Optimal Coding Rate Control for Scalable Streaming Media

Cheng Huang[1*], Philip A. Chou[2], Anders Klemets[2]
[1] Department of Computer Science and Engineering,
Washington University in St. Louis, MO, 63130
[2] Microsoft Corporation, One Microsoft Way, Redmond, WA, 98052
[1]cheng@cse.wustl.edu, [2]{pachou, anderskl}@microsoft.com

Abstract— **Perhaps the major technical problem in streaming media on demand over the Internet is the need to adapt to changing network conditions. In this paper, we investigate the problem of** *coding rate control*, **or equivalently quality adaptation, in response to changing network conditions such as the onset of congestion. Using the theory of optimal linear quadratic control, we design an efficient online rate control algorithm. Extensive analytical and experimental results show that three goals are achieved: fast startup (about 1 s delay without bursting), continuous playback in the face of severe congestion, and maximal quality and smoothness over the entire streaming session. We argue that our algorithm complements any transport protocol, and we demonstrate that it works effectively with both TCP and TFRC transport protocols.**

I. Introduction

Perhaps the major technical problem in streaming media on demand over the Internet is the need to adapt to changing network conditions. As competing communication processes begin and end, the available bandwidth, packet loss and packet delay all fluctuate. Network outages lasting many seconds can and do occur. Resource reservation and quality of service support can help, but even they cannot guarantee that network resources will be stable. If the network path contains a wireless link, for example, its capacity may be occasionally reduced by interference. Thus it is necessary for commercial-grade streaming media systems to be robust to hostile network conditions. Moreover, such robustness cannot be achieved solely by aggressive (nonreactive) transmission. Even constant bit rate transmission with retransmissions for every packet loss cannot achieve a throughput higher than the channel capacity. Some degree of adaptivity to the network is therefore required.

End users expect that a good streaming media system will exhibit the following behavior: content played back on demand will start with low delay; once started, it will play back continuously (without stalling) unless interrupted by the user; and it will play back with the highest possible quality given the average communication bandwidth available. To meet these expectations in the face of changing network conditions, buffering of the content at the client before decoding and playback is required.

Buffering at the client serves several distinct but simultaneous purposes. First, it allows the client to compensate for short-term variations in packet transmission delay (i.e., "jitter"). Second, it gives the client time to perform packet loss recovery if needed. Third, it allows the client to continue playing back the content during lapses in network bandwidth. And finally, it allows the content to be coded with variable bit rate, which can dramatically improve overall quality.[1] By controlling the size of the client buffer over time it is possible for the client to meet the above mentioned user expectations. If the buffer is initially small, it allows a low startup delay. If the buffer never underflows, it allows continuous playback. If the buffer is eventually large, it allows eventual robustness as well as high, nearly constant quality. Thus, client buffer management is a key element affecting the performance of streaming media systems.

The size of the client buffer can be expressed as the number of seconds of content in the buffer, called the buffer *duration*. The buffer duration tends to increase as content enters the buffer and tends to decrease as content leaves the buffer. Content leaves the buffer when it is played out, at a rate of ν seconds of content per second of real time, where ν is the *playback speed* (typically 1 for normal playback, but possibly more than 1 for high speed playback or less than 1 for low speed playback). Content enters the buffer when it arrives at the client over the network, at a rate of r_a/r_c seconds of content per second of real time, where r_a is the *arrival rate*, or average number of bits that arrive at the client per second of real time, and r_c is the *coding rate*, or average number of bits needed to encode one second of content. Thus the buffer duration can be increased by increasing r_a, decreasing r_c, and/or decreasing ν (and vice versa for decreasing the buffer duration). Although the buffer duration can be momentarily controlled by changing r_a (cf. "Fast Start" in Windows Media 9 [1]) or changing ν (cf. "Adaptive Media Playout (AMP)" in [2]), these quantities are generally not possible to control freely for long periods of time. The arrival rate r_a on average is determined by the network capacity, while the playback speed ν on average is determined by user preference. Thus if the network capacity drops dramatically for a sustained period, reducing the coding rate r_c is the only appropriate

*Supported in part by NSF Grants CCR-TC-0209042 and ANI-0322615

[1]Note that even so-called constant bit rate (CBR) coded content is actually coded with variable bit rate within the constraints of a decoding buffer of a given size. The larger the decoding buffer size, the better the quality. The required decoding buffering is part of the larger client buffer.

way to prevent a *rebuffering event* in which playback stops ($\nu = 0$) while the buffer refills.

Thus, adaptivity to changing network conditions requires not only a buffer, but also some means to adjust the coding rate r_c of the content. In this paper we assume that this can be done with fine grained scalable (FGS) coding. A companion paper [3] deals with multi bit rate (MBR) coding, which is more prevalant in today's commercial streaming media systems [4], [1].

Our work focuses on the problem of *coding rate control*, that is, dynamically adjusting the coding rate of the content to control the buffer duration. Outside the scope of our work is the problem of transmission rate control. The *transmission rate* r_x is the rate at which the sender application injects bits into the transport layer and is equal to the arrival rate r_a on average if the transport is lossless. By *transmission rate control* we mean congestion control as well as any other mechanisms affecting the transmission rate such as bursting, tracking the transmission rate to the available bandwidth, and so on. Thus we control the buffer duration by adjusting the coding rate r_c at which bits leave the buffer, while letting the the arrival rate r_a at which bits enter the buffer be determined by other means.

In addition to factoring the problem of network adaptation into transmission rate control and coding rate control, the novelty of our approach lies in the following two aspects. First, we formulate the problem of coding rate control as a standard problem in linear quadratic optimal control, in which the client buffer duration is controlled as closely as possible to a target level while keeping the coding rate (and hence the quality) as constant as possible. To our knowledge this is the first use of optimal control theory for client buffer management. Second, we explicitly take into consideration, using a leaky bucket model, the natural variation in the instantaneous coding rate that occurs for a given average coding rate. We incorporate the leaky bucket model into the control loop so that the changes in buffer duration due to natural variation in the instantaneous coding rate are not mistaken for changes in buffer duration due to network congestion. To our knowledge this is also the first use of a leaky bucket to model source coding rate constraints during client buffer management beyond the initial startup delay.[2]

II. Problem Formulation

A. Temporal Coordinate Systems

It will pay to distinguish between the temporal coordinate systems, or clocks, used to express time. In this paper, *media time* τ refers to the clock running on the device used to capture and timestamp the original content, while *client time* t refers to the clock running on the client used to play back the content. The conversion from media time to client time can be expressed

$$t = t_0 + \frac{\tau - \tau_0}{\nu}, \tag{1}$$

where t_0 and τ_0 represent the time of a common initial event (such as the playback of frame 0), and ν is the *playback speed*.

[2]Ribas, Chou, and Regunathan use a leaky bucket to model source coding rate constraints to reduce initial startup delay [5], while Hsu, Ortega and Reibman use a leaky bucket to model transmission rate contraints [6].

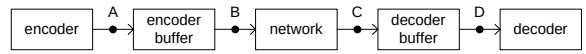


Fig. 1. Communication pipeline.

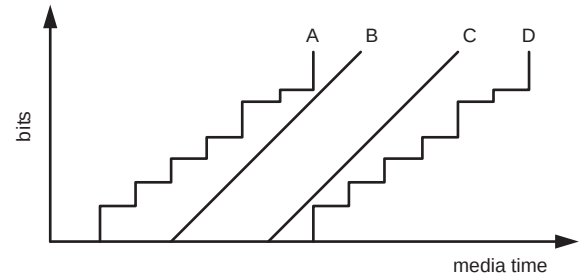


Fig. 2. Schedules at which bits in the coded bit stream pass the points A, B, C, and D in the communication pipeline.

B. Leaky Bucket Model

For the moment we revert to a scenario in which both the encoder and the decoder run in real time over an isochronous communication channel. In this case, to match the instantaneous coding rate to the instantaneous channel rate, an *encoder buffer* is required between the encoder and the channel and a *decoder buffer* is required between the channel and the decoder, as illustrated in Figure 1. A *schedule* is the sequence of times at which successive bits in the coded bit stream pass a given point in the communication pipeline. Figure 2 illustrates the schedules of bits passing the points A, B, C, and D in Figure 1. Schedule A is the schedule at which captured frames are instantaneously encoded and put into the encoder buffer. This schedule is a staircase in which the nth step rises by $b(n)$ bits at time $\tau(n)$, where $\tau(n)$ is the time at which frame n is encoded, and $b(n)$ is the number of bits in the resulting encoding. Schedules B and C are the schedules at which bits respectively enter and leave the communication channel. The slope of these schedules is R bits per second, where R is the communication rate of the channel. Schedule D is the schedule at which frames are removed from the decoder buffer and instantaneously decoded for presentation. Note that Schedule D is simply a shift of Schedule A. Note also that Schedule B is a lower bound to Schedule A, while Schedule C is an upper bound to Schedule D. Indeed, the gap between Schedules A and B represents, at any point in time, the size in bits of the encoder buffer, while the gap between Schedules C and D likewise represents the size of the decoder buffer. The encoder and decoder buffer sizes are complementary.

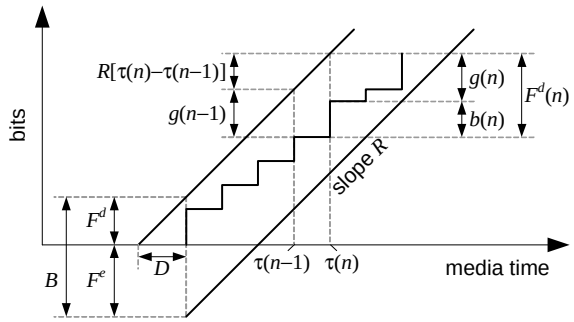


Fig. 3. Buffer tube containing a coding schedule.

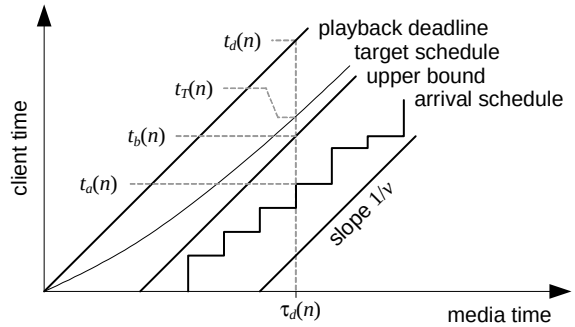


Fig. 4. Arrival schedule and its upper bound in client time. The upper bound is controlled to the target schedule, which is increasingly in advance of the playback deadline to provide greater robustness over time.

Thus the coding schedule (either A or D) can be contained within a *buffer tube*, as illustrated in Figure 3, having slope R, height B, and initial offset F^d from the top of the tube (or equivalently initial offset $F^e = B - F^d$ from the bottom of the tube). It can be seen that $D = F^d/R$ is the *startup delay* between the time that the first bit arrives at the receiver and the first frame is decoded. Thus it is of interest to minimize F^d for a given R.

A *leaky bucket* is a metaphor for the encoder buffer. The encoder dumps $b(n)$ bits into the leaky bucket at time $\tau(n)$, and the bits leak out at rate R. A leaky bucket with leak rate R, bucket size B, and initial bucket fullness F^e thus corresponds to a straight buffer tube bounding the coding schedule as in Figure 3. Each stream in the media file has a coding schedule; thus each stream corresponds to a straight buffer tube with slope R equal to the average coding rate r_c of the stream.

In the sequel we will need to consider the gap $g(n)$ at frame n between the buffer tube *upper bound* and the coding schedule, as depicted in Figure 3. Note that the decoder buffer fullness before frame n is put into the bucket can be expressed

$$F^d(n) = b(n) + g(n) = g(n-1) + \frac{r_c(n)}{f(n)}, \qquad (2)$$

where $f(n) = 1/[\tau(n) - \tau(n-1)]$ is the instantaneous frame rate at frame n, and $r_c(n)$ is the coding rate of the buffer tube, now taking into account that different frames may lie in different buffer tubes with different coding rates as coding rate control is applied and streams are switched.

C. Rate Control Model

Assume for the moment that bits arrive at the client at a constant rate r_a. Then frame n (having size $b(n)$) arrives at the client $b(n)/r_a$ seconds after frame $n-1$. Indeed, the index of a bit is proportional to its arrival time. Dividing the vertical scale of the schedules in Figure 3 by r_a, we obtain the schedules in terms of client time, rather than bits, as shown in Figure 4. The coding schedule divided by r_a becomes the *arrival schedule*, which provides for each n the time $t_a(n)$ of arrival of frame n at the client. The buffer tube upper bound (in bits) divided by r_a becomes the buffer tube upper bound (in time), which provides for each n the time $t_b(n)$ by which frame n is guaranteed to arrive. In the same plot we show the *playback deadline*, which is the time $t_d(n)$ at which frame n is scheduled to be played (after instantaneous decoding). Thus the gap between a frame's arrival time and its playback deadline is the client buffer duration at the time of the frame arrival. This must be non-negative to allow continuous playback.

In reality the arrival rate is not constant. If $t_a(n-1)$ and $t_a(n)$ are the arrival times of frames n and $n-1$ respectively, then we may define

$$r_a(n) = \frac{b(n)}{t_a(n) - t_a(n-1)} \qquad (3)$$

to be the *instantaneous arrival rate* at frame n. In practice we estimate the average arrival rate at frame n by an exponentially weighted moving average $\tilde{r}_a(n)$ of previous values of $r_a(n)$. Hence using (3) we may express the arrival time of frame n in terms of the arrival time of frame $n-1$ as

$$t_a(n) = t_a(n-1) + \frac{b(n)}{r_a(n)} \qquad (4)$$

$$= t_a(n-1) + \frac{b(n)}{\tilde{r}_a(n)} + v(n), \qquad (5)$$

where the $v(n)$ term is an error term that captures the effect of using the slowly moving average $\tilde{r}_a(n)$ instead of the instantaneous arrival rate $r_a(n)$. From (2), however, we have

$$b(n) = \frac{r_c(n)}{f(n)} + g(n-1) - g(n), \qquad (6)$$

whence (substituting (6) into (5)) we have

$$t_a(n) = t_a(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n)}{\tilde{r}_a(n)} + v(n). \qquad (7)$$

Now defining the buffer tube upper bound (in time) of frame n as

$$t_b(n) = t_a(n) + \frac{g(n)}{\tilde{r}_a(n)}, \qquad (8)$$

so that

$$t_b(n) - t_b(n-1) = t_a(n) - t_a(n-1) + \frac{g(n)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)}, \qquad (9)$$

we obtain the following update equation:

$$t_b(n) = t_b(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + w(n-1), \qquad (10)$$

where

$$w(n-1) = \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)} + v(n) \qquad (11)$$

is again an error term that captures variations around a locally constant arrival rate.

Using (8), the client can compute $t_b(n-1)$ from the measured arrival time $t_a(n-1)$, the estimated arrival rate $\tilde{r}_a(n-1)$, and $g(n-1)$ (which can be transmitted to the client along with the data in frame $n-1$ or computed at the client from $g(n-2)$ and $\hat{r}_c(n-1)$). Then using (10), the client can control the coding rate $r_c(n)$ so that $t_b(n)$ reaches a desired value, assuming the frame rate and arrival rate remain roughly constant. From this perspective, (10) can be regarded as the state transition equation of a feedback control system and it is thus possible to use a control-theoretic approach to regulate the coding rate.

D. Control Objective

With the state transition equation defined in (10), uninterrupted playback can be achieved by regulating the coding rate so that the client buffer does not underflow. To introduce a margin of safety that increases over time, we introduce a *target schedule*, illustrated in Figure 4, whose distance from the playback deadline grows slowly over time. By regulating the coding rate, we attempt to control the buffer tube upper bound so that it tracks the target schedule. If the buffer tube upper bound is close to the target schedule, then the arrival times of all frames will certainly be earlier than their playback deadlines and thus uninterrupted playback will be ensured. Note that controlling the actual arrival times (rather than their upper bounds) to the target would result in an approximately constant number of bits per frame, which would in turn result in very poor quality overall. By taking the leaky bucket model into account, we are able to establish a control that allows the instantaneous coding rate to fluctuate naturally according to the encoding complexity of the content, within previously established bounds for a given average coding rate.

Although controlling the upper bound to the target schedule is our primary goal, we also wish to minimize quality variations due to large or frequent changes to the coding rate. This can be achieved by introducing into the cost function a penalty for relative coding rate differences.

Letting $t_T(n)$ denote the target for frame n, we use the following cost function to reflect both of our concerns:

$$I = \sum_{n=0}^{N} \left(\left(t_b(n) - t_T(n)\right)^2 + \sigma \left(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a(n)} \right)^2 \right), \qquad (12)$$

where the first term penalizes the deviation of the buffer tube upper bound from the target schedule and the second term penalizes the relative coding rate difference between successive frames. N is the control window size and σ is a Lagrange multiplier or weighting parameter to balance the two terms.

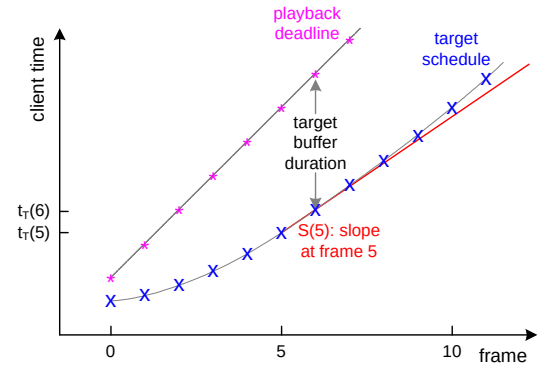


Fig. 5. Target schedule design.

E. Target Schedule Design

Figure 5 shows an illustrative target schedule. The gap between the playback deadline and the target schedule is the desired client buffer duration (in client time). If the gap is small at the beginning of streaming, then it allows a small startup delay, while if the gap grows slowly over time, it gradually increases the receiver's ability to counter jitter, delays, and throughput changes.

We choose the target schedule t_T so that the client buffer duration grows logarithmically over time. Specifically, if t_d is the playback deadline, then for each t_d greater than some start time t_{d0},

$$t_T = t_d - \frac{b}{a} \ln(a(t_d - t_{d0}) + 1), \qquad (13)$$

where $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$ by (1). Setting $b = 0.5$ implies that the client buffer duration will grow initially at two times real time. Further setting $a = 0.15$ implies that the buffer duration will be 7.68 seconds after 1 minute, 15.04 seconds after 10 minutes, and 22.68 seconds after 100 minutes.

III. Optimal Control Solution

Recall from (10) the fundamental state transition equation, which describes the evolution of the buffer tube upper bound $t_b(n)$ in terms of the coding rate $r_c(n)$:

$$t_b(n+1) = t_b(n) + \frac{r_c(n+1)}{f\tilde{r}_a} + w(n). \qquad (14)$$

Here we now assume that the frame rate f and the average arrival rate $\tilde{r}_a$ are relatively constant. Deviations from this assumption are captured by $w(n)$.

We wish to control the upper bound by adjusting the coding rate. As each frame arrives at the client, a feedback loop can send a message to the server to adjust the coding rate. Note, however, that by the time frame n arrives completely at the client, frame $n+1$ has already started streaming from the server. Thus the coding rate $r_c(n+1)$

for frame $n+1$ must already be determined by time $t_a(n)$. Indeed, at time $t_a(n)$, frame $n+2$ is the earliest frame for which the controller can determine the coding rate. Hence at time $t_a(n)$, the controller's job must be to choose $r_c(n+2)$. We must explicitly account for this one-frame delay in our feedback loop.

For simplicity, we linearize the target schedule around the time that frame n arrives. The linearization is equivalent to using a line tangent to the original target schedule at a particular point as an approximate target schedule. Thus we have

$$t_T(n+1) - 2t_T(n) + t_T(n-1) = 0. \tag{15}$$

Rather than directly control the evolution of the upper bound, which grows without bound, for the purposes of stability we use an error space formulation. By defining the error

$$e(n) = t_b(n) - t_T(n), \tag{16}$$

we obtain

$$\begin{aligned} & e(n+1) - e(n) \\ & = (t_b(n+1) - t_T(n+1)) - (t_b(n) - t_T(n)) && (17) \\ & = (t_b(n+1) - t_b(n)) - (t_T(n+1) - t_T(n)) && (18) \\ & = \frac{r_c(n+1)}{f\tilde{r}_a} - (t_T(n+1) - t_T(n)) + w(n), && (19) \end{aligned}$$

from which we obtain in turn

$$\begin{aligned} & (e(n+1) - e(n)) - (e(n) - e(n-1)) \\ & = [r_c(n+1) - r_c(n)]/f\tilde{r}_a \\ & \quad -(t_T(n+1) - 2t_T(n) + t_T(n-1)) \\ & \quad +(w(n) - w(n-1)) && (20) \\ & = \frac{r_c(n+1) - r_c(n)}{f\tilde{r}_a} + (w(n) - w(n-1)). && (21) \end{aligned}$$

We next define the control input

$$u(n) = \frac{r_c(n+2) - \hat{r}_c(n+1)}{\tilde{r}_a}, \tag{22}$$

where $\hat{r}_c(n+1)$ is a possibly quantized version of $r_c(n+1)$ (as defined in Section IV-A) and we define the disturbance

$$d(n) = \frac{\hat{r}_c(n) - r_c(n)}{f\tilde{r}_a} + w(n) - w(n-1). \tag{23}$$

Then (21) can be rewritten

$$e(n+1) = 2e(n) - e(n-1) + \frac{u(n-1)}{f} + d(n). \tag{24}$$

Therefore, defining the error vector

$$\mathbf{e}(n) = \begin{bmatrix} e(n) \\ e(n-1) \\ u(n-1) \end{bmatrix} = \begin{bmatrix} t_b(n) \\ t_b(n-1) \\ \frac{r_c(n+1)}{\tilde{r}_a} \end{bmatrix} - \begin{bmatrix} t_T(n) \\ t_T(n-1) \\ \frac{\hat{r}_c(n)}{\tilde{r}_a} \end{bmatrix}, \tag{25}$$

the error space representation of the system can be expressed

$$\mathbf{e}(n+1) = \begin{bmatrix} 2 & -1 & \frac{1}{f} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e}(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} d(n), \tag{26}$$

or $\mathbf{e}(n+1) = \Phi\mathbf{e}(n) + \Gamma u(n) + \Gamma_d d(n)$ for appropriate matrices Φ, Γ and Γ_d.

Assuming the disturbance $d(n)$ is a pure white noise, and assuming *perfect state measurement* (i.e., we can measure all components of $\mathbf{e}(n)$ without using an estimator), the disturbance $d(n)$ does *not* affect the controller design. Thus we can use a linear controller represented by

$$u(n) = -G\mathbf{e}(n), \tag{27}$$

where G is a *feedback gain*. By the time frame n is completely received, all elements of $\mathbf{e}(n)$ are available at the client and $u(n)$ can thus be computed. The ideal coding rate for frame $n+2$ can then be computed as

$$r_c(n+2) = \hat{r}_c(n+1) - G\mathbf{e}(n)\tilde{r}_a. \tag{28}$$

Finding the optimal linear controller amounts to finding the feedback gain G^* that minimizes the quadratic cost function defined in Section II-D. Before continuing with the design, we first check the system *controllability matrix* $\mathcal{C}$,

$$\mathcal{C} = \begin{bmatrix} \Gamma & \Phi\Gamma & \Phi^2\Gamma \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{f} & \frac{2}{f} \\ 0 & 0 & \frac{1}{f} \\ 1 & 0 & 0 \end{bmatrix}, \tag{29}$$

which has full rank for any frame rate f. Thus, the system is *completely controllable* and the state $\mathbf{e}(n)$ can be regulated to any desirable value. Now recall that the cost function defined in Section II-D is

$$\begin{aligned} I &= \sum_{n=0}^{N} \Big\{ \Big(t_b(n) - t_T(n)\Big)^2 + \sigma\Big(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a}\Big)^2 \Big\} && (30) \\ &= \sum_{n=0}^{N} \Big\{ \mathbf{e}(n)^T Q \mathbf{e}(n) + u(n-1)^T R u(n-1) \Big\}, && (31) \end{aligned}$$

where $Q = C^T C$ (with $C = [1\ 0\ 0]$) and $R = \sigma$. Then, the original control problem of tracking the target schedule while smoothing the coding rate fluctuations (i.e., minimizing the cost function I) is converted to a standard regulator problem in the error space. Letting $N \to \infty$, the infinite horizon optimal control problem can be solved by applying the results in [7, Section 3.3] to obtain an optimal regulator in two steps: 1) solving, to get S, the *discrete algebraic Riccati equation* (DARE)

$$S = \Phi^T\{S - S\Gamma[\Gamma^T S\Gamma + R]^{-1}\Gamma S\}\Phi + Q, \tag{32}$$

and 2) computing the optimal feedback gain

$$G^* = [\Gamma^T S\Gamma + R]^{-1}\Gamma^T S\Phi. \tag{33}$$

The existence and uniqueness of S (and in turn of G^*) is guaranteed when Q is nonnegative definite and R is positive definite, which is straightforward to verify in our case.

To compute the optimal regulator, it is necessary to choose a value for σ in (12) or (30)-(31). This can be done by following the following four steps: 1) pick a σ value to balance $\mathbf{e}(n)$ and $u(n)$; 2) compute the optimal feedback gain; 3) plot the closed-loop root locus (to check

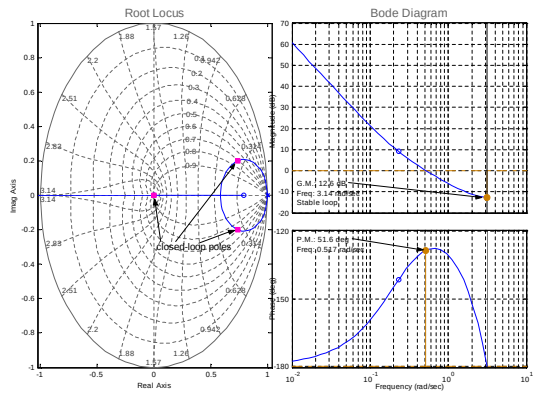


Fig. 6. Root locus and Bode diagram.

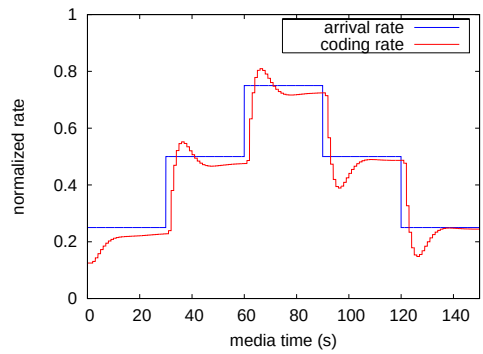
stability) and bode diagram (to check robustness); and 4) perform time domain simulations to verify transient response. Several iterations may be needed to determine a suitable σ value.

Following the above steps in this paper we select $\sigma = 50$. The corresponding optimal feedback control gain is then $G^* = [0.6307 \quad -0.5225 \quad 0.5225]$, for which the closed-loop system has poles at $0.7387 + 0.1999i$, $0.7387 - 0.1999i$ and 0, which are all inside the unit circle. Therefore, the closed-loop system is asymptotically stable. Figure 6 shows the closed-loop root locus and the bode diagram with the optimal feedback. We can again verify the stability of the closed-loop system since all poles are inside the unit circle. Also, the system has a *gain margin* (GM) of 12.60 dB and a *phase margin* (PM) of 51.59 degrees. The GM and PM are usually good indicators of system robustness. In our case, the PM is much larger than 30 degrees, which is often judged as the lowest adequate value [8, Section 6.4]. And this PM is close to 60 degrees, the best PM an optimal controller could achieve if continuous time feedback control was allowed. Therefore, the system achieves good robustness. Finally, Figure 7 provides the time response simulation results, which show good tracking properties with a fairly stable coding rate.

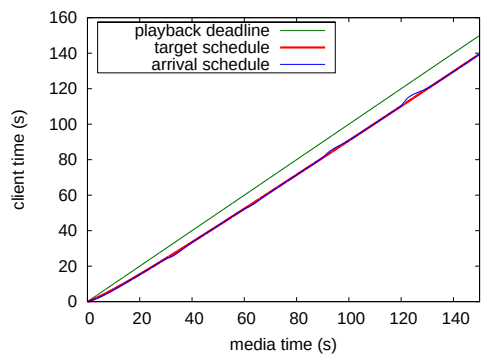
IV. Practical Issues

A. Choosing a Stream Given a Coding Rate

When the client requests a coding rate $r_c(n)$, the server complies by choosing a stream (or substream of a scalable stream) having coding rate $\hat{r}_c(n)$ approximately equal to $r_c(n)$. There are several reasons that $\hat{r}_c(n)$ may differ from $r_c(n)$. The first reason is that there are only a finite number of streams (or substreams) in the media file, even if fine grain scalable coding is used. Thus there may be no stream in the media file with average coding rate exactly equal to $r_c(n)$. The second reason is that, even if there is a stream in the media file with average coding rate exactly equal to $r_c(n)$, the buffer tube for the stream may be too large to allow switching to the stream without risk of client buffer



(a) rate vs. time



(b) schedule vs. time

Fig. 7. Time response simulation.

underflow. In fact, whenever the stream switches, there is generally a discontinuity in the upper bound, which may be either positive or negative. A positive shift in the upper bound is illustrated in Figure 8, which, if large, could cause the client buffer to underflow either immediately or eventually.

Thus the server must choose a stream that causes the upper bound to shift up no more than some amount $\Delta g^{\max}(n-1)$ supplied to it by the client. The client supplies $\Delta g^{\max}(n-1)$ to the server in its feedback along with $r_c(n)$, shortly after client time $t_a(n-2)$ (after frame $n-1$ has already begun streaming). Upon receiving the feedback, the server selects a stream with coding rate $\hat{r}_c(n)$ as high as possible such that $\hat{r}_c(n) \leq r_c(n)$ and, if $\hat{r}_c(n) > \hat{r}_c(n-1)$ (i.e., if it is a switch up in rate), then $g^{new}(n-1) - g^{old}(n-1) \leq \Delta g^{\max}(n-1)$, where $g^{new}(n-1)$ and $g^{old}(n-1)$ are illustrated in Figure 8. The constraint given by $\Delta g^{\max}(n-1)$ is not applied if it is a switch down in rate.

The client chooses $\Delta g^{\max}(n-1)$ to limit what the upper bound would be at time $t_a(n-1)$ if the new coding rate were in effect. Specifically, it chooses $\Delta g^{\max}(n-1)$ such that this hypothetical upper bound $t_b^{new}(n-1)$ is no more than fraction p of the way from the target $t_T(n-1)$ to the playback deadline $t_d(n-1)$. In our experiments, we choose $p = 1/3$.

When a frame with a new average coding rate $\hat{r}_c(n)$

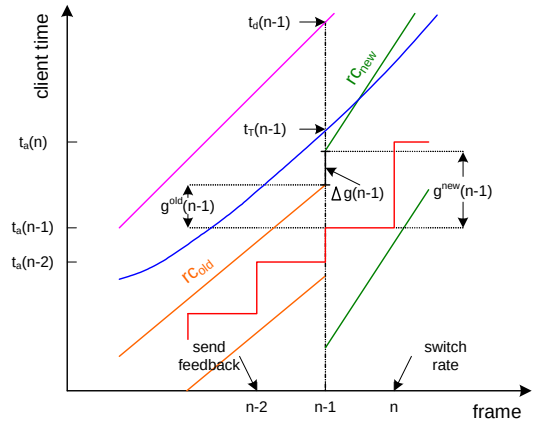


Fig. 8. Buffer tube change and control target adjustment.

arrives at the client at time $t_a(n)$, the upper bound shifts by approximately $\Delta g(n-1)/\tilde{r}_a$, where $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$, as illustrated in Figure 8. This shift can be on the order of seconds and hence, rather than being negligible, can be confusing to the controller. Our solution is to introduce a simultaneous shift in the control target schedule equal to $\Delta g(n-1)/\tilde{r}_a$. The server can send this value to the client along with frame n. If there is no stream change, this value is simply zero.

If the control target schedule is adjusted whenever the coding rate changes, it will no longer follow the designed target schedule. We refer to the adjusted target schedule as the *control target* schedule to distinguish it from the *designed target* schedule (or simply the *target schedule*). The control target schedule, of course, must eventually attempt to return to the designed target schedule. The basic idea is to decrease the slope of the control target schedule when it is above the designed target schedule and to increase the slope when it is below. For details, see [9].

V. Performance Evaluation

In this section, we evaluate the performance of the optimal rate control system when streaming a fine grained scalable (FGS) video stream.

The test video is a 3-minute clip, which we obtain by six repetitions of the concatenation of the three MPEG standard test sequences *Akiyo*, *Stefan*, and *Foreman* in that order. The test video is downsampled to QCIF, 10 fps, for a total of 1800 underlying QCIF frames.[3] The test video is coded using a variant of MPEG-4 FGS [10], with a 10-second I-frame distance and no B frames. Using rate-distortion optimization, from the FGS stream we extract 50 substreams whose average coding rates are uniformly spaced in the log domain between log 50 kbps and log 1000 Kbps.

[3]The original Akiyo and Stefan test sequences are 300 frames, which we downsample to 100 frames each. The original Stefan test sequence is 400 frames, from which we extract the first 300 frames before downsampling to 100 frames.

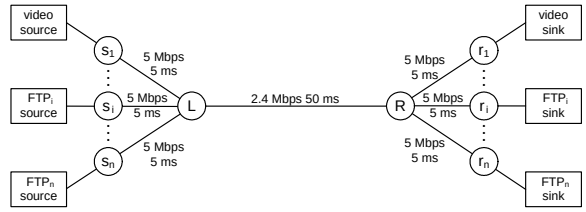


Fig. 9. ns-2 Simulation network setup.

	client time	# of FTPs	fare share BW
Constant Bandwidth	0–180 s	5	400 Kbps
Variable Bandwidth	0–30 s	2	800 Kbps
	30–60 s	5	400 Kbps
	60–90 s	11	200 Kbps
	90–130 s	5	400 Kbps
	130–180 s	2	800 Kbps

TABLE I
Bandwidth Available to the Streaming Session

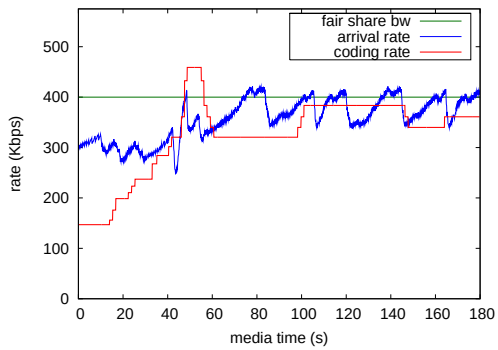
Using the popular network simulator ns-2 [11], we set up a simple network environment as shown in Figure 9. Video traffic is streamed from node s_1 to node r_1 while competing FTP cross traffic (FTP_i) is transmitted node s_i to node r_i ($2 \leq i \leq n$). By adjusting the number of FTP flows and their beginning/ending times, we can create both constant and variable available bandwidth scenarios for the streaming session, as specified in Table I. Experiments are carried out using both TCP and TFRC [12] as alternative transport layer protocols.

A. Basic Performance

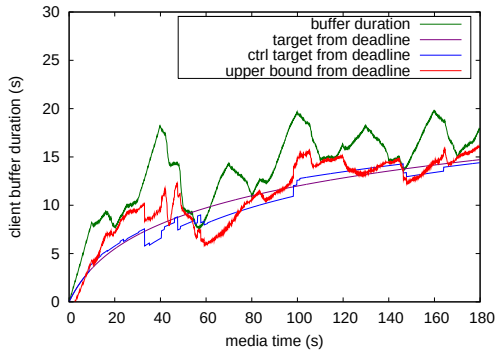
Figures 10 and 11 show results using TCP as the transport protocol, under constant and variable bandwidth conditions, respectively. Figures 10(a) and 11(a) show the evolution of the arrival and coding rates over time, while Figures 10(b) and 11(b) show the number of seconds in the client buffer between the playback deadline and 1) the arrival time, 2) the buffer tube upper bound, 3) the control target, and 4) the ideal (logarithmic) target, respectively.

In both constant and variable bandwidth conditions, in the startup phase, the coding rate is about half of the arrival rate, which allows fast startup and helps to build the client buffer quickly. The coding rate catches up smoothly with the arrival rate and tracks it smoothly despite fluctuations in the available bandwith. As the result of coding rate adjustments, the client buffer is well maintained around the logarithmic target schedule, ensuring that no frame misses its playback deadline.

All of the above performance figures show significant deviation of the buffer tube upper bound from the control target, which is especially obvious in the variable bandwidth case. We can reduce this deviation by decreasing the value of σ. A smaller value of σ value implies a relative larger penalty on the deviation term in the cost function and thus forces the upper bound to track the target more closely. This, however, happens at the cost of sacrificing

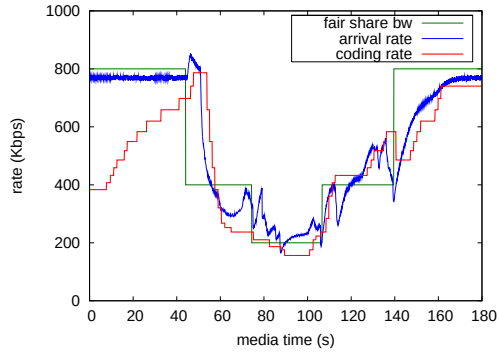


(a) rate vs. time

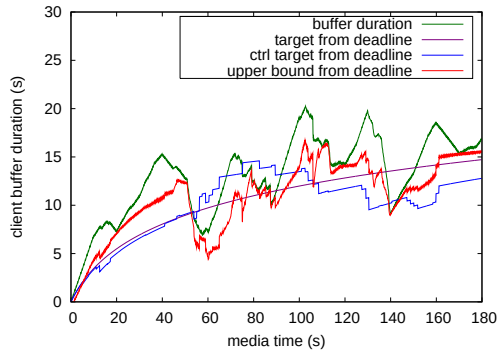


(b) buffer vs. time

Fig. 10. Constant bandwidth over TCP.



(a) rate vs. time



(b) buffer vs. time

Fig. 11. Variable bandwidth over TCP.

coding rate smoothness, since the corresponding term in the cost function will be weighted less. Figure 12 shows simulation results with $\sigma = 500$ under the same network conditions as in Figure 10. It is clear that while the buffer tube upper bound deviates only slightly from the control target, the coding rate has undesirable oscillations.

On the other hand, a large σ value will certainly yield smoother coding rates, but might also incur client buffer underflow since the buffer tube upper bound is allowed to deviate significantly away from the control target. Therefore, a good choice of σ should take into account this trade-off. In our implementation, we choose $\sigma = 4000$ when the coding rate switches up and $\sigma = 2000$ when it switches down. Note that we allow a slightly more aggressive strategy in the latter case to further reduce the chance of client buffer underflow. It is straightforward to verify that this choice of σ maintains a stable closed-loop and good gain/phase margins; this is not repeated here.

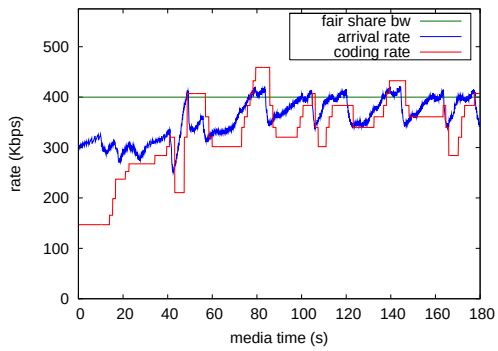
Using TFRC instead of TCP produces similar results, showing that TFRC is not really necessary for streaming media on demand given that the client is able to use a sufficiently large buffer. For further information and a detailed experimental evaluation, see [9].

B. Performance Comparison

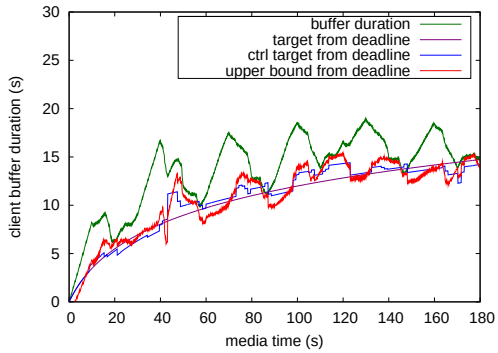
We compare our buffer management algorithm to two existing algorithms.

As a benchmark, the first is the windowing algorithm in [13] (which is part of the rate-distortion optimized sender-driven streaming algorithm therein). In the benchmark algorithm, the server maintains a sending window, which contains the range of frames that are potentially in the client buffer. The sending window slides forward to mimic the playback (consumption) of frames at the client. At each transmission opportunity, the sender selects from the window a data unit that most decreases the distortion at the client (per transmitted bit). The sliding window looks ahead based on a logarithmic function (similar to the logarithmic target schedule herein), which starts small and grows slowly over time. Hence, the client can have low startup delay and can gradually increase its buffer over time.

Although conceptually simple and sound, the benchmark algorithm has two disadvantages. First, it does not send out data units in the order in which they appear in the media file (i.e., decoding order). This demands resources (e.g., caching large segments of data) that may be incompatible with high performance streaming. Second and more importantly, until the window becomes large enough to accommodate constant quality streaming (about 25 seconds for typical movies), the benchmark algorithm demands, essentially, constant bit rate streaming. This is because the duration of the client buffer is determined by the logarithmic function. In contrast, in our algorithm,



(a) rate vs. time



(b) buffer vs. time

Fig. 12. Constant bandwidth over TCP, $\sigma = 500$. The upper bound tracks the control target more closely, while the coding rate is less smooth, compared to Figure 10.

only a portion of the client buffer duration (namely the safety zone between the target and the playback deadline) is determined by the logarithmic function. The remainder of the client buffer duration is determined by the leaky bucket state when processing the video content.

The second is the CBR algorithm, a simple rate control mechanism that takes advantage of the ability of to truncate an FGS encoded frame at any point. Thus it is possible to control the rate by sending the media data in real time, but truncating each frame to match to available transmission rate. If the transmission rate is constant, this yields a constant number of bits per frame. The algorithm is simple and effective in the sense that it successfully avoids any risk of rebuffering by matching the instantaneous coding rate to the transmission rate. However, without taking into account the variable bit rate nature of constant quality coding, this algorithm results in high quality for smooth content (which is easy to encode), and low quality for high-action content (which is hard to encode).

To compare the rate-distortion performance of all aforementioned algorithms, experiments over a wide range of available bandwidth (150-900 Kbps) are carried out. Each experiment sets a constant available bandwidth for the streaming session and TCP protocol is used for all

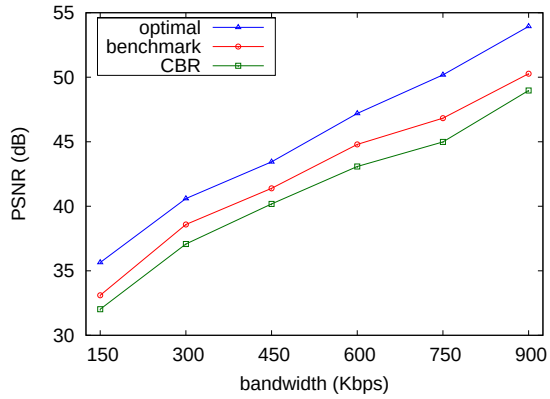


Fig. 13. Rate-Distortion comparison.

experiments. The average distortion in terms of PSNR over each session is computed on the client side and plotted in Figure 13. Note that frames over the first 40 s (media time) are excluded from the average distortion computation. These frames correspond roughly to the time period (about 30 s in client time) when the client buffer is built up by streaming at lower coding rates than the available bandwidth. The quality sacrifice during the initial period will be easily amortized over streaming sessions of reasonable length and it is appropriate not to be considered in this rate-distortion comparison (where each session is just 3 minutes long).

From the reported results, we can see that the optimal coding rate control algorithm has better rate-distortion performance than the benchmark and the CBR algorithms. Over the wide range of bandwidth, the optimal coding rate control algorithm yields about 2-3 dB PSNR gain over the benchmark algorithm. The reason that the CBR algorithm has worse performance than the benchmark algorithm is clear. The CBR algorithm can be regarded as an extreme case of the benchmark algorithm, where the sending window maintained on the server side contains only one frame data at any time. Hence, the limited ability of the benchmark algorithm to smooth quality is further reduced in this case.

VI. Related Work

Hsu, Ortega and Reibman [6] address the problem of joint selection of source and channel rates (which are notions analogous to coding and transmission rates in this paper) for VBR video. They propose a rate-distortion optimization solution that maximizes receiving quality subject to end-to-end delay guarantees. Luna, Kondi and Katsaggelos [14] pursue this direction further by introducing network cost as an optimization objective and balancing the trade-off between user satisfaction and network cost. Both approaches assume networks that offer QoS support while using various policing mechanisms (such as a leaky bucket model) to constrain network traffic. The algorithms in these papers can be modified to address the problem, which we deal with in our paper, where the channel rate is completely determined by network conditions and not

subject to choice. However, a drawback of these algorithms compared to our optimal control mechanism is that they require complete knowledge of channel rates *a priori*, which makes them less practical for streaming media applications, where dynamic rate adjustment is required on the fly. Moreover, these algorithms have higher complexity, even with fast approximation variations [15]. The algorithms are good, however, for determining performance bounds in offline analysis.

To our knowledge, the most closely related contemporaneous work is that by de Cuetos and Ross [16], which also decouples the transmission rate and the coding rate. They assume that the transmission rate is determined by the network transport protocol (TCP or TFRC), which is the same assumption that we make in our paper. They develop a heuristic real time algorithm for adaptive coding rate control and compare its performance to an optimal offline coding rate control policy if the transmission rate is given prior to streaming. Our work differs from theirs in two ways. One is that our rate control algorithm is optimal in a control theoretic sense, in addition to being a low complexity real time algorithm. The other is that we take into account the variable instantaneous bit rate of the media coding and thereby further improve and stabilize the receiving quality.

The work of Rejaie, Handley and Estrin [17] proposes a scheme for transmitting layered video in the context of unicast congestion control, which basically includes two mechanisms. One mechanism is a coarse-grained mechanism for adding and dropping layers (changing the overall coding rate and quality). The other is a fine-grained interlayer bandwidth allocation mechanism to manage the receiver buffer (not changing the overall coding rate or quality). A potential issue with this approach is that it changes the coding rate by adding or dropping one (presumably coarse) layer at a time. If the layers are fine-grained, as in the case of FGS coded media, then adding or dropping one (fine-grained) layer at a time typically cannot provide a prompt enough change in coding rate. Moreover, since the adding and dropping mechanism is rather empirical, the mechanism may simply not be suitable for FGS media.

The work of Q. Zhang, Zhu and Y-Q. Zhang [18] proposes a resource allocation scheme to adapt the coding rate to estimated network bandwidth. The novelty of their approach is that they consider minimizing the distortion (or equivalently maximizing the quality) of all applications, such as file-transfers and web browsing in addition to audio/video streaming. However, their optimization process does not include the smoothness of individual streams and might lead to potential quality fluctuations. In our paper, we explicitly take into account the smoothness of the average coding rate over consecutive frames in our optimal controller, which yields a higher and more stable quality as network conditions change.

VII. Summary

In this paper, we propose and verify an optimal online rate control algorithm for scalable streaming media. Our extensive analytical and experimental results show that three goals are achieved: fast startup (about 1 s delay without bursting), continuous playback in the face of severe congestion, and maximal quality and smoothness over the entire streaming session. We also show that our algorithm works effectively with both TCP and TFRC transport protocols.

References

[1] W. Birney, "Intelligent streaming," May 2003, http://-www.microsoft.com/windows/windowsmedia/howto/articles/-intstreaming.aspx.

[2] M. Kalman, E. Steinbach, and B. Girod, "Adaptive media playout for low delay video streaming over error-prone channels," *IEEE Trans. Circuits and Systems for Video Technology*, to appear.

[3] C. Huang, P. A. Chou, and A. Klemets, "Optimal control of Multiple Bit Rates for streaming media," in *Proc. Picture Coding Symposium*, San Francisco, CA, Dec. 2004.

[4] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik, "Video coding for streaming media delivery on the Internet," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 269–281, Mar. 2001, special issue on Streaming Video.

[5] J. Ribas-Corbera, P. A. Chou, and S. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003.

[6] C.-Y. Hsu, A. Ortega, and A. Reibman, "Joint selection of source and channel rate for VBR video transmission under ATM policing constraints," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 5, pp. 1016–1028, Aug. 1997.

[7] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Prentice Hall, 1990.

[8] G. Franklin, J. Powell, and M. Workman, *Digital Control of Dynamic Systems*, 3rd ed. Addison-Wesley, 1997.

[9] C. Huang, P. A. Chou, and A. Klemets, "Optimal coding rate control for scalable and multi bit rate streaming media," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-04-XXX, Dec. 2004, in preparation.

[10] F. Wu, S. Li, and Y.-Q. Zhang, "A framework for efficient progressive fine granularity scalable video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, Mar. 2001.

[11] K. Fall and e. K. Varadhan, "The *ns* manual," The VINT Project, Tech. Rep., Dec. 2003, http://www.isi.edu/nsnam/ns/.

[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. Data Communication, Ann. Conf. Series (SIGCOMM)*. Stockholm, Sweden: ACM, Aug. 2000.

[13] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Trans. Multimedia*, 2001, submitted.

[14] C. E. Luna, L. P. Kondi, and A. K. Katsaggelos, "Maximizing user utility in video streaming applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 141–148, Feb. 2003.

[15] A. Ortega, K. Ramchandran, and M. Vetterli, "Optimal trellis-based buffered compression and fast approximation," *IEEE Trans. Image Processing*, vol. 3, pp. 26–40, Jan. 1994.

[16] P. de Cuetos and K. W. Ross, "Adaptive rate control for streaming stored fine-grained scalable video," in *Proc. Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002.

[17] R. Rejaie, M. Handley, and D. Estrin, "Layered quality adaptation for Internet streaming video," *IEEE J. Selected Areas in Communications*, vol. 18, no. 12, pp. 2530–2543, Dec. 2000.

[18] Q. Zhang, Y.-Q. Zhang, and W. Zhu, "Resource allocation for multimedia streaming over the Internet," *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 339–355, Sept. 2001.