appeared in IEEE Journal on Selected Areas in Communications October 2001

A Prefetching Protocol for Continuous Media Streaming in Wireless Environments

Frank H.P. Fitzek
Technical University Berlin

Martin Reisslein
Arizona State University

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 prerecorded continuous media from (to) a central base station to (from) multiple wireless clients within a wireless cell. Our protocol prefetches parts of the ongoing continuous media streams into prefetch buffers in the clients (base station). 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 streams 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 and H.263 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 prefetch 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 toward accessing the Internet and Web from wireless mobile devices. Analysts predict that

Manuscript received December 18, 2000; revised June 5, 2001.

A shorter version of this article has appeared under the title A Prefetching Protocol for Streaming Prerecorded Continuous Media in Wireless Environments in *Proceedings of SPIE ITCom 2001, Internet Performance and QoS,* Denver, CO, August 2001

F. Fitzek is with the Department of Electrical Engineering, Technical University Berlin, 10587 Berlin, Germany (e–mail: fitzek@ee.tu-berlin.de).

Corresponding Author: Martin Reisslein, Dept. of Electrical Eng., Arizona State University, P.O. Box 877206, Tempe AZ 85287–7206, phone: (480)965–8593, fax: (480)965–8325, e-mail: `reisslein@asu.edu`, web: `www.eas.asu.edu/~mre`.

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. For uninterrupted video playback the client has to decode and display a new video frame periodically; typically every 40 msec with MPEG encoded video. If the client has not completely received a video frame by its playback deadline, the client looses (a part or all of) the video frame and suffers playback starvation. A small probability of playback starvation (typically, 10^{-5} – 10^{-2}) is required for good perceived video quality. In addition to these stringent timing and loss constraints, the video frame sizes (in byte) of the more efficient Variable Bit Rate (VBR) encodings are highly variable; typically with peak–to–mean ratios of 4 – 10 [4]. These properties and requirements make the real–time streaming of video over packet–switched networks — even for the case of wireline networks — a challenging problem [5]. The problem is even more challenging when streaming video over wireless links. Wireless links are typically highly error prone. They introduce a significant number of bit errors which may render a transmitted packet undecodable. The tight timing constraints of real–time video streaming, however, allow only for limited re–transmissions [6]. Moreover, the wireless link errors are typically time–varying and bursty. An error burst (which may persists for hundreds of milliseconds) may make the transmission to the affected client temporarily impossible. All of these wireless link properties combine to make the timely delivery of the video frames very challenging.

In this article we develop a high performance streaming protocol for the real–time delivery of prerecorded continuous media over wireless links. Our protocol is equally well suited for streaming in the downlink (base station to clients) direction, as well as the uplink (clients to base station) direction. We focus in the following discussion on the downlink direction (uplink streaming is discussed in Section V-C). 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. (These properties may be exploited to some extend when streaming live content with some prefetch delay budget; see Section V-B.) 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 wireless (and possibly mobile) clients within a wireless cell. The JSQ prefetch policy exploits rate adaptation techniques of wireless data packet protocols [7]. 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 may be constant.) The JSQ prefetch policy dynamically allocates more transmission capacity to wireless 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 article is organized as follows. In Section II we describe our architecture for the streaming of prerecorded continuous media in the downlink direction. Our focus is on multi–rate CDMA systems. In Section III we develop the components of our JSQ prefetching protocol. Importantly, we introduce channel probing; a mechanism that handles the location–dependent, time–varying, and bursty errors of wireless environments. In Section IV we evaluate our prefetching protocol both without and with channel probing through extensive simulations. In Section V we discuss several extensions of the prefetching protocol. We consider streaming with client interactions, such as pause/resume and temporal jumps. We also consider the streaming of live content, such as the audio and video feed from a sporting event. We outline how to use the prefetching protocol for prefetching in the uplink (clients to base station) direction. Moreover, we outline how to deploy the prefetching protocol in third generation CDMA systems as well as TDMA and FDMA systems. We also discuss the deployment of the prefetching protocol on top of physical layer error control schemes. We discuss the related work in Section VI and conclude in Section VII.

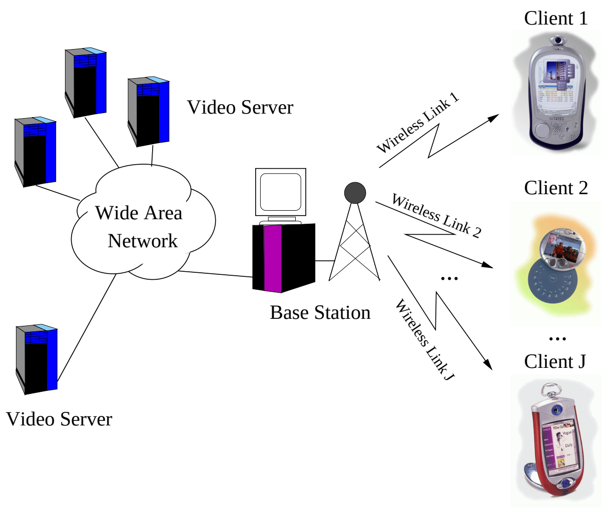

Fig. 1. Architecture: A central base station streams prerecorded continuous media to wireless (and possibly mobile) clients within a wireless cell.

II. Architecture

Figure 1 illustrates our architecture for continuous media streaming in the downlink direction. 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.

We initially consider a multi–code Code Division Multiple Access (CDMA) system. Such a system adapts rates for the synchronous transmissions in the forward direction by aggregating orthogonal code channels, that is, by varying the number of code channels used for transmissions to the individual clients. The second generation CDMA IS–95 (Rev. B) system [8] is an example of such a system; as is the third generation UMTS system [9] in TDD mode. 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 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 (an extension for live content is developed in Section V-B). To fix ideas we focus on video streams. 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–1 and MPEG–4) 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 video stream j 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 article 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 article. 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 [10]. 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. Toward 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 frame $\theta(j)$ is to be removed from the prefetch buffer of client j at a particular instant in time. The base station tracks the prefetch buffer contents by updating

$$b(j) \leftarrow [b(j) - x_{\theta(j)}(j)]^+ \tag{1}$$

and

$$p(j) \leftarrow [p(j) - t_{\theta(j)}(j)]^+, \tag{2}$$

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. To ensure proper synchronization between the clients and the base station in practical systems, each client sends periodically (every few slots, say) an update of its buffer contents (along with a regular acknowledgement) to the base station. These buffer updates are used at the base station to re–validate (and possibly correct) the variables $b(\cdot)$ and $p(\cdot)$.

III. Components of Prefetch Protocol

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 [11]. 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.

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 $p(j) + z(j)$ (ties are broken arbitrarily). The base station schedules one packet for transmission (by assigning a code to it) and increments $z(j^*)$:

$$z(j^*) \leftarrow z(j^*) + \frac{t_{\sigma(j^*)}(j^*)}{x_{\sigma(j^*)}(j^*)},$$

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 [10].) 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 [12].) 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. 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 $p(j) + z(j)$.

Note that the acknowledgment procedure outlined above 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. We adopt this procedure for simplicity and to be conservative. A refined approach would be to selectively account for the acknowledged packets. However, this would lead to ”gaps” in the prefetched video segments that take some effort to keep track of. We also note that because of the bursty error characteristics of wireless links typically either all or none of the packets sent to a client in a forward slot are lost. Also, since the acknowledgements can be sent with a large amount of forward error correction, the probability of losing an acknowledgement is typically small. Nevertheless, we point out the implications of the conservative acknowledgment procedure. Note that the buffer variables $b(j)$ and $p(j)$ maintained at the base station are the basis for the JSQ scheduling. It is therefore important that the buffer variables $b(j)$ and $p(j)$ correctly reflect the actual buffer contents at client j. Our conservative acknowledgment procedure clearly ensures that the actual buffer content at client j is always larger than or equal to the buffer variables $b(j)$ and $p(j)$. Note that only (1) sporadic packet losses (i.e., loss of a subset of the packets sent to a client in a slot), or (2) the loss of an acknowledgement (of a packet that was correctly received) can cause the actual buffer content to be larger than the corresponding buffer variables. As pointed out above these two events are rather unlikely, therefore the actual buffer content is typically not larger than the buffer variables. Also, if the buffer content is larger than the buffer variables, then typically by a small margin. Now, whenever the actual buffer content at a client j happens to be larger than the corresponding buffer variables $b(j)$ and $p(j)$, then the buffer content and the buffer variables are reconciled when either one of the following three events happens. (i) The packets that were considered lost by the base station (but actually received by the client) are re–transmitted (as part of the regular JSQ schedule) and their acknowledgments arrive at the base station. (The client acknowledges all successfully received packets, and then discards packets that are received in duplicate.) (ii) The frame (of which packets are considered lost at the base station) is consumed (before a re–transmission could take place). In this case the operations (1) and (2) set the buffer variables to zero. The client empties its prefetch buffer and decodes the part (if any) of the frame that it received. (iii) A buffer update message gives the base station the actual client buffer content. We finally note that in our performance evaluation in Section IV the losses are counted based on the buffer variables $p(j)$ and $b(j)$ at the base station. Thus, our performance results are on the conservative side.

We note that if $R(j) \geq S$ for all j, $j = 1, \ldots, J$, it is not necessary to signal the assigned codes to the clients as each client is capable of monitoring all of the S orthogonal channels. In case there are some clients with $R(j) < S$, the code assignment must be signaled to these clients, this could be done in a signalling time slot (right before the forward slot) on a common signalling channel, e.g., the Network Access and Connection Channel (NACCH) of the Integrated Broadband Mobile System (IBMS) [13].

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

$$r(j^*) \leq R(j^*) - 1, \tag{3}$$

and

$$b(j^*) \leq B(j^*) - 1. \tag{4}$$

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 (3) and (4) 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 (3) or (4) (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. *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 the previous section 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. 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 [14]. To see the need for the channel probing refinement consider a scenario where one of the clients experiences a persistent burst error on its wireless link to the base station. The burst error cuts the client off from the base station and the client continues video playback from its prefetched reserve. As its prefetched reserve decreases the JSQ prefetch policy allocates larger and larger transmission capacities to the affected client. The excessive transmission resources expended on the affected client, however, reduce the transmissions to the other clients in the wireless cell. As a result the prefetched reserves of all the other clients in the wireless cell are reduced. 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 [15]. 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. 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.

IV. 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 three different encodings of video streams: (i) video streams encoded at a Constant Bit Rate (CBR) and a constant frame rate, (ii) video streams encoded at a Variable Bit Rate (VBR) and a constant frame rate (e.g., MPEG–1 encodings), and (iii) video streams encoded at a variable frame rate and a variable bit rate (e.g., H.263 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 scenario we generated ten pseudo traces by scaling MPEG–1 traces obtained from the public domain [16], [17], [18] to an average rate of $\bar{r}$ = 64 kbps. The traces have a fixed frame rate of 25 frames/sec and are 40.000 frames long. The generated pseudo traces are highly bursty with peak–to–mean ratios in the range from 7 to 18; see [19] for details. For the H.263 scenario we generated ten H.263 encodings with an average rate of $\bar{r}$ = 64 kbps and one hour length each. The frame periods of the encodings are variable; they are multiples of the reference frame period of 40 msec. The H.263 encodings have peak–to–mean ratios in the range from 5 to 8; see [4] for details.

For each of the J ongoing streams in the wireless cell we randomly select one of the MPEG (H.263) 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 runtime of T seconds). (We chose the exponential distribution because it has been found to be a good model for stream lifetimes. We discuss the impact of the stream lifetime on the prefetch protocol performance in more detail in Section IV-B.) 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 randomly select 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 [20], [21]. 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”. (The two state model is a useful and accurate wireless channel characterization for the design of higher layer protocols [22], [23], [24]. It may be obtained from a more complex channel model, which may incorporate adaptive error control techniques (see Section V-E), using weak lumpability or stochastic bounding techniques [25].) We assume that all parallel code channels of a wireless link (to a particular client) are either in the good state or the bad state. The Gilbert–Elliot channel model is typically characterized by (1) the steady state probability of being in the good state, denoted by π_g, (π_b denotes the steady probability of being in the bad state), and (2) the average sojourn time in the bad channel state, denoted by τ_{bad}. For a flat fading channel the Gilbert–Elliot model parameters can be expressed analytically in terms of the Rayleigh fading parameters [26]. With ρ denoting the ratio of the Rayleigh fading envelope to the local root mean square level, the steady state probability of being in the good state is given by $\pi_g = e^{-\rho^2}$ (and $\pi_b = 1 - e^{-\rho^2}$). We conservatively assume a fade margin of 10 dB (i.e., ρ = – 10 dB) of the client's radio front end (even 20 dB may be assumed [27]). This gives the steady state probabilities π_g = 0.99 and π_b = 0.01, which are typical for burst–error wireless links [28]. These steady state probabilities are used throughout our simulations. The average sojourn time in the bad channel state, that is, the average time period for which the received signal is below a specified level, is given by $\tau_{\text{bad}} = (e^{\rho^2} - 1)/(\sqrt{2\pi}\rho f)$. Here, f denotes the maximum Doppler frequency given by $f = \nu/\lambda$, where ν denotes the speed of the wireless terminal and λ denotes the carrier wavelength. A UMTS system carrier wavelength of λ = 0.159 m (corresponding to a carrier frequency of 1.885 GHz) and a typical client speed of ν = 0.2 m/s suggest τ_{bad} = 32 msec. However, we chose a larger τ_{bad} of one second for most of our simulations. We did this for two reasons. First, to account for the detrimental effects of long term shadowing, which may persist for hundreds of milliseconds even up to seconds, as well as Rayleigh fading. Secondly, we observed in our simulations (see Section IV-B) that our prefetch protocol gives slightly larger client starvation probabilities for larger τ_{bad}. Thus, a larger τ_{bad} gives conservative performance results. We set the channel error probabilities such that all the packets sent to a client in a slot are 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.

For our quantitative evaluation of the JSQ prefetch protocol we 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.,

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

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 [29] and obtain confidence intervals on the client starvation probability P_{loss} using the method of batch means [30]. We run the rag replacements

PSfrag replacements

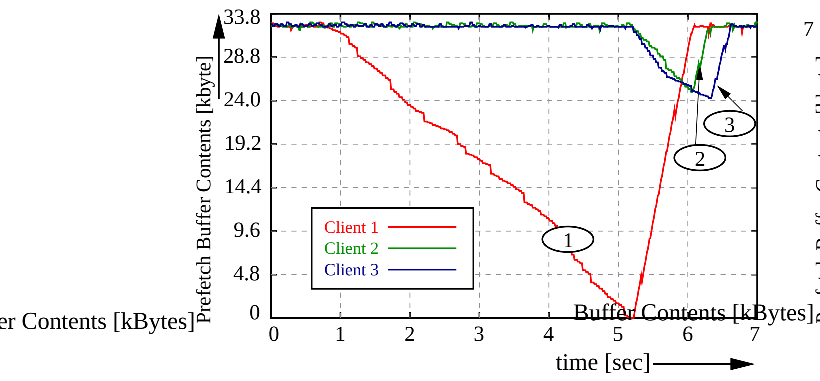

Fig. 2. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 starts over

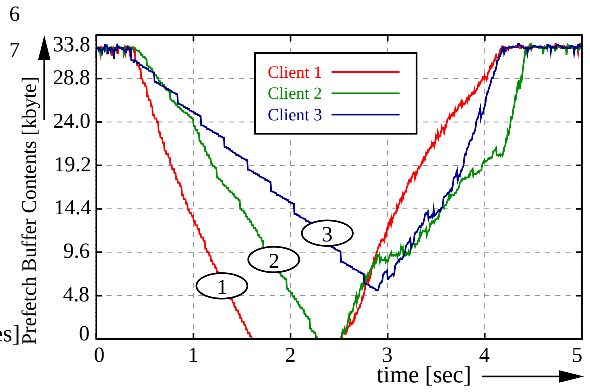

Fig. 3. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 experiences a bad channel.

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 streams is set to $T = 10$ minutes unless stated otherwise. Throughout, the base station has a total of $S = 15$ channels available for video streaming.

A. Simulation of Refined Prefetch Protocol without Channel Probing

Figures 2 and 3 show typical sample path plots from the simulations. In this experiment we 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 strives to balance the lengths of the prefetched video segments (in seconds of video runtime) 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. 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. For this reason we have introduced the channel probing refinement in Section III-C. In the next section we conduct a detailed quantitative evaluation of the JSQ prefetch protocol with channel probing.

B. *Simulation of Refined Prefetch Protocol with Channel Probing*

To evaluate the performance of the simple channel probing scheme introduced in Section III-C we compare it with an ideal scenario where the base station has perfect knowledge of the states of the wireless channels. In the perfect knowledge scenario the base station schedules packets only for clients in the good channel state. The base station does not schedule any packet (not even a probing packet) for clients experiencing a bad channel. (Note that in this perfect knowledge scenario no packets are lost due to a bad channel; however, packets that are sent to clients with a good channel are lost with probability P_l^g.)

Figure 4 shows the client starvation probability P_{loss} without channel probing, with channel probing, and with perfect knowledge of the channel state 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. For fast varying channels with an average sojourn time of τ_{bad} = 12 msec in the bad channel state, prefetching with our simple channel probing scheme gives a loss probability of $P_{\text{loss}} = 5.5 \cdot 10^{-4}$, while prefetching without channel probing gives $P_{\text{loss}} = 1.1 \cdot 10^{-3}$ (for J = 13). As τ_{bad} increases the client starvation probability for prefetching with channel probing increases only slightly; the gap between prefetching with channel probing and prefetching without channel probing, however, increases dramatically. For τ_{bad} = 350 msec and larger the client starvation probability of prefetching with channel probing is over one order of magnitude smaller than the client starvation probability of prefetching without channel probing. We also observe that the client rag replacements

PSfrag replacements

1.0

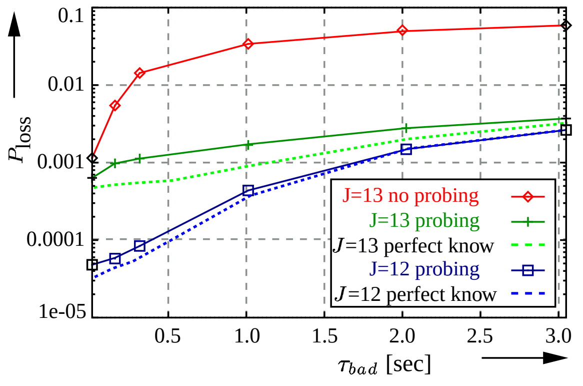

Fig. 4. Client starvation probability P_{loss} as a function of the average sojourn time in the "bad" channel state for prefetching of VBR video without channel probing, with channel probing, and with perfect knowledge of the channel state.

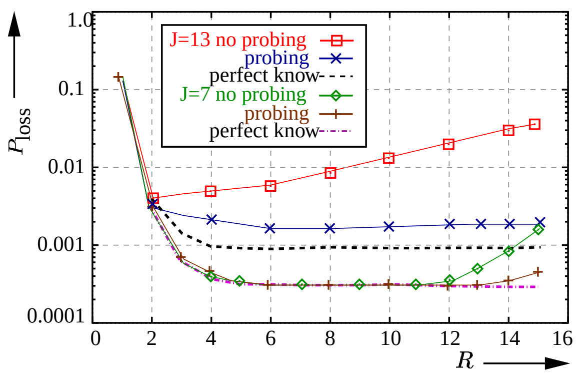

Fig. 5. Client starvation probability P_{loss} as a function of the maximum number of channels R per client for prefetching of VBR video without channel probing, with channel probing, and with perfect knowledge of the channel state.

starvation probabilities achieved by our simple channel probing scheme are only slightly above the client starvation probabilities achieved with perfect knowledge of the channel state. This indicates that more sophisticated channel probing schemes could achieve only small reductions of the client starvation probability. A more sophisticated channel probing scheme could probe with multiple packets per slot when the affected client has a small prefetched reserve and with one packet every kth slot, $k > 1$, for clients with a large prefetched reserve. We also note that reducing the slot length of the TDD would make channel probing more effective at the expense of increased overhead; inspired by the UMTS standard [9] we used a short fixed TDD slot length of 12 msec throughout. We set the average sojourn time in the "bad" channel state to one second for all the following experiments.

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 channel probing, with channel probing, and with perfect knowledge of the channel state. We observe from Figure 5 that for all three approaches, 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 and with perfect knowledge of the channel state, 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

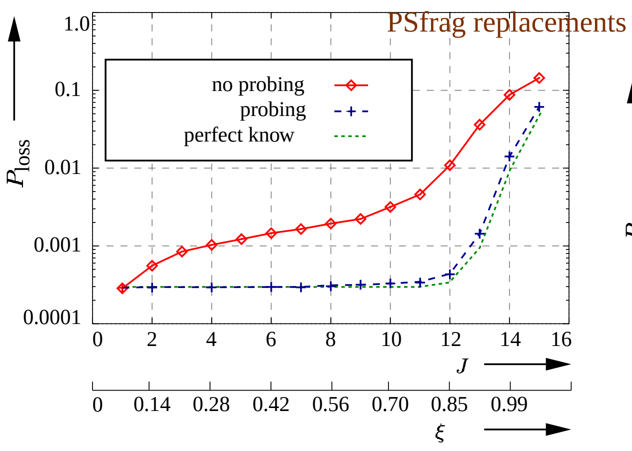

Fig. 6. Client starvation probability P_{loss} as a function of the bandwidth efficiency ξ (obtained by varying the number of clients J) for VBR video and prefetch buffer B = 32 kByte.

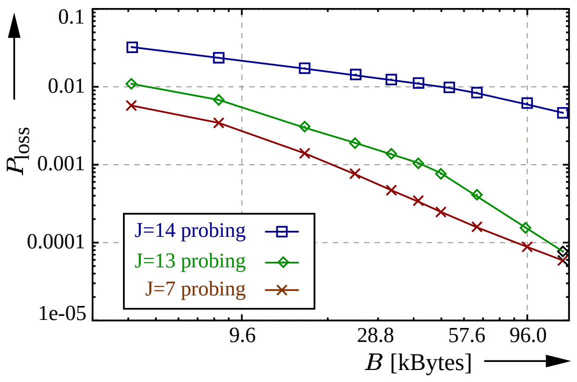

Fig. 7. Client starvation probability P_{loss} as a function of the client buffer capacity B for VBR video.

that can be assigned to an individual client, the larger 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 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 the simple channel probing scheme. The client starvation probability P_{loss} achieved with channel probing is (i) generally over one order of magnitude smaller than without channel probing, and (ii) only slightly larger than with perfect knowledge of the channel state. Throughout the remainder of this article we use prefetching 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

PSfrag replacements

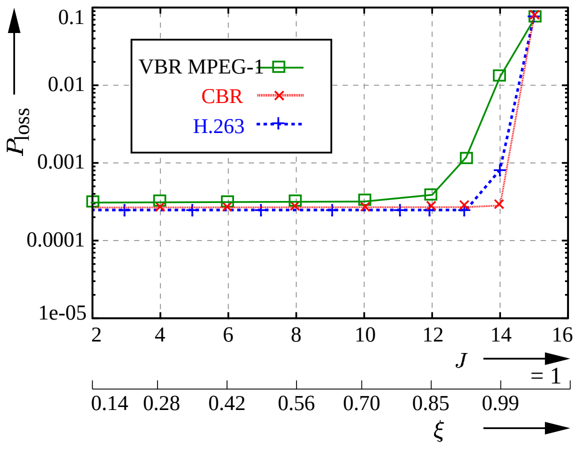

Fig. 8. Client starvation probability P_{loss} as a function of the bandwidth efficiency ξ (obtained by varying the number of clients J) for B = 32 kBytes for VBR MPEG–1, CBR, and H.263 video.

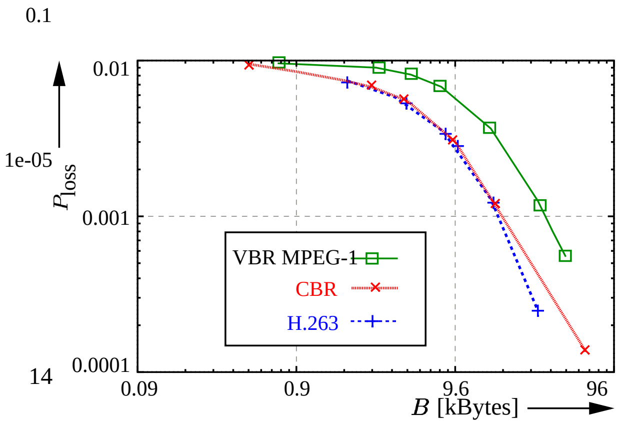

Fig. 9. Client starvation probability P_{loss} as a function of the client buffer capacity B for J = 13 streams of VBR MPEG–1, CBR, and H.263 video.

= 13 ongoing VBR streams the client starvation probability drops by over two orders of magnitude as the client buffers increase from 8 kBytes to 128 kBytes. (A buffer of 128 kBytes can hold on average 16 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 VBR MPEG–1 video, CBR video, and H.263 video. The prefetch buffer is fixed at B = 32 kBytes in this experiment. Figure 9 shows the client starvation probability P_{loss} as a function of the client buffer capacity B for the streaming of J = 13 streams of VBR MPEG–1 video, CBR video, and H.263 video. We observe from the plots that H.263 video gives generally smaller client starvation probabilities than VBR MPEG–1 video. This is primarily because H.263 video has for a given average bit rate, larger frame sizes $x_n(j)$ and correspondingly larger frame periods $t_n(j)$ than MPEG video [4]. For H.263 video the prefetch protocol has therefore more freedom in scheduling the frames’ packets, resulting in smaller client starvation probabilities. Moreover, the used H.263 traces are less variable than the VBR MPEG–1 traces. We also observe from the plots that CBR video gives smaller client starvation probabilities than the very variable VBR MPEG–1 video.

Table I gives the client starvation probability P_{loss} as a function of the average stream lifetime T for

TABLE I
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 II
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

the streaming of VBR MPEG–1 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 note that (given a fixed mean) the distribution of the stream lifetime has typically little impact on the performance of the prefetch protocol. It is important, however, that the starting times of the streams do not collude too frequently. This is because the JSQ policy allocates more transmission resources to a new stream (with an empty prefetch buffer) to quickly build up its prefetched reserve (see Figure 2 for an illustration where the new stream's reserve is build up in less than one second). If several streams start at the same time it takes longer to build up their reserves. The longer build–up period makes losses more likely. This is because streams without sufficient reserves are more vulnerable to playback starvation due to wireless link failures (or bursts in the video traffic). In our simulations we draw random stream lifetimes from an exponential distribution and start up a new stream immediately after an ongoing stream has terminated. With this approach the starting times of the stream collude only infrequently. We believe that this is an appropriate model for an on–demand streaming service.

We observed in our simulations that losses typically occur 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 II 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 II 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

TABLE III
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

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. EXTENSIONS OF PREFETCH PROTOCOL

A. Client Interactions

In this section we adapt our streaming protocol to allow for client interactions, such as pause/resume and temporal jumps. 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 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 III 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 ξ = 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.

B. *Prefetching for Live Streams*

Although we have developed our prefetching protocol primarily for the streaming of prerecorded continuous media, in this section we explore the prefetching of *live* (i.e., not prerecorded) continuous media. In particular, we focus on the transmission of the coverage of a live event, such as the video and audio feed from a conference, concert, or sporting event. In the case of a live video feed the delay budget from capturing a video frame to its display on the client's screen is typically on the order of hundreds of milliseconds. (One satellite hop, for instance, introduces a propagation delay of 240 msec.) We introduce an additional prefetch delay budget (on the order of a few hundred milliseconds) to allow for prefetching over the wireless link (i.e., the last hop). Specifically, let $P(j)$ denote the prefetch delay budget for stream j in seconds. With prefetching the beginning of the playback at the user is delayed by an additional $P(j)$. However, we believe that an increase of the start–up delay of a few hundred milliseconds is acceptable to most users. (We note that the situation is different for interactive communication, e.g., video conferencing, where the total delay budget is typically limited to 250 msec.)

When commencing the transmission of a live stream the base station immediately starts to transmit the stream's packets. The client starts to play out the stream $P(j)$ seconds after the stream transmission has commenced. The base station may support live streams and prerecorded streams at the same time. The base station schedules both the live streams and prerecorded streams according to the refined JSQ prefetch policy with channel probing. Packets are scheduled for the stream j^* with the smallest prefetched reserve $p(j^*)$, that satisfies (i) the client reception constraint (1) (which is set to one packet per slot if client j^* is in probing mode) and (ii) the client buffer constraint (2). In addition, for a live stream the base station checks whether the packet considered for transmission exceeds the prefetch delay budget. Specifically, when considering a packet for transmission to client j^* the base station verifies whether

$$p(j^*) \leq P(j^*) - \frac{t_{\sigma(j^*)}(j^*)}{x_{\sigma(j^*)}(j^*)}, \tag{5}$$

where $\sigma(j^*)$ is the frame (number) of stream j^* that is (partially) carried by the considered packet. This constraint ensures that the base station does not exceed the prefetch delay budget by scheduling a packet that has yet to be delivered from the live video source. The constraint (5) ensures that the base

TABLE IV
CLIENT STARVATION PROBABILITY P_{LOSS} AS A FUNCTION OF THE PREFETCH DELAY BUDGET $P(j)$ FOR THE STREAMING OF *live* VBR MPEG–1 VIDEO.

$P(j)$ [sec]	0.04	0.08	0.24	0.4	1.0	2.0	4.0	∞
P_{loss} [10^{-3}]	17.8	15.5	12.3	9.4	5.9	1.9	0.56	0.46

station does not prefetch more than $P(j)$ seconds ”into the future”.

In our simulation study we consider a scenario where all ongoing streams are *live* streams. (Note that our performance results are thus somewhat conservative compared to a more realistic scenario where the base station supports a mixture of live streams and prerecorded streams. This is because the base station is not constrained by the prefetch delay budget (5) when prefetching for the prerecorded streams.) We consider the streaming of live VBR MPEG–1 streams to J = 12 clients, each with a buffer capacity of B = 32 kBytes and support for R = 15 parallel channels. The average lifetime of the video streams is set to T = 10 minutes. The base station has a total of S = 15 channels available for video streaming; the bandwidth efficiency is fixed at ξ = 0.85. Table IV gives the client starvation probability P_{loss} as a function of the prefetch delay budget $P(j)$ (which is the same for all streams). In the considered scenario, prefetching with a prefetch delay budget of 400 msec (the typical delay requirement for voice over IP) gives a client starvation probability of less than 10^{-2} (a typical loss requirement for MPEG–4 video). The client starvation probability for prefetching of live content with a prefetch delay budget of four seconds is almost as small as for the streaming of prerecorded content without a prefetch delay budget constraint.

C. Uplink Streaming

In this section we outline the streaming of prerecorded continuous media from multiple clients to a central base station. We assume a multi–rate CDMA system with a Time Division Duplex (TDD) timing structure. For every uplink stream a prefetch buffer is allocated in the base station. The base station tracks the prefetch buffer contents and schedules the uplink packet transmissions according to the JSQ policy. The base station sends out the uplink transmission schedule in the forward (downlink) slot. In the subsequent backward (uplink) slot the clients send the scheduled packets to the base station; pseudo noise codes are used for these asynchronous transmissions. The base station processes the received packets, computes the next uplink transmission schedule, and sends it out in the following forward slot.

When packets are missing at the end of the backward slot (i.e., when the schedule or a packet was lost) the affected client is probed. While probing the affected client sends one (probing) packet per backward slot. The probing terminates when a probing packet is received by the base station.

D. Prefetching in third generation CDMA systems and TDMA systems

We have discussed 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 downlink streaming service and client j can receive (and process) $R(j)$ code channels in parallel. 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 [31]. 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 [9]. 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 article, 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) [32]. 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 [33]. 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 [34]). The prefetching protocol may be employed in a Frequency Division Multiple Access (FDMA) system with Time Division Duplex in analogous fashion.

E. Physical Layer Refinements

In this section we outline how our prefetch protocol may be used in conjunction with physical layer techniques designed to improve the throughput over wireless channels. Recall from Section III that the prefetching protocol distinguishes two states of the wireless channel: a ”good” state where packets are successfully received, decoded, and acknowledged, and a ”bad” state where packets (and/or acknowledgments) are lost (due to an excessive number of bit errors or complete shadowing). The prefetch protocol may run on top of adaptive error control schemes employed at the physical layer. These schemes may adapt the forward error correction [35] or transmission power [8] based on pilot tone or signal to noise and interference measurements. As long as these techniques are able to successfully return an acknowledgment for a transmitted packet, the prefetching protocol ”sees” a good channel and schedules packets according to the length (in runtime) of the prefetched reserve. When the physical layer techniques fail (due to severely deteriorated channel conditions or complete shadowing), the prefetching protocol switches to the probing mode.

An avenue for future research is to use interleaving schemes or turbo codes [36] in conjunction with the prefetching protocol. Interleaving or turbo codes could exploit the delay tolerance of clients with a large prefetched reserve to successfully decode packets under adverse channel conditions.

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.* [37] for a survey. Choi and Shin [38] 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 [39], [40], [41], [42]. These works adapt fair scheduling algorithms originally developed for wireline packet–switched networks to wireless environments. They address the key difference between scheduling and resource 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 [43], [22]. Krunz and Kim [44], [45] study the packet delay and loss distributions as well as the effective bandwidth of an on–off flow over a wireless link with automatic repeat request and forward error correction.

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*. Prerecorded continuous media, however, 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 [46] as well as Reisslein and Ross [47] 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. The proposed approaches fall into two main categories: non–collaborative prefetching and collaborative prefetching. Non–collaborative prefetching schemes [48], [49], [50] smooth (i.e., reduce the peak rate and rate variability of) an *individual* VBR video stream. The smoothing is achieved by transmitting the video traffic at a piecewise constant–rate transmission schedule. The piecewise constant–rate transmission schedule relies on prefetching of some parts of the prerecorded video (also referred to as work–ahead) into the client buffer. The non–collaborative prefetching schemes compute (typically off–line) a transmission schedule that is as smooth as possible while ensuring that the client buffer neither overflows nor underflows. The video streams are then transmitted according to the individually precomputed transmission schedules. The statistical multiplexing of the individually smoothed video streams is studied in [51], [52].

Collaborative prefetching schemes [53], [54], on the other hand, determine the transmission schedule of a video stream on–line as a function of *all* the other ongoing video streams. In particular, the collaborative prefetching schemes for wireline networks take the prefetch buffer contents at all the clients into consideration. We have built our prefetch protocol for wireless environments on the principle of collaborative prefetching for two main reasons. First, the time–varying, bursty, and location–dependent wireless link errors make it very difficult (if not impossible) to transmit according to a fixed transmission schedule that is pre–computed for an individual video stream by some non–collaborative prefetching scheme. Wireless systems experience situations where the shared transmission capacity is temporarily available only to a subset of the clients. At any given point in time there may be good transmission conditions on the wireless links to some clients while the transmission conditions are adverse on the wireless links to some other clients. It is therefore natural to build upon the principle of collaborative prefetching when designing a prefetching protocol for wireless environments. Secondly, it has been shown in the context of wireline networks that collaborative prefetching outperforms non–collaborative prefetching when multiple video streams share a single bottleneck link [47];

that is collaborative prefetching achieves higher average link utilizations and smaller client starvation probabilities. Wireless networks and wireline networks have fundamentally different characteristics. However, streaming from a central base station to wireless clients is similar to streaming over a wireline bottleneck link. In both cases multiple video streams share a common resource. In the wireless setting this common resource is the downlink transmission capacity of the base station.

Among the collaborative prefetching schemes for wireline environments is a prefetching scheme based on the Join–the–Shortest–Queue (JSQ) principle developed by Reisslein and Ross [53]. 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 article differs from the protocol in [53] in two major aspects. First, the protocol in [53] 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 [53] 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 article, 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 [55] 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 requirements of flows based on MAC packet deadlines and channel conditions. The Simultaneous MAC Packet Transmission (SMPT) scheme of Fitzek *et al.* [56] 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 article differs from the network/transport level schemes [55], [56] in several aspects. First, [55], [56] 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 [55], [56]; 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, [55], [56] do not take the characteristics of the application layer traffic into consideration; the scheme [55] operates on one MAC packet at a time and SMPT [56] operates on one TCP or UDP segment at a time. Our protocol in this article, 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.

We note in closing that Chaskar and Madhow [57] and Andrews *et al.* [58] study the sharing of the base station's downlink transmission capacity by multiple flows. Chaskar and Madhow [57] propose a link shaping scheme where the flows are assigned individual packet slots or entire rows in an interleaver block matrix. Andrews *et al.* [58] propose a Modified Largest Weighted Delay First (M–LWDF) scheme, which is designed to provide throughput and/or delay assurances. In the M–LWDF scheme the base station maintains a transmission queue for each ongoing downlink flow. Roughly speaking, packets are scheduled for the flow with (i) the largest transmission queue, and (ii) relatively good transmission conditions.

VII. Conclusion

We have developed a high performance prefetching protocol for the streaming of prerecorded continuous media in a cellular wireless system. 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 crucial 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 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 are also extending the prefetching protocol to scalable (layered) encoded video. With layered encoding the video is typically encoded into a base layer, which provides a basic video quality, and one (or more) enhancement layer(s), which improve the video quality. Layered encoded video has the decoding constraint that an enhancement layer can only be decoded if all lower layers are given. Layered encoded video makes it possible to provide different video qualities to clients with different decoding and display capabilities. Importantly, layered encoded video also allows for graceful degradation of the video quality. During periods of bad channel conditions the client may run out of the enhancement layer(s) and continue displaying the lower quality base layer video (provided a sufficiently long segment of the base layer has been prefetched). With layered encoded video there is a trade–off between prefetching (1) a shorter segment of the complete (base layer + enhancement layer(s)) stream, or (2) a longer base layer segment. We are exploring this trade–off and are developing policies that prefetch the enhancement layer(s) only if a minimum–length base layer segment has been prefetched.

Acknowledgment: We are grateful to Prof. Adam Wolisz for providing the environment that allowed us to pursue the work presented in this article.

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] Inktomi Inc., "Streaming media caching white paper," Technical Report, Inktomi Corporation, 1999.

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

[4] F. Fitzek and M. Reisslein, "MPEG–4 and H.263 video traces for network performance evaluation," *IEEE Network, accepted for publication*, 2001, Preprint and video traces available at `http://www.eas.asu.edu/˜mre`.

[5] G. Karlsson, "Asynchronous transfer of video," *IEEE Communications Magazine*, vol. 34, no. 8, pp. 106–113, Feb. 1996.

[6] H. Liu and M. El Zarki, "H.263 video transmission over wireless networks using hybrid ARQ," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 9, pp. 1775–1785, Dec. 1997.

[7] 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.

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

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

[10] 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.

[11] 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.

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

[13] M. Bronzel et al, "Integrated broadband mobile system (IBMS) featuring wireless ATM," in *Proceedings of ACTS Mobile Communication Summit*, Aalborg, Denmark, Oct. 1998.

[14] M. D. Yacoub, *Foundations of Mobile Radio Engineering*, CRC Press, 1993.

[15] M. Zorzi and R. R. Rao, "Error control and energy consumption in communications for nomadic computing," *IEEE Transactions on Computers*, vol. 46, no. 3, pp. 279–289, Mar. 1997.

[16] M. W. Garret, *Contributions toward Real-Time Services on Packet Networks*, Ph.D. thesis, Columbia University, May 1993.

[17] 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.

[18] 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, Feb. 1995.

[19] F. Fitzek and M. Reisslein, "A prefetching protocol for continuous media streaming in wireless environments (extended version)," Tech. Rep. TKN–00–05, Technical University Berlin, Dept. of Electrical Eng., Germany, June 2001, available at `http://www-tkn.ee.tu-berlin.de/˜fitzek` and `http://www.eas.asu.edu/˜mre`.

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

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

[22] 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.

[23] M. Zorzi, R. R. Rao, and L. B. Milstein, "On the accuracy of a first–order markovian model for data block transmission on fading channels," in *Proceedings of IEEE International Conference on Universal Personal Communications*, Nov. 1995, pp. 211–215.

[24] H. S. Wang and N. Moayeri, "Finite–state markov model — a useful model for radio communication channels," *IEEE Transactions on Vehicular Technology*, vol. 44, pp. 163–177, Feb. 1995.

[25] R. R. Rao, "Higher layer perspectives on modeling the wireless channel," in *Proceedings of IEEE ITW*, Killarney, Ireland, June 1998, pp. 137–138.

[26] G. L. Stüber, *Principles of Mobile Communications*, Kluwer Academic Publishers, second edition, 2001.

[27] C. Chien, M. B. Srivastava, R. Jain, P. Lettieri, V. Aggarwal, and R. Sternowski, "Adaptive radio for multimedia wireless links," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 793–813, May 1999.

[28] C. Hsu, A. Ortega, and M. Khansari, "Rate control for robust video transmission over burst–error wireless channels," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 756–773, May 1999.

[29] L. W. Schruben, "Detecting initialization bias in simulation output," *Operations Research*, vol. 30, pp. 569–590, 1982.

[30] G. S. Fishman, *Principles of Discrete Event Simulation*, Wiley, 1991.

[31] D. N. Knisely, S. Kumar, S. Laha, and S. D. Nanda, "Evolution of wireless data services: IS–95 to CDMA 2000," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 140–149, Oct. 1998.

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

[33] K. Balachandran, R. Ejzak, S. Nanda, S. Vitebskiy, and S. Seth, "GPRS–136: High–rate packet data service for north american TDMA digital cellular systems," *IEEE Personal Communications*, vol. 6, no. 3, pp. 34–47, June 1999.

[34] 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*, vol. 5, no. 5, pp. 53–64, Oct. 1998.

[35] H. Liu, H. Ma, M. ElZarki, and S. Gupta, "Error control schemes for networks: An overview," *Mobile Networks and Applications*, vol. 2, no. 2, pp. 167–182, Oct. 1997.

[36] T. Duman and M. Salehi, "New performance bounds for turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 6, pp. 717–723, June 1998.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

[44] J. G. Kim and M. Krunz, "Bandwidth allocation in wireless networks with guaranteed packet loss performance," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 337–349, June 2000.

[45] M. M. Krunz and J. G. Kim, "Fluid analysis of delay and packet discard performance for QoS support in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 2, pp. 384–395, Feb. 2001.

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

[47] 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.

[48] W. Feng and J. Rexford, "A comparison of bandwidth smoothing techiniques for the transmission of prerecorded compressed video," in *Proceedings of IEEE Infocom*, Kobe, Japan, Apr. 1997, pp. 58–67.

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

[50] Z. Jiang and L. Kleinrock, "A general optimal video smoothing algorithm," in *Proceedings of IEEE Infocom '98*, San Francisco, CA, Apr. 1998, pp. 676–684.

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

[52] Z. 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.

[53] 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.

[54] S. Bakiras and V. O. K. Li, “Smoothing and prefetching video from distributed servers,” in *Proceedings of IEEE International Conference on Networking Protocols (ICNP)*, Toronto, Canada, Oct. 1999, pp. 311–318.
[55] 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.
[56] F. H. P. Fitzek, B. Rathke, M. Schlager, 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.
[57] H. M. Chaskar and U. Madhow, “Statistical multiplexing and QoS provisioning for real–time traffic on wireless downlinks,” *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 2, pp. 347–354, Feb. 2001.
[58] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar, “Providing quality of service over a shared wireless link,” *IEEE Communications Magazine*, vol. 39, no. 2, pp. 150–154, Feb. 2001.

Frank Fitzek is working towards his Ph.D. in Electrical Engineering in the Telecommunication Networks Group at the Technical University Berlin, Germany. He received his Dipl.–Ing. degree in electrical engineering for the University of Technology — Rheinisch–Westfälisch Technische Hochschule (RWTH) — Aachen, Germany, in 1997. His research interests are in the areas of multimedia streaming over wireless links and Quality of Service support in wireless CDMA systems.

Martin Reisslein is an Assistant Professor in the Department of Electrical Engineering at Arizona State University, Tempe. He is affiliated with ASU’s Telecommunications Research Center. 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. While in Berlin he was teaching courses on performance evaluation and computer networking at the Technical University Berlin. He 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). His research interests are in the areas of Internet Quality of Service, wireless networking, and optical networking. He is particularly interested in traffic management for multimedia services with statistical Quality of Service in the Internet and wireless communication systems.