelling in ATM systems," Tech. Rep. 101, University of Wuerzburg, Institute of Computer Science, Am Hubland, 97074 Wuerzburg, Germany, Feb. 1995.

[17] J. Lee, T. Kim, and S. Ko, "Motion prediction based on temporal layering for layered video coding," in *Proceedings of ITC–CSCC, Vol. 1*, July 1998, pp. 245–248.

[18] E. N. Gilbert, "Capacity of a burst–noise channel," *Bell Systems Technical Journal*, vol. 39, pp. 1253–1266, Sept. 1960.

[19] E. O. Elliot, "Estimates of error rates for codes on burst–noise channels," *Bell Systems Technical Journal*, vol. 42, pp. 1977–1997, Sept. 1963.

[20] M. D. Yocoub, *Foundations of Mobile Radio Engineering*, McGraw Hill, second edition, 1986.

[21] M. Zorzi and R. R. Rao, "Error control and energy consumption in communications for nomadic computing," *IEEE Transactions on Information Theory*, vol. 46, pp. 279–289, Mar. 1997.

[22] I. F. Akyildiz, J. McNair, L. C. Martorell, R. Puigjaner, and Y. Yesha, "Medium access control protocols for multimedia traffic in wireless networks," *IEEE Network*, vol. 13, no. 4, pp. 39–47, July/August 1999.

[23] S. Choi and K. G. Shin, "A unified wireless LAN architecture for real–time and non–real–time communication services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 44–59, Feb. 2000.

[24] C. Fragouli, V. Sivaraman, and M. B. Srivastava, "Controlled multimedia wireless link sharing via enhanced class–based queueing with channel–state–dependent packet scheduling," in *Proceedings of IEEE Infocom '98*, San Francisco, CA, Apr. 1998.

[25] T. S. E. Ng, I. Stoica, and H. Zhang, "Packet fair queueing algorithms for wireless networks with location dependent errors," in *Proceedings of IEEE Infocom '98*, San Francisco, CA, Apr. 1998, pp. 1103–1111.

[26] S. Lu, T. Nandagopal, and V. Bharghavan, "A wireless fair service algorithm for packet cellular networks," in *Proceedings of ACM/IEEE MobiCom '98*, Dallas, TX, Oct. 1998, pp. 10–20.

[27] P. Ramanathan and P. Agrawal, "Adapting packet fair queueing algorithms to wireless networks,," in *Proceedings of ACM MobiCom 1998*, Dallas, TX, Oct. 1998.

[28] E. Amir, H. Balakrishnan, S. Seshan, and R. Katz, "Efficient TCP over networks with wireless links," in *Proceedings of ACM Conference on Mobile Computing and Networking*, Berkeley, CA, Dec. 1995.

[29] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. K. Tripathi, "Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs," *ACM Wireless Networks*, vol. 3, pp. 91–102, 1997.

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

[31] M. Reisslein and K. W. Ross, "A Join–the–Shortest–Queue prefetching protocol for VBR video on demand," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Atlanta, GA, Oct. 1997, pp. 63–72.

[32] M. Elaoud and P. Ramanathan, "Adaptive allocation of CDMA resources for network–level QoS assurance," in *Proceedings of ACM MobiCom 2000*, Boston, MA, Aug. 2000.

[33] F. H. P. Fitzek, B. Rathke, M. Schläger, and A. Wolisz, "Quality of Service Support for Real–Time Multimedia Applications over Wireless Links using the Simultaneous MAC–Packet Transmission (SMPT) in a CDMA environment," in *Proceedings of 5th International Workshop on Mobile Multimedia Communications (MoMuC)*, Berlin, Germany, Oct. 1998, pp. 367–378.