Optimal Coding Rate Control of Scalable and Multi Bit Rate 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

Technical Report
MSR-TR-2005-47

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. Finally, we demonstrate that our algorithm is directly applicable to and can significantly improve the performance of existing multi bit rate streaming schemes.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

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 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. This can be done by stream switching in combination with multi bit rate (MBR) coding or coarse grained or fine grained scalable coding. Today's commercial streaming media systems [3], [1] rely on MBR coding as well as *thinning*, which is a form of coarse grained scalability.[2] Future commercial systems may support fine grained scalability (FGS) as well.[3] FGS coding offers great flexibility in adapting to variable network conditions, and can demonstrably improve quality under such conditions.

In this paper we focus 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 this paper 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 the streaming media literature, with few exceptions (e.g., [9], [10] and the works based thereon; also [14]), there has been little attention paid to the the distinction between the coding rate r_c and the arrival rate r_a or the transmission rate r_x. Indeed, in typical streaming media systems (e.g., [1]), after an initial buffering period (in which $\nu = 0$ and possibly

[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.

[2] In MBR coding, semantically identical content is encoded into alternative bit streams at different coding rates and stored in the same media file at the server, allowing the content to be streamed at different levels of quality corresponding to the coding rates r_c, possibly using bit stream switching [4]. In coarse grained scalable coding (such as MPEG-2/4 temporal or SNR scalability [5]) the content is encoded into several substreams or *layers*, so that the coding rate r_c can be changed in large deltas by adding or dropping (at possibly restricted times) one layer of content at a time. Thinning is a special case of coarse grained scalability in which dependent video frames (P and B frames) are dropped before independent video frames (I frames), which are in turn are dropped before audio frames.

[3] Fine grained scalable coding (such as 3D SPIHT [6], MPEG-4 FGS [7], or EAC [8]) allows the coding rate r_c to change at any time in deltas sometimes as small as one byte per presentation.

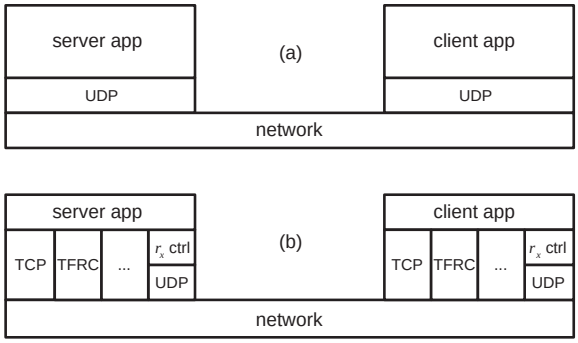


Fig. 1. (a) Traditional streaming media architecture. (b) Proposed streaming media architecture with congestion control factored out.

$r_x/r_c > 1$), r_x/r_c is locked to ν. A difficulty with locking the transmission rate to the coding rate via the playout speed is that it essentially removes any means of controlling the client buffer duration after the initial buffering period.[4] A further difficulty is that the transmission rate, if it is locked to the coding rate, will typically be incompatible with transports that use standard congestion control, such as TCP and TFRC [15].

By decoupling the coding and transmission rates, it is possible to continually control the client buffer duration. This allows the buffer to grow over time, for example, providing a low startup delay, asymptotically high robustness, and eventual constant quality. Furthermore, decoupling the coding and transmission rates makes possible an architecture in which the transport and congestion control protocol may be factored out of the streaming problem, if desired. Figure 1(a) illustrates the traditional architecture in which congestion control is integrated into the streaming media application running on top of UDP. Figure 1(b) illustrates the proposed architecture in which congestion control is factored out of the streaming media application, allowing standard transport mechanisms (such as TCP and TFRC) to be used, as well as custom transport solutions using custom transmission rate control over UDP.

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.[5]

We present the major ideas in our paper as follows. In Section II, we introduce the preliminaries, including temporal coordinate systems, the leaky bucket model, and the coding rate control objective. In Section III, we model the client buffer as the plant in a feedback control system and we show how to control the buffer duration using optimal linear quadratic control. In Section IV we address several additional practical issues related to scalable streaming. In Section V we explain implementation issues on both sender and receiver sides. Section VI presents experimental results in ns-2 with real scalable encoded video data. In Section VII, the optimal control approach is applied to the multiple bit rate situation and yields satisfactory performance. We discuss related work in Section VIII and conclude with Section IX.

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* refers to the clock running on the device used to capture and timestamp the original content, while *client time* refers to the clock running on the client used to play back the content. We assume that media time is real time (i.e., one second of media time elapses in one second of real time) at the time of media capture, while client time is real time at the time of media playback. We use the symbol τ to express media time and the symbol t to express client time, with subscripts and other arguments to indicate corresponding events. For example, we use $\tau_d(0), \tau_d(1), \tau_d(2), \ldots$ to express the playback deadlines of frames $0, 1, 2, \ldots$ in media time, while we use $t_d(0), t_d(1), t_d(2), \ldots$ to express the playback deadlines of frames $0, 1, 2, \ldots$ at the client. Content may be played back at a rate ν times real time. Thus 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 (or the playback of the first frame after a seek or rebuffering event) in media and client coordinate systems, respectively.

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 2. 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 3 illustrates

[4]However, congestion, as evidenced by a drop in r_a and hence a drop in the buffer duration, can still be alleviated by reducing r_x and r_c by the same factor.

[5]Ribas, Chou, and Regunathan use a leaky bucket to model source coding rate constraints to reduce initial startup delay [16], while Hsu, Ortega and Reibman use a leaky bucket to model transmission rate contraints [17].

Fig. 2. Communication pipeline.

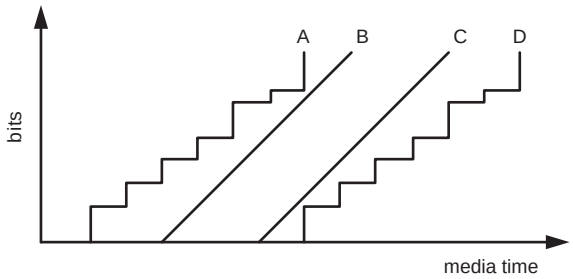


Fig. 3. Schedules at which bits in the coded bit stream pass the points A, B, C, and D in the communication pipeline.

the schedules of bits passing the points A, B, C, and D in Figure 2. 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. Thus the coding schedule (either A or D) can be contained within a *buffer tube*, as illustrated in Figure 4, 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. In general it is possible for the leak rate R to be high enough so that the bucket occasionally empties. Thus the encoder buffer fullness $F^e(n)$ immediately before frame n is added to the bucket and the encoder buffer fullness $B^e(n)$ immediately after frame n is added to the bucket evolve from an initial encoder buffer fullness $F^e(0) = F^e$ according to the dynamical system

$$B^e(n) = F^e(n) + b(n), \quad (2)$$

$$F^e(n+1) = \max\{0, B^e(n) - R/f(n)\}, \quad (3)$$

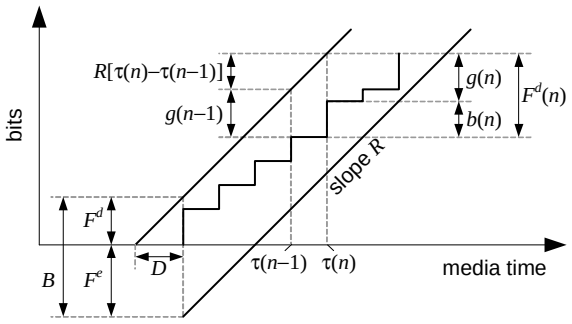


Fig. 4. Buffer tube containing a coding schedule.

where

$$f(n) = \frac{1}{\tau(n+1) - \tau(n)} \quad (4)$$

is the instantaneous frame rate, for $n = 0, 1, 2, \ldots$. If R is sufficiently low, then the bucket will never run dry (underflow), but if R is too low the bucket will eventually overflow. We take the largest R such that the buffer will never run dry to be the average coding rate r_c of the bit stream. This is made more precise in the following two paragraphs.

A leaky bucket with size B, rate R, and initial fullness F^e is said to *contain* a stream having a schedule characterized by the steps $\{(b(n), \tau(n))\}$ if $B^e(n) \le B$ for all n. We define the minimum bucket size needed to contain the stream given leak rate R and initial fullness F^e as

$$B^e_{\min}(R, F^e) = \min_n B^e(n), \quad (5)$$

while we define the corresponding initial decoder buffer fullness as

$$F^d_{\min}(R, F^e) = B^e_{\min}(R, F^e) - F^e. \quad (6)$$

We denote the minimum of each of these over F^e as

$$B^e_{\min}(R) = \min_{F^e} B^e_{\min}(R, F^e), \quad (7)$$

$$F^d_{\min}(R) = \min_{F^e} F^d_{\min}(R, F^e). \quad (8)$$

It is shown in [16, Proposition 2] that remarkably, these are each minimized by the same value of F^e, which is hence equal to

$$F^e_{\min}(R) = B^e_{\min}(R) - F^d_{\min}(R). \quad (9)$$

Thus given a bit stream with schedule $\{(b(n), \tau(n))\}$, for each bit rate R there is a unique leaky bucket that contains the stream and that has the minimum buffer size B as well as the minimum startup delay $D = F^d/R$. These parameters can be computed with the above equations.

For sufficiently low leak rates R, the leaky bucket does not underflow, when beginning with initial fullness $F^e = F^e_{\min}(R)$. We may use the maximum such rate R as the average coding rate r_c of a bit stream with coding schedule $\{(b(n), \tau(n))\}$.

Leak rates R greater than r_c will also be used in this paper. It is shown in [16] that both $B^e_{\min}(R)$ and $F^d_{\min}(R)$ are decreasing, piecewise linear, and convex in R. Hence if the transmission rate R is greater than the average coding rate r_c,

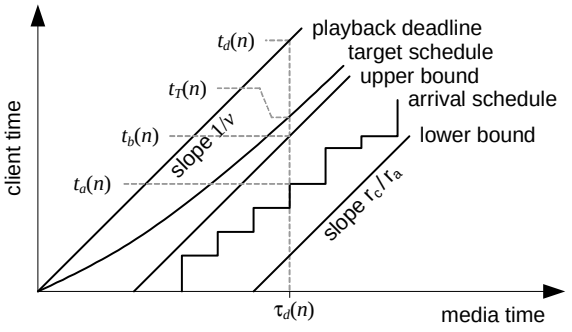


Fig. 5. Arrival schedule and its upper bound in client time. The upper bound is controlled to the target schedule, which is increasingly in advance of the playback deadline to provide greater robustness over time.

the startup delay $D = F^d_{\min}(R)/R$ can be reduced compared to $D = F^d_{\min}(r_c)/R$. This fact will be used in Section IV-A.

A leaky bucket with leak rate $R = r_c$, size $B = B^e_{\min}(r_c)$ and initial decoder buffer fullness $F^d = F^d_{\min}(r_c)$ thus corresponds to a straight buffer tube bounding the coding schedule as in Figure 4. Each stream in the media file has a coding schedule; thus each stream corresponds to a straight buffer tube with slope equal to the average coding rate r_c of the stream. The size B of the buffer tube and its offset F^e (or F^d) relative to the coding schedule can be either computed by the above formula for a variable bit rate (VBR) stream (such as a constant-quality substream of a scalable stream), or obtained from the size B and initial state F^e of the actual encoder buffer used to encode the stream if it is a constant bit rate (CBR) 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 4. Note that the decoder buffer fullness $F^d(n) = B - F^e(n)$ can also be expressed

$$F^d(n) = b(n) + g(n) = g(n-1) + \frac{r_c(n)}{f(n)}, \qquad (10)$$

where $r_c(n)$ 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 4 by r_a, we obtain the schedules in terms of client time, rather than bits, as shown in Figure 5. 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

$$r_a(n) = \frac{b(n)}{t_a(n) - t_a(n-1)} \qquad (11)$$

to be the *instantaneous arrival rate* at frame n. In practice we estimate the average arrival rate at frame n by a moving average $\tilde{r}_a(n)$ of previous values of $r_a(n)$, as detailed in Section IV-C. Hence using (11) we may express the arrival time of frame n in terms of the arrival time of frame $n-1$ as

$$t_a(n) = t_a(n-1) + \frac{b(n)}{r_a(n)} \qquad (12)$$

$$= t_a(n-1) + \frac{b(n)}{\tilde{r}_a(n)} + v(n), \qquad (13)$$

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 (10), however, we have

$$b(n) = \frac{r_c(n)}{f(n)} + g(n-1) - g(n), \qquad (14)$$

whence (substituting (14) into (13)) we have

$$t_a(n) = t_a(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n)}{\tilde{r}_a(n)} + v(n). \qquad (15)$$

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

$$t_b(n) = t_a(n) + \frac{g(n)}{\tilde{r}_a(n)}, \qquad (16)$$

so that

$$t_b(n) - t_b(n-1) = t_a(n) - t_a(n-1) + \frac{g(n)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)}, \qquad (17)$$

we obtain the following update equation:

$$t_b(n) = t_b(n-1) + \frac{r_c(n)}{f(n)\tilde{r}_a(n)} + w(n-1), \qquad (18)$$

where

$$w(n-1) = \frac{g(n-1)}{\tilde{r}_a(n)} - \frac{g(n-1)}{\tilde{r}_a(n-1)} + v(n) \qquad (19)$$

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

Using (16), 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 $r_c(n-1)$ using (10)). Then using (18), 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, (18) 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 (18), 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 5, 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 schedule for frame n, we use the following cost function to reflect both of our concerns:

$$I = \sum_{n=0}^{N} \left(\big(t_b(n) - t_T(n)\big)^2 + \sigma \left(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a(n)} \right)^2 \right), \tag{20}$$

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.

III. Optimal Control Solution

Before presenting the optimal control solution, we first describe the design rational of the target schedule.

A. Target Schedule Design

Figure 6 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.

The slope of the target schedule relates the average coding rate to the average arrival rate. Let $t_T(n)$ be the target for frame n. As illustrated in Figure 6, the slope of the target schedule at frame n is

$$s(n) = \frac{t_T(n+1) - t_T(n)}{\tau(n+1) - \tau(n)}. \tag{21}$$

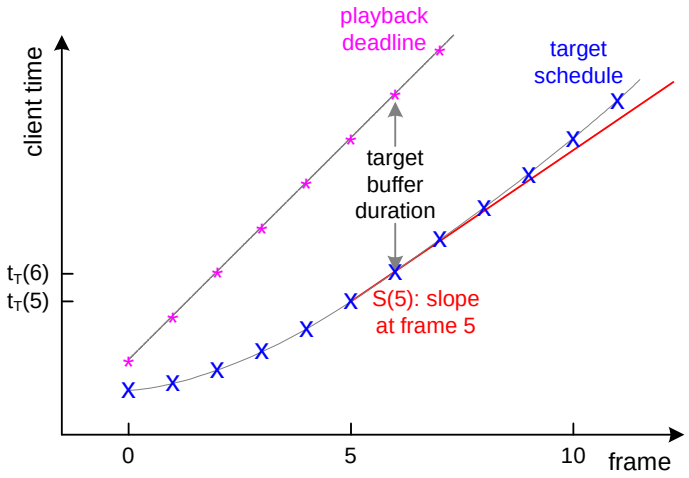


Fig. 6. Target schedule design.

If the upper bound $t_b(n)$ aligns perfectly with the target schedule (i.e., $t_b(n) = t_T(n)$) and the arrival rate r_a is constant (i.e., the $w(n-1)$ term vanishes), we get from (18)

$$s(n) = \frac{t_b(n+1) - t_b(n)}{\tau(n+1) - \tau(n)} = \frac{r_c(n)}{r_a}. \tag{22}$$

Thus initially, when the slope is low, i.e., less than $1/\nu$, r_a/r_c is greater than ν and more than ν seconds of content are received per second of client time, causing the client buffer (which is playing out only ν seconds of content per second of client time) to grow. Over time, as the slope approaches $1/\nu$, r_a/r_c approaches ν and the buffer remains relatively constant (except for changes due to variations in the instantaneous coding rate), since content is received and played back at the same speed ν. We next present two target schedule functions that illustrate the general design idea.

1) Logarithmic Target Schedule: One way to choose the target schedule t_T is to have the client buffer duration grow logarithmically over time. Specifically, if t_d is the playback deadline, then for each t_d greater than some start time t_{d0},

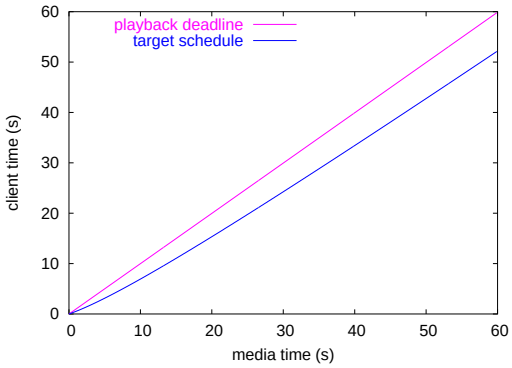
$$t_T = t_d - \frac{b}{a} \ln(a(t_d - t_{d0}) + 1). \tag{23}$$

Since by (1), $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$, we have

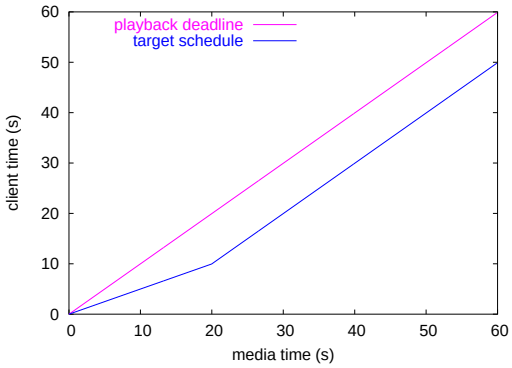
$$s = \frac{dt_T}{d\tau_d} = \frac{dt_T}{dt_d}\frac{dt_d}{d\tau_d} = \frac{1}{\nu} - \frac{b}{a(\tau_d - \tau_{d0}) + \nu}, \tag{24}$$

and hence the initial slope at frame 0 (when $t_d = t_{d0}$) is $s(0) = (1-b)/\nu$. Setting $b = 0.5$ implies that initially $r_c/r_a = 0.5/\nu$, causing the client buffer to grow initially at two times real time. Further setting $a = 0.15$ implies that the client buffer duration will be 7.68 seconds after 1 minute, 15.04 seconds after 10 minutes, and 22.68 seconds after 100 minutes, regardless of ν.

2) Two-piece Linear Target Schedule: Another way to choose the target schedule t_T is to have the client buffer duration grow linearly at rate b seconds of media time per second of client time until the buffer duration reaches a seconds of



(a) logarithmic ($a = 0.15$, $b = 0.5$)



(b) two-piece linear ($a = 10$, $b = 0.5$)

Fig. 7. Target schedules.

media time, after which it remains constant. Specifically, for each t_d greater than some start time t_{d0},

$$t_T = \begin{cases} t_d - b(t_d - t_{d0}) & t_d \le t_{d0} + a/b \\ t_d - a & t_d \ge t_{d0} + a/b \end{cases}. \quad (25)$$

The initial slope is again $s(0) = (1-b)/\nu$. Setting $b = 0.5$ implies that initially $r_c/r_a = 0.5/\nu$, causing the client buffer to grow initially at two times real time. Further setting $a = 10$ implies that the client buffer duration will reach 10 seconds of media time after 20 seconds of client time, regardless of ν.

Figure 7 shows the above two target schedules. As one can see, if a client buffer duration of 10 seconds is considered to be a safe level against jitter, delay and network fluctuations, then the two-piece linear target schedule reaches the safe level in 20 seconds, much faster than the logarithmic target schedule. On the other hand, the slope of the two-piece linear target schedule remains lower for longer (hence the coding rate and quality are lower for longer) and furthermore experiences an abrupt change at 20 seconds when its slope changes from $0.5/\nu$ to $1/\nu$. Consequently, the coding rate will not change as smoothly as with the logarithmic target schedule, although it will not be as abrupt as the schedule itself because of the smoothness objective in the controller design. Hence, we investigate the effect of both target schedules.

B. Optimal Controller Design

Recall from (18) the fundamental state transition equation, which describes the evolution of the buffer tube upper bound $t_b(n)$ in terms of the coding rate $r_c(n)$:

$$t_b(n+1) = t_b(n) + \frac{r_c(n+1)}{f\tilde{r}_a} + w(n). \quad (26)$$

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

$$t_T(n+1) - 2t_T(n) + t_T(n-1) = 0. \quad (27)$$

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

$$e(n) = t_b(n) - t_T(n), \quad (28)$$

we obtain

$$\begin{aligned} & e(n+1) - e(n) \\ &= (t_b(n+1) - t_T(n+1)) - (t_b(n) - t_T(n)) && (29) \\ &= (t_b(n+1) - t_b(n)) - (t_T(n+1) - t_T(n)) && (30) \\ &= \frac{r_c(n+1)}{f\tilde{r}_a} - (t_T(n+1) - t_T(n)) + w(n), && (31) \end{aligned}$$

from which we obtain in turn

$$\begin{aligned} & (e(n+1) - e(n)) - (e(n) - e(n-1)) \\ &= [r_c(n+1) - r_c(n)]/f\tilde{r}_a \\ &\quad -(t_T(n+1) - 2t_T(n) + t_T(n-1)) \\ &\quad +(w(n) - w(n-1)) && (32) \\ &= \frac{r_c(n+1) - r_c(n)}{f\tilde{r}_a} + (w(n) - w(n-1)). && (33) \end{aligned}$$

We next define the control input

$$u(n) = \frac{r_c(n+2) - \hat{r}_c(n+1)}{\tilde{r}_a}, \quad (34)$$

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

$$d(n) = \frac{\hat{r}_c(n) - r_c(n)}{f\tilde{r}_a} + w(n) - w(n-1). \quad (35)$$

Then (33) can be rewritten

$$e(n+1) = 2e(n) - e(n-1) + \frac{u(n-1)}{f} + d(n). \quad (36)$$

Therefore, defining the state vector

$$\mathbf{e}(n) = \begin{bmatrix} e(n) \\ e(n-1) \\ u(n-1) \end{bmatrix} = \begin{bmatrix} t_b(n) \\ t_b(n-1) \\ \frac{r_c(n+1)}{\tilde{r}_a} \end{bmatrix} - \begin{bmatrix} t_T(n) \\ t_T(n-1) \\ \frac{\hat{r}_c(n)}{\tilde{r}_a} \end{bmatrix}, \quad (37)$$

the error space representation of the system can be expressed

$$\mathbf{e}(n+1) = \begin{bmatrix} 2 & -1 & \frac{1}{f} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e}(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} d(n), \quad (38)$$

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

$$u(n) = -G\mathbf{e}(n), \quad (39)$$

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

$$r_c(n+2) = \hat{r}_c(n+1) - G\mathbf{e}(n)\tilde{r}_a. \quad (40)$$

Finding the optimal linear controller amounts to finding the feedback gain G^* that minimizes the quadratic cost function (20), as defined in Section II-D. Before continuing with the design, we first check the system *controllability matrix* $\mathcal{C}$,

$$\mathcal{C} = \begin{bmatrix} \Gamma & \Phi\Gamma & \Phi^2\Gamma \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{f} & \frac{2}{f} \\ 0 & 0 & \frac{1}{f} \\ 1 & 0 & 0 \end{bmatrix}, \quad (41)$$

which has full rank for any frame rate f. Thus, the system is *completely controllable* [19] 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

$$I = \sum_{n=0}^{N} \Big\{ \Big(t_b(n) - t_T(n) \Big)^2 + \sigma \Big(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a} \Big)^2 \Big\} \quad (42)$$

$$= \sum_{n=0}^{N} \Big\{ \mathbf{e}(n)^T Q \mathbf{e}(n) + u(n-1)^T R u(n-1) \Big\}, \quad (43)$$

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 [18, Section 3.3] to obtain an optimal regulator in two steps: 1) solving, to get S, the *discrete algebraic Riccati equation* (DARE)

$$S = \Phi^T \{ S - S\Gamma[\Gamma^T S\Gamma + R]^{-1}\Gamma^T S \}\Phi + Q, \quad (44)$$

and 2) computing the optimal feedback gain

$$G^* = [\Gamma^T S\Gamma + R]^{-1}\Gamma^T S\Phi. \quad (45)$$

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.

C. Frame Rate

In the previous section, we assumed that the frame rate is constant. This assumption is reasonable when streaming a single medium, such as video without audio.[6] However, usually video and audio are streamed together, and their merged coding schedule may have no fixed frame rate. Even if there is a fixed frame rate f, we may wish to operate the controller at a rate lower than f, to reduce the feedback rate, for example.

To address these issues, in practice we use the notion of a *virtual frame rate*. We choose a virtual frame rate f, for example $f = 1$ frame per second (fps); we partition media time into intervals of size $1/f$; and we model all of the (audio and video) frames arriving within each interval as a *virtual frame* whose decoding and playback deadline is the end of the interval.

This approach has several advantages. First, it allows us to design offline a universal feedback gain, which is independent of the actual frame rate of the stream or streams. Second, it allows us to reduce the rate of feedback from the client to the server. And finally, since the interval between virtual frames is typically safely larger than a round trip time (RTT), a one-frame delay in the error space model (as described in the previous section) is sufficient to model the feedback delay. Otherwise we would have to model the feedback delay with approximately RTT/f additional state variables to represent the network delay using a shift register of length RTT/f.

In the sequel we therefore use a virtual frame rate $f = 1$ fps, and we refer to this simply as the frame rate.

D. Stability and Robustness

To compute the optimal regulator, it is necessary to choose a value for σ in (20) or (42)-(43). 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 stability) and bode diagram (to check robustness) [19]; 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 $\sigma = 50$. With $f = 1$, the corresponding optimal feedback control gain is then $G^* = [0.6307 \ -0.5225 \ 0.5225]$, 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 8 shows the closed-loop root locus and the bode diagram with the optimal feedback. We can again verify the stability of the

[6] Variable frame rate video is usually achieved by skipping frames, which we can accommodate by setting $b(n) = 0$.

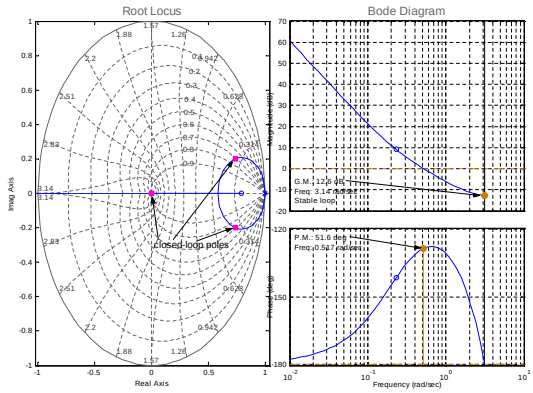
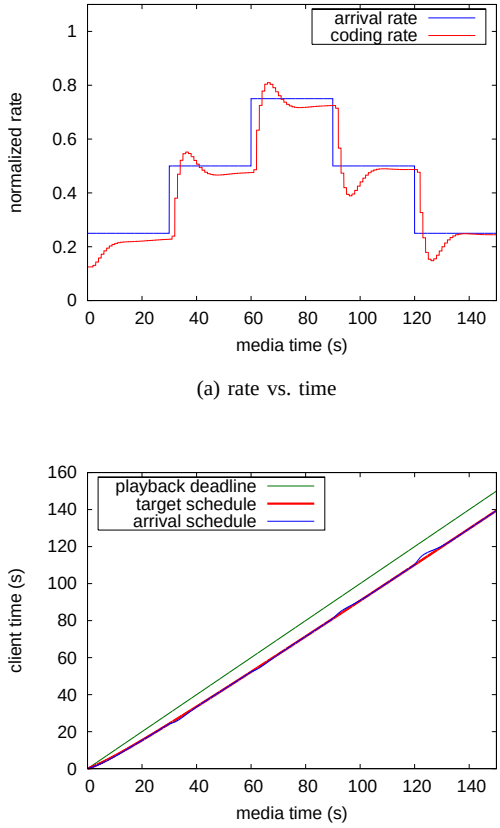


Fig. 8. Root locus and Bode diagram.



(a) rate vs. time

(b) schedule vs. time

Fig. 9. Time response simulation.

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 [19, 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 9 provides the time response simulation results, which show good tracking properties with a fairly stable coding rate.

E. Controller Interpretation

With the above coefficients for G^*, we are now able to give an intuitive explanation of the coding rate control (40). Plugging the coefficients of G^* into (40), we obtain

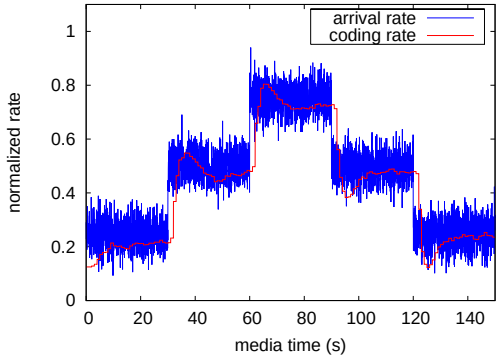
$$\begin{aligned} r_c(n+2) &= \hat{r}_c(n+1) \\ &\quad -0.1082e(n)\tilde{r}_a \qquad (46) \\ &\quad -0.5225[e(n)-e(n-1)]\tilde{r}_a \qquad (47) \\ &\quad -0.5225[r_c(n+1)-\hat{r}_c(n)]. \qquad (48) \end{aligned}$$

Focusing on the first term (46), it can be seen that the coding rate r_c tends to decrease if the current error $e(n) = t_b(n) - t_T(n)$ is positive, and it tends to increase if $e(n)$ is negative, in proportion to $e(n)$ with proportionality constant 0.1082 times the estimated arrival rate $\tilde{r}_a$. This has the effect of moving the upper bound t_b towards the target t_T, whether it is above or below the target. At the same time, from the second term (47), it can be seen that the coding rate tends to decrease if the current error $e(n)$ is numerically greater than the previous error $e(n-1)$, whether $e(n)$ is positive or negative. This has the effect of either stengthening the compensation or preventing the controller from overcompensating, since if $e(n)$ is positive then $e(n) > e(n-1)$ indicates that the magnitude of the error is still growing, while if $e(n)$ is negative then $e(n) > e(n-1)$ indicates that the magnitude of the error is shrinking too fast to be sustainable. The proportionality constant for this second effect is 0.5225 times $\tilde{r}_a$, which is even larger than that for the first effect. Finally, from the third term (48), it can be seen that the coding rate tends to decrease if it had previously increased, with proportionality constant 0.5225. This ensures appropriate damping and smoothing of the coding rate.

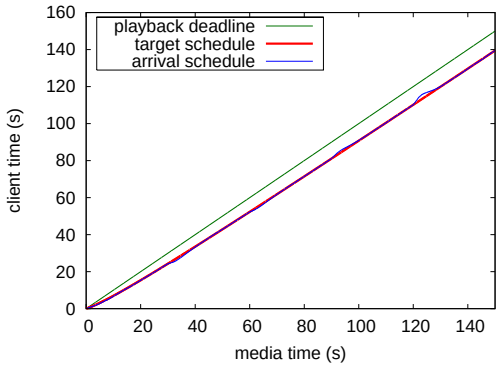
It is important to emphasize that the optimal feedback gain G^* is completely determined given σ and f, and that it is independent of the transmission rate and the coding rate. Thus, G^* can be obtained off line, and only a linear calculation is required to compute the coding rate $r_c(n+2)$ on the fly.

F. Performance Study

In this section, a series of time response simulations are performed to examine the responsiveness and robustness of the optimal controller. The following factors are considered: 1) adding Gaussian noise to the arrival rate r_a; 2) using for r_a either the instantaneous arrival rate or the average arrival rate output from a low pass filter; 3) decreasing the coding rate update frequency by withholding feedback for at least 5 seconds; and 4) withholding feedback of the coding rate unless it decreases/increases by more than 15%. Various combinations of these factors and the corresponding results are summarized in Table I. In the simulations, we assume a stream with a physical frame rate of 30 fps. When Gaussian noise is added to the arrival rate, it is added to the arrivals of the physical frames. However, the virtual frame rate is 1 fps

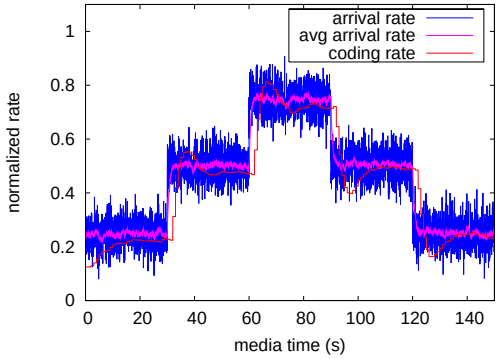


(a) rate vs. time

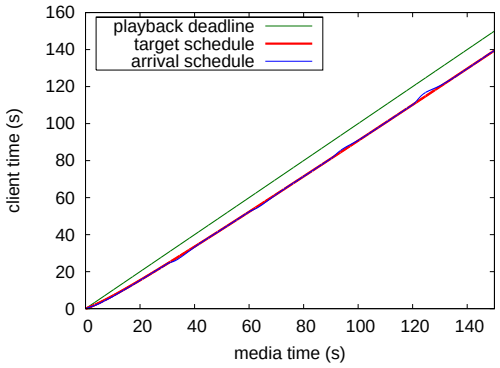


(b) schedule vs. time

Fig. 10. Time response simulation.



(a) rate vs. time



(b) schedule vs. time

Fig. 11. Time response simulation.

	Figure Index						
	9	10	11	12	13	14	15
Added Gaussian noise (0.15) to r_a		√	√	√	√	√	√
Used instantaneous arrival rate as r_a		√					√
Used average arrival rate as r_a	√		√	√	√	√	
Withheld r_c feedback for at least 5 s				√		√	√
Withheld feedback unless r_c changed more than 15%					√	√	√

TABLE I
PERFORMANCE STUDY INDEX TO FIGURES

and hence the coding rate update frequency is at most once per second.

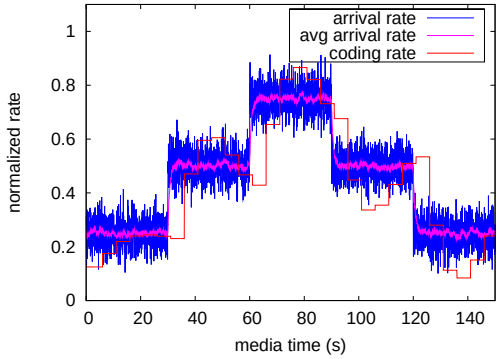
The following conclusions can be drawn from the simulation results: 1) The arrival rate fluctuations certainly cause a decrease in the smoothness of the coding rate (Figures 9, 10 and 11). 2) Using the average arrival rate output from a low pass filter increases the smoothness of the coding rate (Figures 10, 11, 14 and 15). 3) Decreasing the coding rate update frequency (here, 5 seconds per update) affects responsiveness, but still yields acceptable stability (Figures 12, 14 and 15). 4) Withholding coding rate feedback until it reaches a certain threshold (here, 15% difference) yields better coding rate smoothness while not sacrificing responsiveness (Figures 13, 14 and 15).

IV. PRACTICAL ISSUES WITH STREAMING

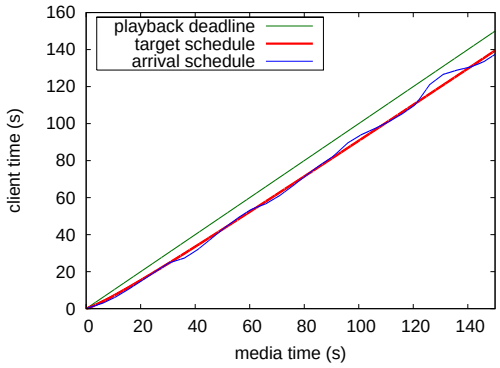
A. Fast Startup

As discussed in previous sections, the startup delay is the length of the period from the time that content first begins to arrive at the client to the time that playback begins. During this period, content accumulates in the receiver buffer to counter packet jitter, retransmission delay, variations in network bandwidth, and variations in instantaneous coding rate. It is conceivable that a longer startup delay would increase the chances of being able to maintain continuous playback in a dynamic network environment. On the other hand, users expect the startup delay to be as small as possible. Thus, it is desirable to investigate techniques that can reduce the startup delay while retaining robustness. One possible approach is to transmit the content at a faster than normal rate at the beginning of streaming. This *bursting* technique will certainly build up the buffer duration in a small amount of time. It, however, puts extra pressure on the network by demanding a higher than normal initial bandwidth, which may not even be available.

In this paper, we use an alternative *fast startup* technique, which takes advantage of the properties of adaptive media. As discussed in previous sections, by choosing an initial coding

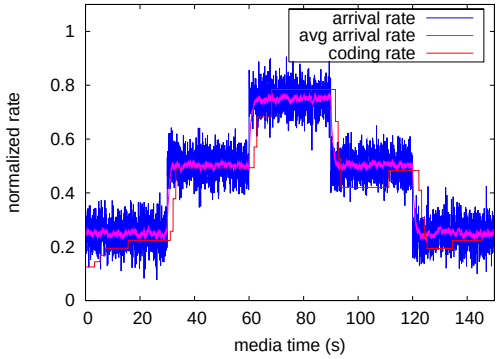


(a) rate vs. time

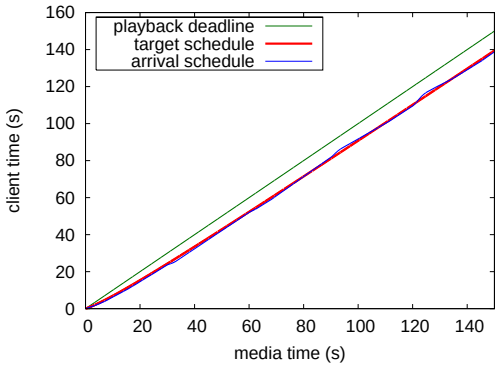


(b) schedule vs. time

Fig. 12. Time response simulation.



(a) rate vs. time



(b) schedule vs. time

Fig. 13. Time response simulation.

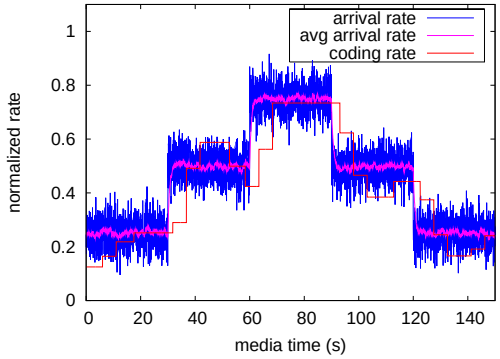
rate r_c equal to half the arrival rate r_a (divided if necessary by the playback speed ν), the client buffer duration can grow at two times real time during playback. Growing the client buffer during playback enables the startup delay to be low, because playback can begin while the buffer duration is still low. Beginning playback while the buffer duration is low is not particularly risky over the short term, because the probability of deep congestion occuring in any short interval is low. However, the probability of deep congestion occuring in a long interval is high, so it is important for the buffer duration to be high over the long term. Without the ability to grow the buffer duration during playback, startup would have to be delayed until the buffer duration was sufficiently high to guarantee continuous playback over the long term.

Moreover, if the transmission rate is twice the coding rate, the startup delay can be further reduced by taking advantage of properties of the leaky bucket model [16]. As detailed in Section II-B, the startup delay for a given bit stream is $D = F^d_{\min}(R)/R$ when the stream is transmitted at rate R. This is ordinarily equal to $F^d_{\min}(r_c)/r_c$ when transmitting the stream at its coding rate. However, when transmitting the stream at a rate $r_a > r_c$ ($r_c = 0.5r_a/\nu$), then the startup delay drops to $F^d_{\min}(r_a)/r_a$. Thus the startup delay D decreases both because the numerator decreases and because the denominator increases.

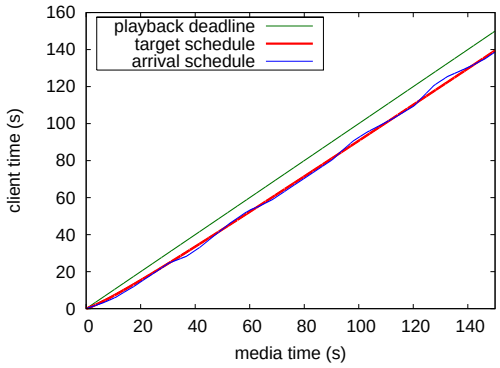
Figure 16 illustrates the decrease in the initial decoder buffer fullness $F^d_{\min}(R)$ as R changes from r_c to r_a. In particular, it depicts the coding schedule for a given bit stream, as well as upper and lower bounds, denoted Tube I and Tube II, corresponding to two leaky buckets with leak rates r_c and r_a respectively, both containing the coding schedule. Tube II is smaller than Tube I, since the minimum size $B_{\min}(R)$ of a leaky bucket containing a given stream is decreasing in the leak rate R [16]. Likewise, the initial decoder buffer fullness $F_{\min}(R)$ is decreasing in R [16]. Hence the playback deadline for frame 0 can begin as early as client time $t_{0\_II} = F^d_{\min}(r_a)/r_a$, instead of $t_{0\_I} = F^d_{\min}(r_c)/r_a$. From there, the playback deadline advances at $1/\nu$ seconds of client time per second of media time.

B. Controller Initialization

As illustrated in Figure 16, the target schedule starts at the same time as the playback deadline and grows according to a predefined function. The controller attempts to control the upper bound of Tube I to the target schedule. Initially the upper bound of Tube I is above the target schedule (and is indeed above the playback deadline, though we know that this is safe). Hence, when the playback starts, the controller would try to close the gap by decreasing the coding rate. This, however, would not be desirable because the current coding rate is already lower than the arrival rate to allow the client

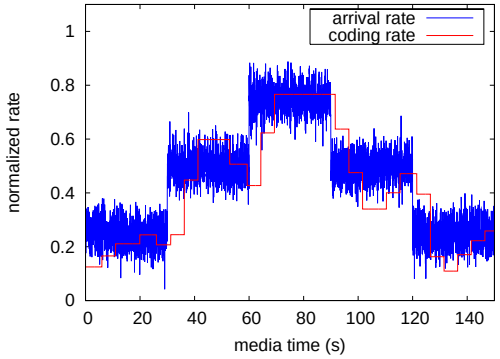


(a) rate vs. time

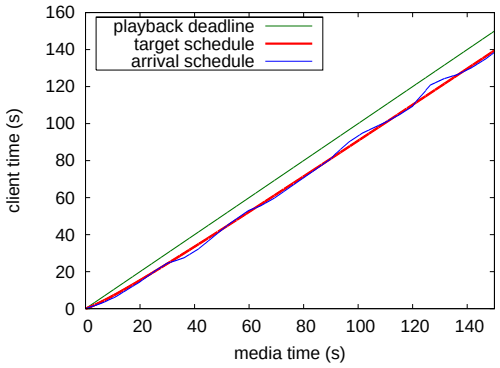


(b) schedule vs. time

Fig. 14. Time response simulation.



(a) rate vs. time



(b) schedule vs. time

Fig. 15. Time response simulation.

buffer to grow. Further reduction of the coding rate would not be proper. To avoid this effect, we initialize the controller when the upper bound of Tube I exceeds the target schedule i.e., at point B in Figure 16. Point B can be found analytically, but in practice there is no need to explicitly solve for it. The controller can be initialized as soon as the upper bound of Tube I exceeds the target.

C. Exponential Averaging of the Arrival Rate

As shown in the performance studies in Section III-F, using the average arrival rate from a low pass filter (instead of the instantaneous arrival rate) helps to reduce coding rate oscillations. This section details our exponential averaging algorithm for the arrival rate.

Let $\tilde{r}_a(k)$ and $r(k)$ be the average arrival rate and the instantaneous arrival rate, respectively, when packet k is received. Note that unlike the controlling operation, the rate averaging operation may be performed after the arrival of every *packet*, rather than after the arrival of every *frame*. Hence we use the discrete packet index k rather than the frame index n. Instead of using the widely adopted exponentially weighted moving average (EWMA)

$$\tilde{r}_a(k) = \beta(k)\tilde{r}_a(k-1) + (1-\beta(k))r_a(k) \qquad (49)$$

with constant $\beta(k) = \beta$, we perform the exponential averaging more carefully. In our algorithm, the factor $\beta(k)$ is not constant, but varies according to the packets' interarrival gaps. Our algorithm has several advantages over the EWMA algorithm with constant $\beta(k)$. First, the estimate of the average arrival rate $\tilde{r}_a(k)$ goes to zero naturally as the gap since the last packet goes to infinity, rather than being bounded below by $\beta\tilde{r}_a(k-1)$. Second, the estimate of the average arrival rate $\tilde{r}_a(k)$ does not go to infinity as the gap since the last packet goes to zero. This is especially important, since packets often arrive in bursts, causing extremely high instantanous arrival rates. And finally, the estimate of the average arrival rate $\tilde{r}_a(k)$ does not over-weight the initial condition, as if it represented the infinite past. This is especially important in the early stages of estimation.

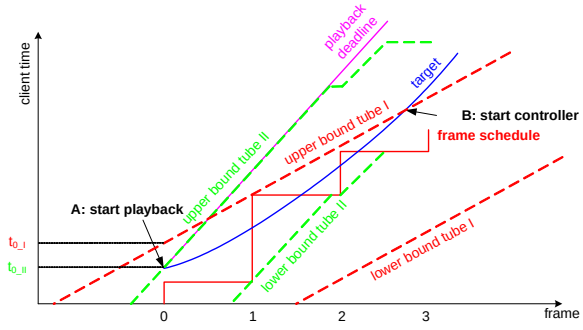


Fig. 16. Leaky buckets (buffer tubes) for various transmission rates.

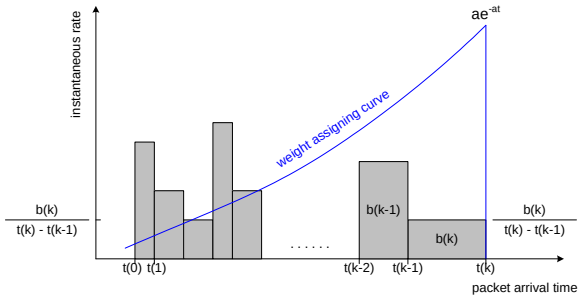


Fig. 17. Exponential averaging.

As in (11), we define the instantaneous arrival rate after packet k as

$$r_a(k) = \frac{b(k)}{t_a(k) - t_a(k-1)}, \tag{50}$$

where here $b(k)$ denotes the size of packet k and $t_a(k)$ denotes the arrival time of packet k. We extend the discrete time function $r_a(k)$ to the piecewise constant continuous time function $r_a(t)$ by

$$r_a(t) = r_a(k) \quad \text{for all } t \in (t_a(k-1), t_a(k)], \tag{51}$$

as illustrated in Figure 17. Then we filter the function $r_a(t)$ by the exponential impulse response $\alpha e^{-\alpha t}$, $t \geq 0$, for some time constant $1/\alpha$:

$$\tilde{r}_a(k) = \frac{\int_{t(0)}^{t(k)} r_a(t')\alpha e^{-\alpha(t(k)-t')}dt'}{\int_{t(0)}^{t(k)} \alpha e^{-\alpha(t(k)-t')}dt'}. \tag{52}$$

(Here and in the remainder of this subsection we suppress the subscript from the arrival time $t_a(k)$.) Noting that $\int_t^\infty \alpha e^{-\alpha t'}dt' = e^{-\alpha t}$, the denominator integral can be expressed $1 - e^{-\alpha(t(k)-t(0))}$. Now, we split the range of the numerator integral into ranges $(t(0), t(k-1)]$ and $(t(k-1), t(k)]$ to obtain a recursive expression for $\tilde{r}_a(k)$ in terms of $\tilde{r}_a(k-1)$ and $r_a(k)$,

$$\begin{aligned}
&\tilde{r}_a(k) \\
&= \frac{1 - e^{-\alpha[t(k-1)-t(0)]}}{1 - e^{-\alpha[t(k)-t(0)]}} e^{-\alpha[t(k)-t(k-1)]} \tilde{r}_a(k-1) \\
&\quad + \frac{1 - e^{-\alpha[t(k)-t(k-1)]}}{1 - e^{-\alpha[t(k)-t(0)]}} r_a(k) \qquad (53) \\
&= \beta(k)\tilde{r}_a(k-1) + (1-\beta(k))r_a(k), \qquad (54)
\end{aligned}$$

where

$$\beta(k) = \frac{e^{-\alpha[t(k)-t(k-1)]} - e^{-\alpha[t(k)-t(0)]}}{1 - e^{-\alpha[t(k)-t(0)]}}. \tag{55}$$

Note that $\beta(k)$ is numerically stable as k goes to infinity. However, as the gap $\delta = t(k) - t(k-1)$ goes to zero, $1 - \beta(k)$ goes to zero while $r_a(k)$ goes to infinity. Their product, however, is well behaved. Indeed,

$$\begin{aligned}
\tilde{r}_a(k) &= \frac{1 - e^{-\alpha[t(k-1)-t(0)]}}{1 - e^{-\alpha[\delta+t(k-1)-t(0)]}} e^{-\alpha\delta} \tilde{r}_a(k-1) \\
&\quad + \frac{1 - e^{-\alpha\delta}}{1 - e^{-\alpha[t(k)-t(0)]}} \frac{b(k)}{\delta} \qquad (56) \\
&\to \tilde{r}_a(k-1) + \frac{\alpha b(k)}{1 - e^{-\alpha[t(k)-t(0)]}} \qquad (57)
\end{aligned}$$

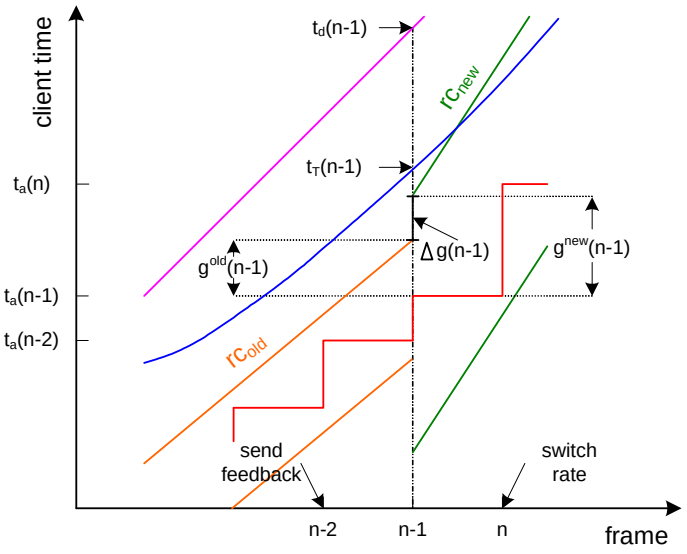


Fig. 18. Buffer tube change and control target adjustment.

as $\delta \to 0$, using l'Hôpital's rule. Thus (57) is the update rule in the case when $t(k) = t(k-1)$.

D. 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 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 18, 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 $\Delta^{\max} g(n-1)$ supplied to it by the client. The client supplies $\Delta^{\max} g(n-1)$ 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 $g^{new}(n-1) - g^{old}(n-1) \leq \Delta^{\max} g(n-1)$, where $g^{new}(n-1)$ and $g^{old}(n-1)$ are illustrated in Figure 18. The constraint given by $\Delta^{\max} g(n-1)$ is not applied if it is a switch down in rate.

The client chooses $\Delta^{\max} g(n-1)$ to limit (its prediction of) what the upper bound would be at time $t_a(n-1)$ if the new

coding rate were in effect, namely,

$$\begin{aligned} & t_b^{new}(n-1) \\ & \approx \quad t_b(n-2) + \frac{\hat{r}_c(n-1)}{f\tilde{r}_a} + \frac{\Delta g(n-1)}{\tilde{r}_a} \qquad (58) \\ & \leq \quad t_T(n-1) + p[t_d(n-1) - t_T(n-1)]. \qquad (59) \end{aligned}$$

That is, the client chooses $\Delta^{\max} g(n-1)$ to limit $t_b^{new}(n-1)$ so that it would be 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 $p = 1/3$.

E. Control Target Adjustment

When a frame with a new average coding rate $\hat{r}_c(n)$ arrives at the client at time $t_a(n)$, there is a shift in the upper bound. Real scalable stream data (cf. Figure 20) shows that this shift can be on the order of seconds and hence, rather than being negligible, can be confusing to the controller. If the shift is upward, for example, the controller will immediately try to reduce the coding rate $r_c(n+2)$. If the shift is downward, on the other hand, the controller will immediately try to increase the coding rate $r_c(n+2)$. Either way is probably not good; the intention is that $\hat{r}_c(n)$ will be maintained unless there is a disturbance in the arrival rate. Our solution is to introduce a simultaneous shift in the control target schedule equal to $\Delta g(n-1)/\tilde{r}_a$, where $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$ is the actual shift in the upper bound (in bits) at frame $n-1$ computed at the server, as illustrated in Figure 18. 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 have a tendency to approach 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 the logarithmic target schedule $t_T = t_d - \frac{b}{a}\ln(at_d + 1)$ (where $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$), according to (24) the slope at media time τ_d is

$$s = \frac{dt_T}{d\tau_d} = \frac{1}{\nu} - \frac{b}{a(\tau_d - \tau_{d0}) + \nu}. \qquad (60)$$

If we define d as the distance between the playback deadline and the target schedule, namely

$$d = \frac{b}{a}\ln\left(a\left(\frac{\tau_d - \tau_{d0}}{\nu}\right) + 1\right), \qquad (61)$$

then the slope may be expressed as a function of d,

$$s = \frac{1}{\nu} - \frac{b}{\nu e^{(a/b)d}}. \qquad (62)$$

Hence whenever d is the distance between the playback deadline and the control target, we set the slope of the control target to s in (62). Specifically, if $t_{\hat{T}}(n)$ is the control target at frame n after the shift, then we reset $t_{\hat{T}}(n-1)$ to be $T_{\hat{T}}(n) - s/f$. We then use $t_{\hat{T}}(n)$ and $t_{\hat{T}}(n-1)$ in place of $t_T(n)$ and $t_T(n-1)$ to compute the error vector $\mathbf{e}(n)$ in (37). The resulting error vector is then used to compute the ideal coding rate in (40).

For the two-piece linear target schedule, the slope is easy to compute by using a predefined time period over which the control target schedule is expected to return to the target schedule. The slope of the control target schedule can then be computed from the distance d and the period. We set the period to 50 seconds in our experiments.

V. Implementation Details

This section highlights implementation details on both the sender and the receiver side.

A. Generation of Virtual Streams

In our implementation, a fine grained scalable (FGS) stream comprises a set of data units, each tagged by a Lagrange multiplier λ representing the per-bit decrease in distortion if the data unit is received by the client. If the λ for the data unit is above a threshold, then the data unit is included in a virtual stream corresponding to that threshold. Each threshold corresponds to an overall number of bits and hence an average coding rate for the virtual stream. In our experiments, we generate $N = 50$ virtual streams. A threshold is chosen for each stream such that the resulting streams have coding rates that are uniformly spaced in the log domain between lower and upper bounds.

During streaming, when the server reads a data unit from the media file, it includes the data unit in the virtual stream currently being transmitted if its Lagrange multiplier λ is above the threshold for the stream.

B. Leaky Bucket Computations at the Sender

For each virtual stream, leaky bucket parameters $(R, B_{\min}(R), F^d_{\min}(R))$ are precomputed off line for $R = R_{avg}$ and $R = R_{\max}$, where $R_{avg} = r_c$ is the average coding rate of the stream, and $R_{max} = 2r_c$. These leaky bucket parameters are sent to the client in a preamble.

In addition, during streaming the server performs on-line leaky bucket simulations for each stream. Specifically, whenever the server reads a data unit from the media file, it determines the virtual streams to which the data unit belongs, using the Lagrange multiplier of the data unit and the list of thresholds for each stream. The sender then updates, for the determined streams, the states of those leaky buckets having leak rates equal to an average coding rate R_{avg}, using (2) and (3). Once all the data units in a frame are read from the media file, the sender computes $g(n) = B_{\min}(R_{avg}) - B^e(n)$ for each of the virtual streams. On a stream switch (i.e., $\hat{r}_c(n) \neq \hat{r}_c(n-1)$), the gap $g^{new}(n)$ for the new stream is transmitted to the client along with $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$ as described below. It is easy to see that the cost of updating the leaky bucket states is quite low. However, it is also possible to precompute these values and store them with each data unit in the media file.

C. Initial Coding Rate Selection

At the beginning of a streaming session, the sender needs to have some knowledge of the available network bandwidth so that it can choose an initial coding rate (usually half of the bandwidth). The bandwidth estimate can be drawn from proactive measurements, using approaches such as packet pair [20], path chirp [21], etc., or reactive approximations based on history values. The exact form of the initial bandwidth estimation is beyond the scope of this work.

D. Coding Rate Switching

The rate control feedback from the client contains the frame number at which feedback is generated (e.g., $n-2$ in the previous section) and the maximum allowable shift of the upper bound in bits (e.g., $\Delta^{max} g(n-1)$ in the previous section). If the sender finds a suitable coding rate and makes a switch at frame n, it will transmit three values to the client along with the frame: the new coding rate $\hat{r}_c^{new}(n)$, the current gap to the upper bound $g^{new}(n)$, and the shift $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$. With this information, the client can properly adjust its control target schedule as well as its upper bound. Note that coding rate switching always happens at the beginning of a new frame, never inside a frame.

E. Optimal Rate Control at the Client

Whenever a new coding rate starts, the client receives the value $g(n)$ along with the new frame. The values of $g(n)$ for successive frames can be then inferred by the client itself based on the coding rate $\hat{r}_c(n)$ and the frame size $b(n)$. The client records the arrival frame time $t_a(n)$, calculates the buffer tube upper bound $t_b(n)$ and then computes the deviation $e(n)$. If there is a coding rate switch, it will also compute the buffer tube shift and adjust the control target schedule accordingly. Then $e(n)$ is feed to the optimal rate controller, which then outputs a desired new coding rate. The latest new coding rate is fed back to the sender whenever there is a feedback opportunity, which could be generated at regular intervals or on-demand.

VI. 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.[7] The test video is coded using a variant of MPEG-4 FGS [7], 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.

[7]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.

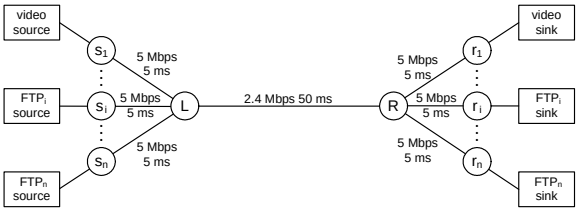


Fig. 19. ns-2 Simulation network setup.

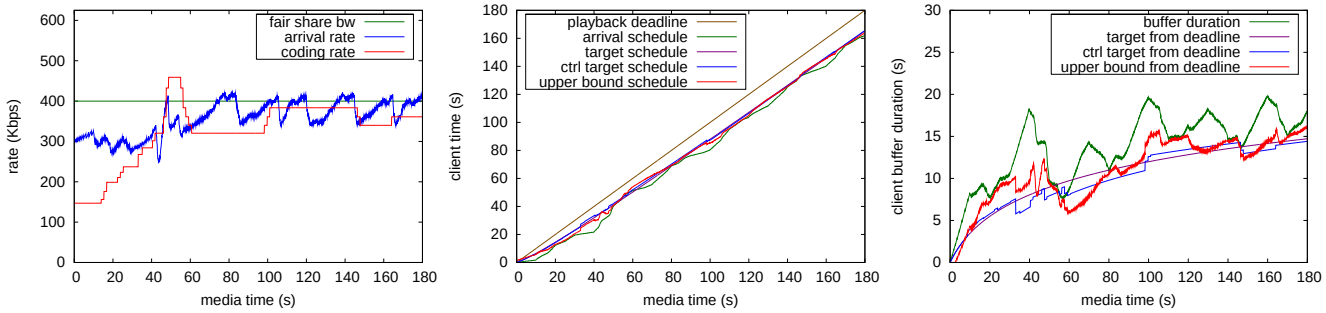
	client time	# of FTPs	fair share BW
Constant Bandwidth	0–180 s	5	400 Kbps
Variable Bandwidth	0–30 s	2	800 Kbps
	30–60 s	5	400 Kbps
	60–90 s	11	200 Kbps
	90–130 s	5	400 Kbps
	130–180 s	2	800 Kbps

TABLE II
BANDWIDTH AVAILABLE TO THE STREAMING SESSION

Using the popular network simulator ns-2 [22], we set up a simple network environment as shown in Figure 19. 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 ($2 \le i \le n$). 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 II. Experiments are carried out using both TCP and TFRC [15] as alternative transport layer protocols.

A. Startup Delay

Figure 20 shows the startup delay as a function of the transmission/arrival rate r_a, for two streams, one at average coding rate $r_c = r_a$, and another at $r_c = 0.5r_a$. Specifically, for the virtual stream with average coding rate r_c, let $F^d_{\min}(R|r_c)$ denote the minimum initial decoder buffer size computed for a leaky bucket with leak rate R. (We know that for a fixed r_c, this function decreases in R). The top curve in the figure shows the startup delay $F^d_{\min}(r_a|r_a)/r_a$, when the coding rate is chosen to match the transmission rate. The middle curve shows the startup delay $F^d_{\min}(0.5r_a|0.5r_a)/r_a$, when the coding rate is chosen to be half of the transmission rate, but the initial decoder buffer fullness is based on the coding rate. And the bottom curve shows the startup delay $F^d_{\min}(r_a|0.5r_a)/r_a$, when the coding rate is chosen to be half of the transmission rate, and the initial decoder buffer fullness is based on the transmission rate, thus further reducing the startup delay. The three curves in the figure are calculated using leaky bucket simulations with the virtual streams' coding schedules, but we notice that the bottom curve matches nicely with experimental results from our ns-2 simulations at rates at 150 Kbps, 300 Kbps, 450 Kbps, 600 Kbps, 750 Kbps and 900 Kbps, all of which have delay much lower than 1 second.



(a) rate vs. time (b) schedule vs. time (c) buffer vs. time

Fig. 21. Constant bandwidth over TCP.

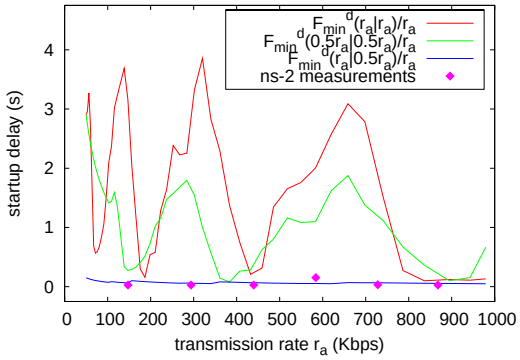
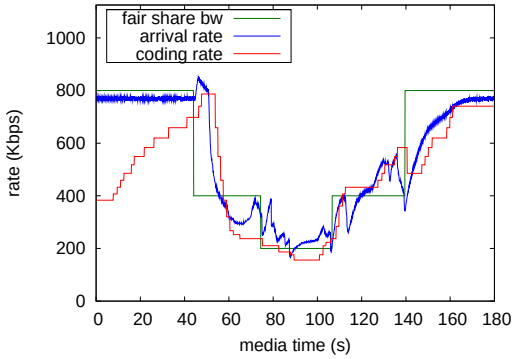
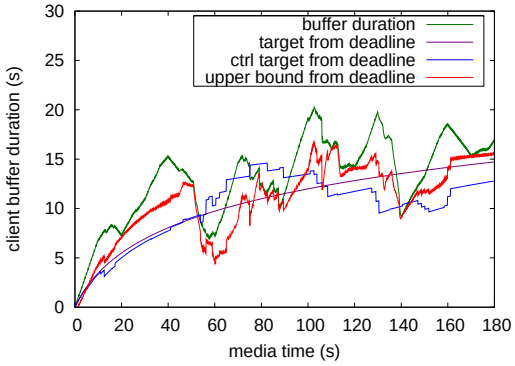


Fig. 20. Startup delay vs. transmission rate.



(a) rate vs. time



(b) buffer vs. time

Fig. 22. Variable bandwidth over TCP.

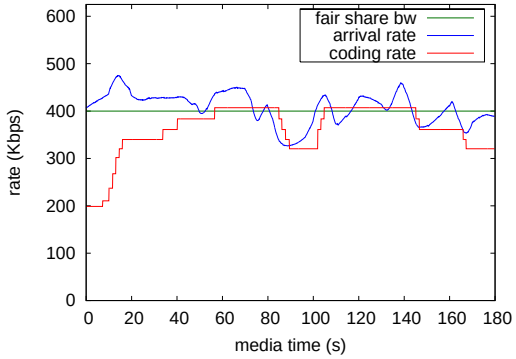
B. Constant vs. Variable Bandwidth

Figures 21 and 22 show results using TCP as the transport protocol, under constant and variable bandwidth conditions, respectively. In either case, 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.

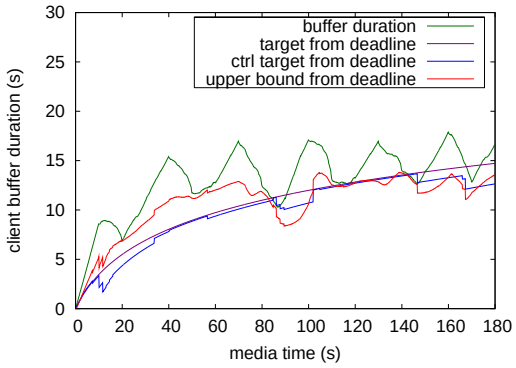
Figure 21(c) presents essentially the same information as Figure 21(b), but plots the *difference* between the playback deadline and 1) the arrival schedule, 2) the buffer tube upper bound schedule, 3) the control target schedule, and 4) the logarithmic target schedule, respectively. Note that the gap between the playback deadline and the arrival schedule is the client buffer duration. In the remainder of this paper, we present all schedules using this format.

C. TFRC vs. TCP Transport

Figures 23 and 24 show results using TFRC as the transport protocol. It is interesting to see that although TFRC yields more stable arrival rates than TCP (consistent with the design philosophy of TFRC and revealed by comparing Figures 21(a) and 22(a) with Figures 23(a) and 24(a)), the traces of the coding rates under TFRC and TCP are similar. Note that when we use TFRC as the transport protocol for the media stream, we also use TFRC for the cross traffic (i.e., FTP over TFRC). The reason is that despite its name, TFRC appears to take somewhat more than its fair share of bandwidth when competing with TCP. Using TFRC cross traffic with the TFRC media stream ensures that corresponding TCP and TFRC media streams receive approximately the same bandwidth.

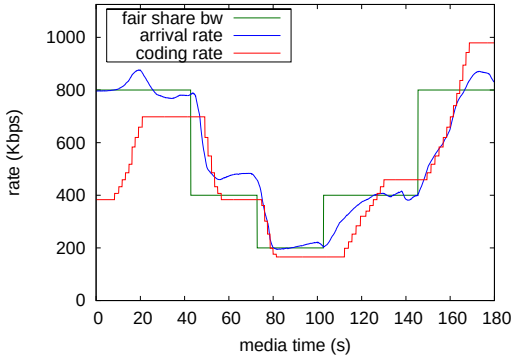


(a) rate vs. time

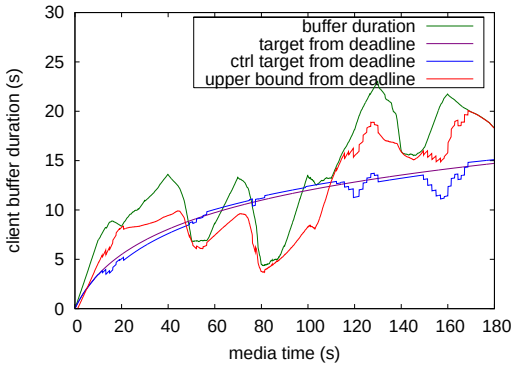


(b) buffer vs. time

Fig. 23. Constant bandwidth over TFRC.



(a) rate vs. time



(b) buffer vs. time

Fig. 24. Variable bandwidth over TFRC.

D. Two-piece linear vs. logarithmic target schedule.

Figures 25, 26, 27 and 28 show results using TCP and TFRC as the transport protocol with the two-piece linear target schedule. Compared to the logarithmic target schedule, the two-piece linear target schedule holds the initial lower coding rate for a longer period (thus sacrificing more quality) in the startup phase, so that the client buffer can build up more quickly. After the startup phase, there is no further need to sacrifice quality to maintain the client buffer level. In contrast, with the logarithmic target schedule, there is some sacrifice in quality over the entire streaming session, although the sacrifice diminishes gradually as the slope of the schedule approaches a constant.

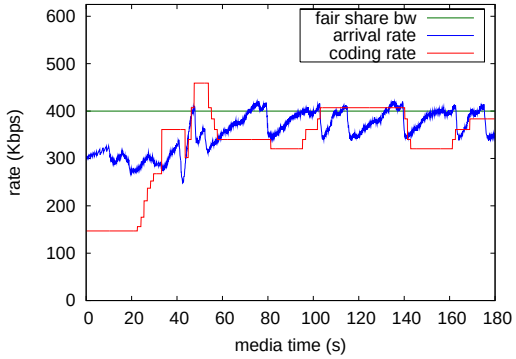
It is clear that both target schedules work well under either constant bandwidth or variable bandwidth situations. The choice, which reflects a balance between quality and buffer level in the startup phase as well as asymptotically, can be deferred to particular applications.

It should be noted that if the client buffer has a limited size (in bytes), then the target schedule can be designed, if desired, to take this size into account. If the client buffer becomes full, then the client must stop accepting packets from the network. This reduces the arrival rate and consequently, ultimately reduces the average coding rate as the controller attempts to get the target duration into the buffer. The average coding rate is ultimately capped at the buffer size in bytes divided by the buffer duration. Thus, if the buffer size in bytes is fixed, then it may be desirable to design the target schedule so that it corresponds to a maximum buffer duration yielding a good compromise between average coding rate and robustness.

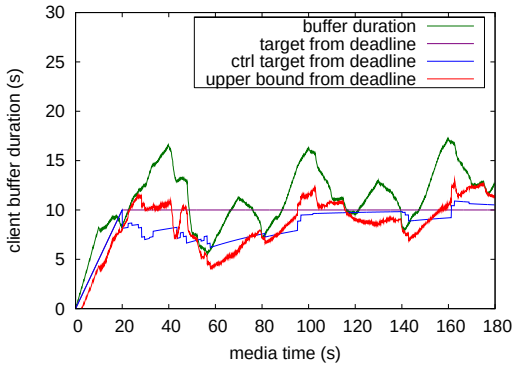
E. Controller Performance Tuning

1) Tuning σ: The performance figures show significant deviation of the buffer tube upper bound from the control target, which is especially obvious in the variable bandwidth case. It is clear from our controller design rationale that we can reduce this deviation by decreasing the σ value. 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 coding rate smoothness, since the corresponding term in the cost function will be weighted less. Figure 29 shows simulation results with $\sigma = 500$ under the same network conditions as in Figure 21. 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,

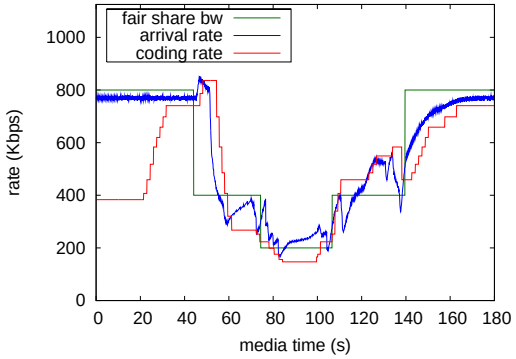


(a) rate vs. time

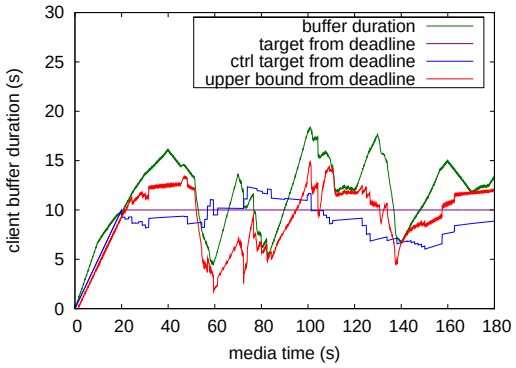


(b) buffer vs. time

Fig. 25. Constant bandwidth over TCP with the two-piece linear target schedule.



(a) rate vs. time



(b) buffer vs. time

Fig. 26. Variable bandwidth over TCP with the two-piece linear target schedule.

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.

2) Smoothing $e(k)$: The frame arrival time t_a, which is used to compute the controller input, is the client time at which a frame is completely received. This time could increase significantly if part of the frame arrives in retransmitted packets. When the controller is fed with $e(n)$, which is a deviation computed from the arrival time, the controller may misinterpret the increase and may generate oscillatory output over time. Note that this variation in arrival time is different from the variation in transmission rate and is not specifically addressed in our mathematical model. Thus, we need an additional mechanism to deal with it.

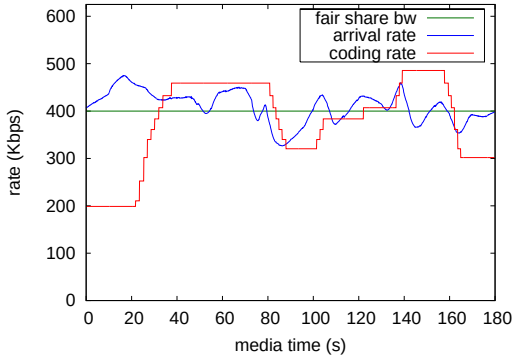
A straightforward approach is to apply our exponential averaging method on $e(k)$, which will certainly smooth out spiky values of the deviation and let the controller react upon the long time trend. Let $\tilde{e}(n)$ be a smoothed sequence input to the controller instead of $e(n)$, specifically

$$\tilde{e}(n) = \frac{e^{-\alpha} - e^{-\alpha n}}{1 - e^{-\alpha n}} \tilde{e}(n-1) + \frac{1 - e^{-\alpha}}{1 - e^{-\alpha n}} e(n). \qquad (63)$$

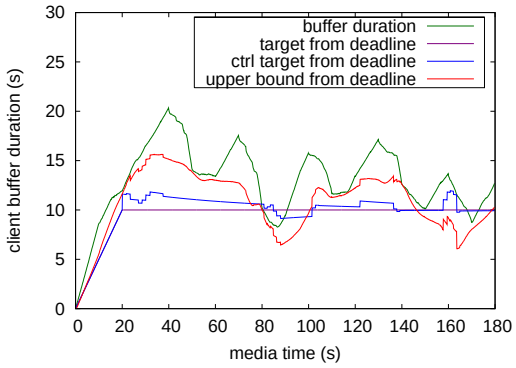
We choose $\alpha = 1/f$ (the frame rate) to focus on history values in the last second, which will also allow $\tilde{e}(n)$ to follow the trend promptly when a significant change in bandwidth occurs. All results reported in this section use this mechanism.

F. Comparison with Benchmark Algorithm

As a benchmark, we compare our buffer management algorithm to the windowing algorithm in [10] (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.

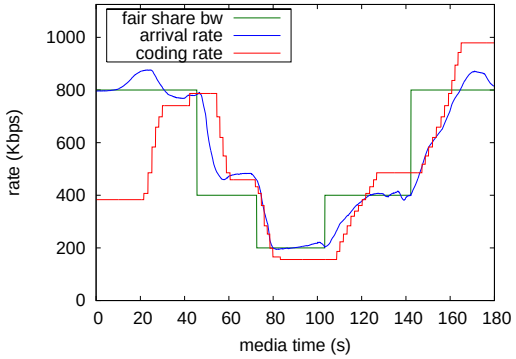


(a) rate vs. time

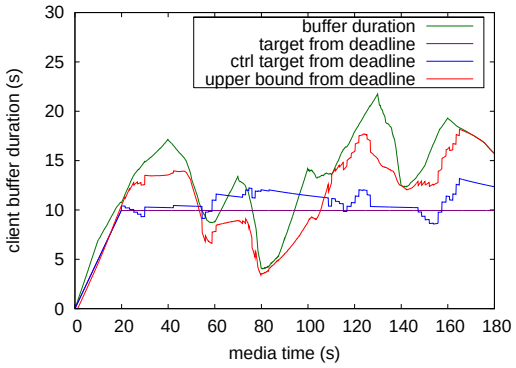


(b) buffer vs. time

Fig. 27. Constant bandwidth over TFRC with the two-piece linear target schedule.



(a) rate vs. time



(b) buffer vs. time

Fig. 28. Variable bandwidth over TFRC with the two-piece linear target schedule.

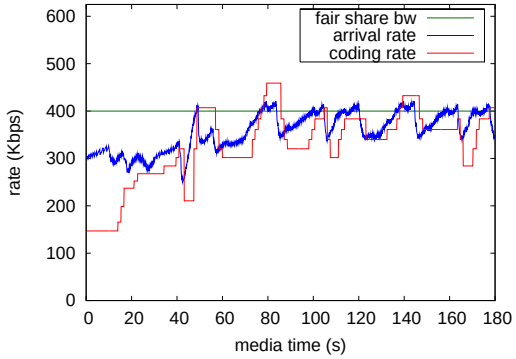
Although conceptually simple and sound, the benchmark algorithm has two disadvantages. First, it does not send out data units in the order in which they appear in the media file (i.e., decoding order). This demands resources (e.g., caching large segments of data) that may be incompatible with high performance 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, 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.

Figure 30 shows the buffer tube containing the coding schedule for a video sequence consisting of *Akiyo*, *Stefan* and *Foreman* (10 s each) at an average coding rate of 500 Kbps. Note that *Akiyo* requires relatively few bits per second of media time, and *Stefan* requires relatively more bits per second of media time, to achieve quality similar to *Foreman*. Thus if the three subsequences are all coded with roughly the same number of bits per second of media time, *Akiyo* will have higher quality, and *Stefan* will have lower quality, relative to *Foreman*.

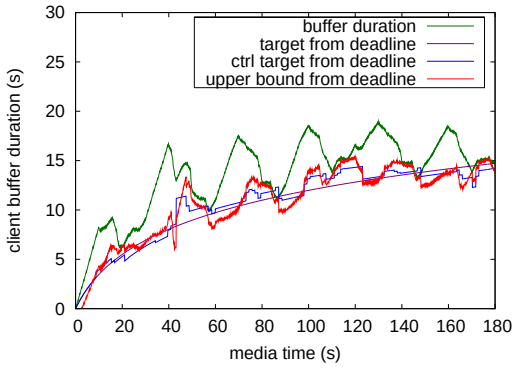
Figures 31 and 32 show the PSNR results after streaming with a constant bandwidth of 400 Kbps over TCP and TFRC. Our optimal control algorithm with either target schedule is much smoother in terms of PSNR compared to the benchmark algorithm. Note that even with optimal control, the PSNR value shows a repetitive pattern over the entire session, instead of a constant value. This happens because the scalable codec we use in the experiments is a bit plane codec. There could be one bit plane difference (about 6 dB in PSNR) between frames of the same coding rate.

G. Comparison with Constant Bit Rate Algorithm

The CBR algorithm is 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



(a) rate vs. time



(b) buffer vs. time

Fig. 29. 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 21.

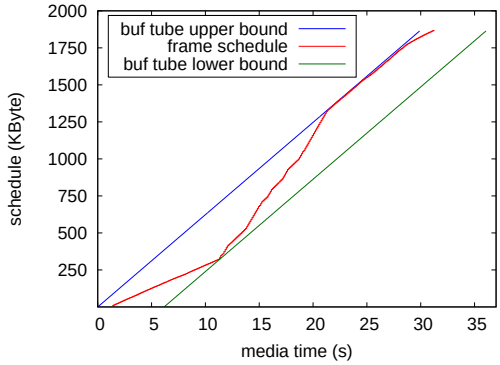
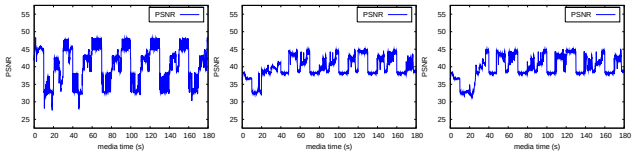


Fig. 30. Coding schedule of a mixed video sequence (Akiyo, Stefan and Foreman) at an average coding rate of 500 Kbps.



(a) benchmark algorithm

(b) optimal control algorithm with logarithmic target

(c) optimal control algorithm with linear target

Fig. 31. PSNR with constant bandwidth (400 Kbps) over TCP.

for high-action content (which is hard to encode). The quality oscillation is significant over constant bandwidth channels as shown in Figure 33 and 34. The experimental settings for these figures are the same as for Figures 31 and 32, respectively.

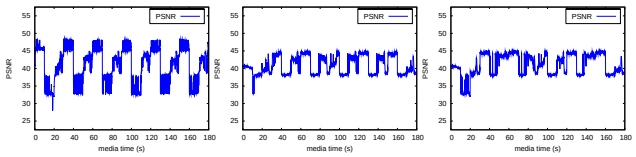
H. Rate-Distortion Comparison

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 the TCP protocol is used for all experiments. The average distortion in terms of PSNR over each session is computed on the client side and plotted in Figure 35. 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. We can also see that, in general, the linear target schedule has slightly better performance than the logarithmic target schedule. This is understandable since the quality sacrifice happens only during the initial period for the linear target schedule, while it spreads over the entire streaming session for the logarithmic target schedule. The reason that the CBR algorithm has worse performance than the benchmark algorithm is also 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.

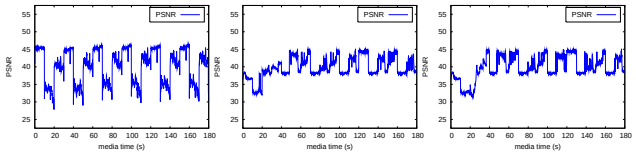
VII. Multiple Bit Rate Streaming

Multiple bit rate (MBR) streaming is a network adaptive technique that is widely used in commercial streaming media systems (e.g. Windows Media 9 Series [1]). In MBR streaming, in contrast to scalable streaming, the content is encoded into several (typically at most 5–7) independent streams at different coding rates. Often, each stream is optimized for a common type of network connection (e.g., dial-up, DSL, cable). During an MBR streaming session, the proper coding rate is dynamically selected based on the available network bandwidth, with the goal of achieving the maximum possible quality under the condition of uninterrupted playback. It is easy to see that MBR streaming is analogous to scalable streaming. Indeed MBR streaming can be viewed as a special



(a) benchmark algorithm (b) optimal control algorithm with logarithmic target (c) optimal control algorithm with linear target

Fig. 32. PSNR with constant bandwidth (400 Kbps) over TFRC.



(a) CBR algorithm (b) optimal control algorithm with logarithmic target (c) optimal control algorithm with straight target

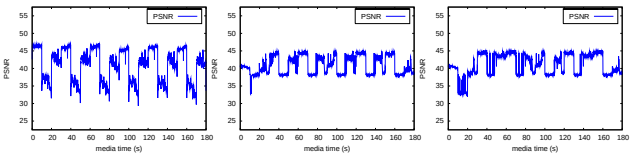
Fig. 33. PSNR with constant bandwidth (400 Kbps) over TCP.

case of scalable streaming with a limited number of coding rates available. Hence, our optimal control approach should be applicable to this case.

There are, however, several differences that complicate MBR streaming, which need to be carefully addressed. First, as just mentioned, in MBR streaming there are only a limited number of coding rates available. This coarse quantization of the desired coding rate introduces a significant nonlinearity into the closed loop system. In fact, the large gaps between the available coding rates introduce oscillations. For example, if two neighboring coding rates straddle a constant arrival rate, the controller will oscillate between the two coding rates in an attempt to keep the client buffer at a target level.

Second, in MBR streaming the coding rate cannot be switched at an arbitrary time. In fact, before the server can switch to a new stream, it must wait for the next clean point (e.g., *I* frame) in the new stream, which could be five or ten seconds away. Thus, the old coding rate may continue for quite a while before it changes to the new coding rate. From the controller's perspective, this long random extra delay tends to destabilize the closed-loop system.

Third and finally, in MBR streaming, server performance issues are critical. The commercial-grade streaming media systems that use MBR streaming do so because of the minimal computational load that it imposes on the server compared to scalable streaming. Thus, for MBR streaming it is important to keep almost all computation and state maintenance on the client side. In particular, the server will not be able to update the leaky bucket information for each stream, as we have proposed in previous sections. Instead, the client must use some mechanism for estimating and maintaining this information.



(a) CBR algorithm (b) optimal control algorithm with logarithmic target (c) optimal control algorithm with straight target

Fig. 34. PSNR with constant bandwidth (400 Kbps) over TFRC.

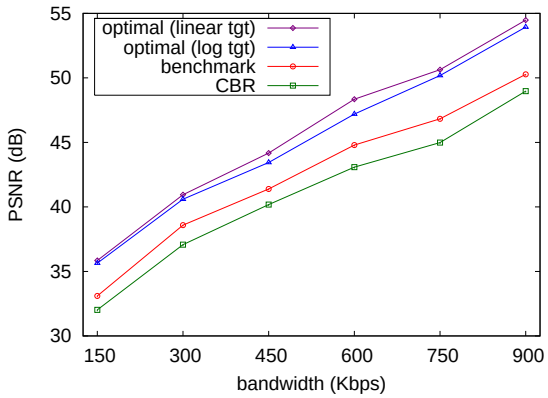


Fig. 35. Rate-Distortion comparison.

A. Conservative Up-Switching

In this subsection we discuss a technique to help stabilize the control system and reduce steady state oscillations to a period of at least a minute. With this technique, rapid down-switching is permitted. In fact, we reduce the value of σ from 4000(2000) to 1000(500), changing the balance between responsiveness and smoothness of the coding rate in favor of rapid switching response. However, only conservative up-switching is permitted. Conservative up-switching ensures that spurious changes in coding rate do not occur, and that oscillations in the coding rate have a low frequency. In particular, conservative up-switching reduces the oscillations between two adjacent but widely spaced MBR coding rates, one above the arrival rate and one below the arrival rate.

The method behind conservative up-switching is to establish a conservative limit on how high the coding rate can be raised above the arrival rate. If the current coding rate is below the arrival rate, and the client buffer duration begins to increase above its target level, then the coding rate can be switched up to a new coding rate above the arrival rate only if the new coding rate is below the conservative limit. When the client buffer duration begins at the target level, the conservative limit is equal to the arrival rate. However, as the client buffer duration increases, the conservative limit increases as well. Thus, if the current coding rate is below the arrival rate, and the next higher coding rate is above the arrival rate, then it will be possible to switch up to the next higher coding rate only after the client buffer duration has increased sufficiently so that the conservative limit rises above the higher coding rate. Once the coding rate is switched up to the higher coding

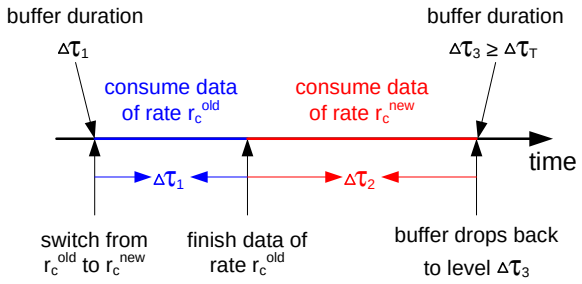


Fig. 36. Conservative rate up-switching.

rate, the client buffer begins to drain since the coding rate is then above the arrival rate. Eventually, when the buffer drains back below its target level, the controller will rapidly switch the coding rate back down to the coding rate below the arrival rate.

Given the current client buffer duration, the conservative limit is set to a value such that if the coding rate is switched up to a new coding rate at this value, the client buffer would take at least Δt seconds of client time to drain back to the target level. Thus, the mechanism ensures that the period of oscillation will be at least Δt seconds. In our experiments, we set Δt to be 60 seconds.

Figure 36 shows how we compute the conservative limit. Let $\Delta\tau_1$ be the client buffer duration (in media time) at the moment that the coding rate is switched up from r_c^{old} to r_c^{new}. Thus $\Delta\tau_1$ is the number of seconds of content that will be consumed at the old coding rate r_c^{old} before content at the new coding rate begins to be consumed. (For simplicity we assume that all of the content in the client buffer at the time of the switch is coded at rate r_c^{old}.) Let $\Delta\tau_2$ be the number of seconds of content that is consumed at the new coding rate r_c^{new} before the client buffer duration drops to some level $\Delta\tau_3$ seconds (in media time), greater than the target level $\Delta\tau_T$. The duration of this phase is determined such that the total time since the switch is exactly $\Delta t = (\Delta\tau_1 + \Delta\tau_2)/\nu$ seconds (in client time). Now, the number of bits that arrive in this time is $r_a\Delta t = r_c^{new}(\Delta\tau_2 + \Delta\tau_3) \geq r_c^{new}(\Delta\tau_2 + \Delta\tau_T) = r_c^{new}(\nu\Delta t - \Delta\tau_1 + \Delta\tau_T)$, or

$$r_c^{new} \leq \frac{r_a\Delta t}{\nu\Delta t - \Delta\tau_1 + \nu\Delta t_T}, \tag{64}$$

where Δt_T is the target buffer duration in client time. The parameter Δt can be tuned to yield the desired behavior. A large Δt means that up-switching will be more conservative, while a smaller Δt means that up-switching will be more prompt. In our implementation, Δt is set to 60 seconds while the target Δt_T is typically about 10 seconds.

B. Buffer Tube Upper Bound Estimation

In Section V-D we specified that the server sends three values to the client at the beginning of each change in coding rate: the new coding rate $\hat{r}_c^{new}$, the current gap to the upper bound $g^{new}(n)$, and the control target shift $\Delta g(n-1) = g^{new}(n-1) - g^{old}(n-1)$. The server computes the latter two values by running a leaky bucket simulator for each coding rate. The client continues to update $g(n)$ for the new coding rate by running its own leaky bucket simulator for the new coding rate. That is, beginning with the initial condition $F^e(n) = B - b(n) - g^{new}(n)$, for each successive frame the client computes

$$B^e(n) = F^e(n) + b(n) \tag{65}$$
$$F^e(n+1) = \max\{0, B^e(n) - \hat{r}_c/f(n)\}, \tag{66}$$

where

$$f(n) = \frac{1}{\tau(n+1) - \tau(n)} \tag{67}$$

is the instantaneous frame rate, as in (2), (3), and (4). From this, the client can compute

$$g(n) = B - B^e(n) \tag{68}$$

for each frame.

However, if the server is unable to simulate the leaky buckets and cannot send $g^{new}(n)$ to the client, then the client must estimate this information for itself. In this case we recommend that the client estimates $g^{new}(n)$ as an upper bound such as $\hat{g}^{new}(n) = B - b(n) \geq g^{new}(n)$. Then, beginning with initial condition $\hat{F}^e(n) = B - b(n) - \hat{g}^{new}(n)$ (which equals 0 in this case), for each successive frame the client computes

$$\hat{B}^e(n) = \hat{F}^e(n) + b(n) \tag{69}$$
$$\hat{F}^e(n+1) = \max\{0, \hat{B}^e(n) - \hat{r}_c/f(n)\}, \tag{70}$$

as well as

$$\hat{g}(n) = B - \hat{B}^e(n). \tag{71}$$

It is easy to see by induction that $\hat{F}^e(n) \leq F^e(n)$, $\hat{B}^e(n) \leq B^e(n)$, and $\hat{g}(n) \geq g(n)$. Moreover, these bounds each become tighter by $\delta(n) = \hat{r}_c/f(n) - B^e(n)$ whenever $\delta(n) > 0$, i.e., whenever $F^e(n+1)$ is clipped to 0 in (70). In fact, given enough time they may eventually become tight.

Note that whenever the bounds tighten by $\delta(n) > 0$, the control target must be shifted by $\Delta g(n)/\tilde{r}_a$, where $\Delta g(n) = -\delta(n)$. Furthermore, whenever n is the first frame of a new coding rate, the control target must be shifted by $\Delta g(n)/\tilde{r}_a$, where $\Delta g(n) = \hat{g}^{new}(n) - \hat{g}^{old}(n)$. Here, $\hat{g}^{old}(n)$ can be determined by running (69), (70), and (71) for one extra step, namely if n is the first frame of the new coding rate,

$$\hat{F}^e(n) = \max\{0, \hat{B}^e(n-1) - \hat{r}_c^{old}/f(n-1)\} \tag{72}$$
$$\hat{B}^e(n) = \hat{F}^e(n) + b(n) \tag{73}$$
$$\hat{g}^{old}(n) = B - B^e(n). \tag{74}$$

It is easy to see that if $\hat{g}^{new}(n) = B - b(n)$, then $\Delta g(n) = \hat{F}^e(n)$ as computed in (72).

We may also use for $\hat{g}^{new}(n)$ any better bound on $g^{new}(n)$. Better bounds are the subject of future study.

C. Virtual Streams

In MBR streaming, video and audio data are usually encoded separately, each generating multiple streams (hereafter called *substreams*). Although our optimal coding rate control method is derived based on a single stream model, it can

be easily extended to accommodate combinations of audio, video and/or other simultaneous substreams by introducing the concept of a *virtual stream*. A virtual stream is a combination of one or more media substreams having an aggregate leaky bucket characterization. Our optimal coding rate control method can then make switching decisions among a collection of virtual streams.

Happily, the leaky bucket (B, F^e, R) of a virtual stream can be easily derived as the sum of the leaky buckets of its component substreams. For example, if (B_a, F_a^e, R_a) and (B_v, F_v^e, R_v) are the leaky buckets for the component substreams (say audio and video substreams) of a virtual stream, then

$$B = B_a + B_v \quad (75)$$
$$F^e = F_a^e + F_v^e \quad (76)$$
$$R = R_a + R_v, \quad (77)$$

characterize a leaky bucket of the virtual stream. This is because, as is intuitively clear from the leaky bucket metaphor, if the separate leaky buckets contain their substreams without overflowing or underflowing, then the combined leaky bucket will contain the combination of substreams without overflowing or underflowing. (On the other hand, the combined bucket is in general not the smallest leaky bucket that is able to contain the combined substreams.) It is simple to show this mathematically, though we will not do so here. The important thing to note is that our optimal coding rate control method can work just as easily on combinations of audio and video substreams by making switching decisions among a collection of virtual streams, whose leaky buckets are easily derived from their component substreams.

Combining audio and video substreams can lead to a large number of choices of aggregate bit rates (and thus quality levels). In principle, each of the (say) N_a audio substreams can be matched with each of the N_v video substreams, producing all possible $N_a \times N_v$ combinations. However, most of these combinations are not desirable. In fact, typically there are only on the order of $N_a + N_v$ desirable combinations. For example, if audio quality is more important than video quality, then during network congestion it may be desirable to reduce video quality through N_v levels before reducing audio quality through an additional N_a levels. On the other hand it may instead be desirable to reduce the audio and video bit rates together. A principled way to decide which of the $N_a \times N_v$ combinations are desirable is the following. Assign a distortion $D_a(i)$ and a bit rate $R_a(i)$ to each audio substream $i = 0, 1, \ldots, N_a$ (which includes the empty substream $i = 0$) and a corresponding distortion $D_v(j)$ and bit rate $R_v(j)$ to each video substream $j = 0, 1, \ldots, N_v$. Define for each combined stream (i, j) an overall distortion and an overall bit rate,

$$D(i,j) = \alpha D_a(i) + D_v(j) \quad (78)$$
$$R(i,j) = R_a(i) + R_v(j), \quad (79)$$

allowing the audio distortion to be arbitrarily weighted by a parameter α relative to the video distortion. Select a 'desirable' subset of the audio/video substream combinations (i, j) such that for each (i, j) in the subset, $D(i, j) \leq D(i', j')$ for all (i', j') such that $R(i', j') \leq R(i, j)$. That is, desirable combinations have the property that they have the lowest total distortion among all combinations with the same or lower total bit rate. One such desirable subset consists of the combinations (i, j) whose rate-distortion pairs $[R(i, j), D(i, j)]$ lie on the lower convex hull of the set of rate-distortion pairs for all possible combinations. Pairs on this lower convex hull can be easily found by minimizing a Lagrangian for some positive Lagrange multiplier $\lambda > 0$, that is,

$$\begin{aligned}(i_\lambda, j_\lambda) &= \arg\min_{(i,j)}\{D(i,j) + \lambda R(i,j)\} \\ &= \arg\min_{(i,j)}\{\alpha D_a(i) + D_v(j) + \lambda[R_a(i) + R_v(j)]\} \\ &= (\arg\min_i\{D_a(i) + (\lambda/\alpha)R_a(i)\}, \\ &\quad \arg\min_j\{D_v(j) + \lambda R_v(j))\}. \end{aligned} \quad (80)$$

Thus, as λ is swept from 0 to ∞, a sequence of $N_a + 1$ audio substreams i_λ (including the null substream $i = 0$) can be chosen by minimizing the Lagrangian $D_a(i) + (\lambda/\alpha)R_a(i)$, and (independently) a sequence of $N_v + 1$ video substreams j_λ can be chosen by minimizing the Lagrangian $D_v(j) + \lambda R_v(j))$. These can be paired by matching their Lagrange multipliers λ. Note that it is a simple matter to re-pair them if the relative audio weight α changes, possibly under user control.

This approach can be easily extended to more substreams than just audio and video. For example, suppose there are M media elements in a streamed video game and $m = 1, 2, \ldots, M$ indexes the media elements, and suppose that for each media element m, there is a set of substreams $i_m = 0, 1, \ldots, N_m$ (including the null substream $i_m = 0$), one of which can be combined with substreams from other media elements in a compound virtual stream. Then following the above arguments it is easy to see that for each media element m, one can select for each $\lambda > 0$ a substream $i_{m,\lambda} = \arg\min_i\{D_m(i) + \lambda R_m(i)\}$, where $[R_m(i), D_m(i)]$ is the rate, distortion pair for the ith substream of media element m. These can then be aligned by λ to choose the components of the 'desirable' virtual streams, a process that is linear in M instead of exponential in M. Even further simplifications accrue when $N_m = 1$ for all m. In that case, as λ goes from 0 to ∞, for each m there is a simple threshold, namely $\lambda_m = [D_m(0) - D_m(1)]/R_m(1)$, such that when $\lambda \leq \lambda_m$ we have $i_{m,\lambda} = 1$ (i.e., the substream for media element m is included in the virtual stream) and when $\lambda > \lambda_m$ we have $i_{m,\lambda} = 0$ (i.e., the substream for media element m is not included in the virtual stream). Thus the set of desirable virtual streams can be obtained by sorting the media elements on λ_m and including them, in order, into the virtual streams.

D. Performance Evaluation

The performance of the controller is evaluated using an MBR file containing a 20-minute clip of *The Matrix* coded at five different combinations of audio and video bit rates, as listed in Table III, using a 5-second leaky bucket for each coding rate. The ns-2 simulation set up is similar to the set up in Figure 19. The connection between the media server and client is either TCP or TFRC. The bandwidth available to the

Audio (Kbps)	Video (Kbps)	Audio +Video (Kbps)
32	32	64
32	64	96
32	189	221
32	314	346
32	464	496

TABLE III

BIT RATES IN MBR FILE.

	without initial burst	with initial burst
0–5 s	500 Kbps	2 Mbps
5–25 s	500 Kbps	1 Mbps
25–70 s	400 Kbps	400 Kbps
70–130 s	286 Kbps	286 Kbps
130–190 s	200 Kbps	200 Kbps
190–220 s	286 Kbps	286 Kbps
220–550 s	400 Kbps	400 Kbps

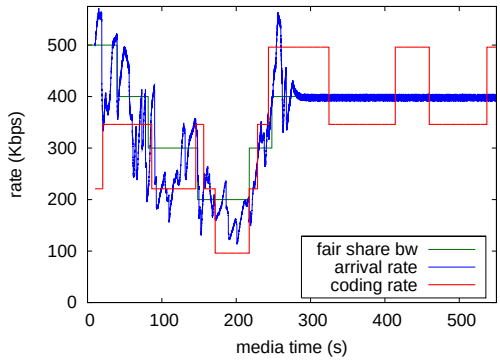
TABLE IV

BANDWIDTH CONDITIONS WITH AND WITHOUT INITIAL TRANSMISSION RATE BURST

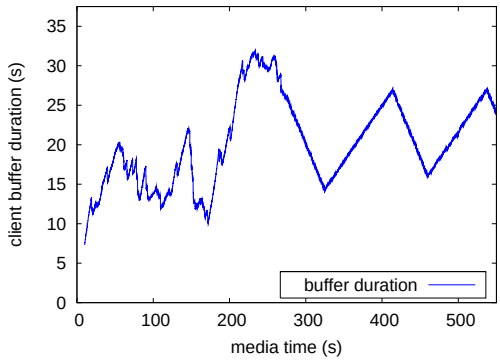
TCP or TFRC connection varies over time according to the schedule listed in Table IV, simulating congestion conditions that cause multiple rebuffering events in Windows Media 9. There are two sets of experiments: one with and one without an initial burst of available bandwidth.

When TCP is used to carry data from the server to the client, our coding rate controller satisfies users' expectations, with less than one second of startup delay, no rebuffering, and maximal quality and smoothness over the entire session. Indeed, Figure 37(a) shows that the coding rate (and hence the quality) is as high as possible given the average arrival rate, except during the first 15 seconds or so, in which the coding rate is lower than the arrival rate to build up the client buffer without incurring a large startup delay. Smoothness is also achieved, since the coding rate does not change spuriously, dropping only when the client buffer falls below its target and rising only when it can sustain the higher bit rate for at least 60 seconds in steady state. Correspondingly, Figure 37(b) shows that after the initial 15 seconds, the buffer duration hovers between 10 and 35 seconds, and does not underflow. Figure 38 shows similar results when there is an initial burst of available bandwidth. The corresponding initial burst in transmission rate however allows the initial coding rate to be fairly high while the client buffer builds.

When the TFRC protocol is used to carry data from the server to the client, similar results are obtained when there is no packet loss in the network, as shown in Figure 39(a). (Here there is no initial burst in available bandwidth.) When there is 5% packet loss in the network, TFRC reduces the transmission rate accordingly, as shown in Figure 39(c). The additional dynamics of transmission rate, however, makes it difficult to understand the effect of packet loss on the controller. In this case, the simulation is modified to induce loss only within the client application, and 5% packet loss (Figure 39(b)) is essentially the same as no data loss (Figure 39(a)). This indicates that the controller is robust to a significant number of frames



(a) rate vs. time



(b) buffer vs. time

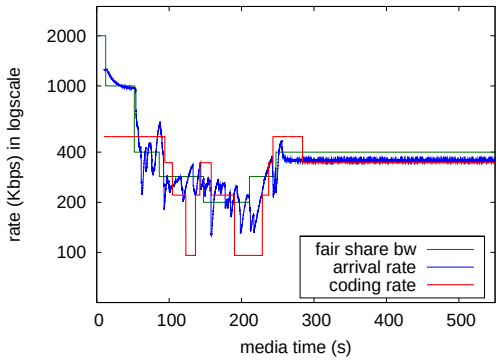
Fig. 37. TCP variable bandwidth experiment (without initial transmission rate burst).

being dropped. The reason for such robustness is that the client in any case groups together all frames within approximately 1-second intervals, creating large virtual frames at a virtual frame rate of $f = 1$ frame per second. A dropped frame simply causes the virtual frames to be slightly smaller, and the estimates for $t_b(n)$ to be slightly larger (more conservative). Thus, our controller should also work well even in wireless networks with significant packet loss due to interference and noise.

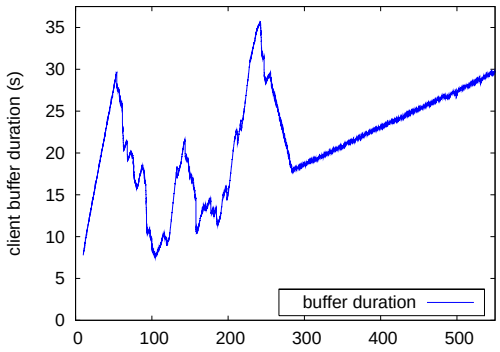
Finally, we study the effect of round trip time (RTT) on controller performance. Since our virtual frame rate is $f = 1$ frame per second, and the buffer size is on the order of 10 seconds or more, the controller is unaffected by large RTTs, as illustrated in Figure 40.

VIII. RELATED WORK

Hsu, Ortega and Reibman [17] 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 [23] 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

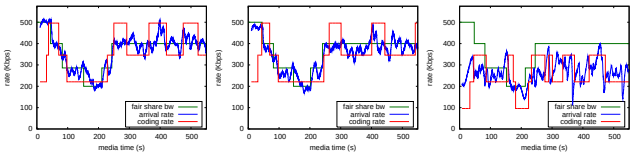


(a) rate vs. time



(b) buffer vs. time

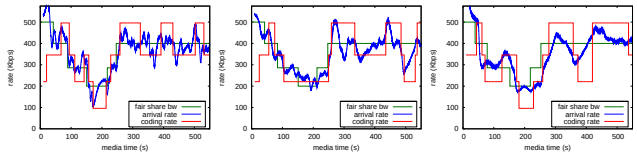
Fig. 38. TCP variable bandwidth experiment (with initial transmission rate burst).



(a) no data loss (b) application data loss (5% loss rate) (c) network data loss (5% loss rate)

Fig. 39. Performance Impact of Data Loss (over TFRC protocol)

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 [24]. The algorithms are good, however, for determining performance bounds in offline analysis.



(a) RTT = 80ms (b) RTT = 160ms (c) RTT = 320ms

Fig. 40. Performance Impact of RTT (over TCP protocol)

Nelakuditi, Harinath, Kusmierek and Zhang [11] design a bidirectional scan algorithm to optimize perceived video quality, measured by a set of smoothness metrics, assuming prior knowledge of the network bandwidth. Their work uses layered video and simplifies the analysis by assuming that each layer has a constant bit rate. The recent work of Kim and Ammar [12] further develops this direction, proposing a more sophisticated algorithm targeting optimal quality adaptation for MPEG-4 FGS VBR video. Both works also provide online heuristics, when the available bandwidth is not known in advance. These online heuristics appear to have reasonably good performance for limited scalability (one base layer and two enhancement layers in both works), although it is not clear how well they would work with a rich set of available bit rates (e.g., 50 streams in our case). In a similar way, it may be difficult to extend the dynamic bandwidth allocation algorithm proposed by Saparilla and Ross [13] beyond a limited number of bit rates.

To our knowledge, the most closely related contemporaneous work is that by de Cuetos and Ross [14], 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 work (this paper as well as [10]). 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 [25] 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 [26] 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.

IX. Summary

In this paper, we propose and verify an optimal online rate control algorithm for scalable and MBR streaming media. Our extensive analytical and experimental results show that three goals are achieved: fast startup (about 1 sec 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.

X. Acknowledgment

The authors would like to acknowledge the help and support of Sanjay Bhatt, Alex Colburn, Eduardo Oliviera, Dave Roth, Adam Trevor, Lihao Xu, and NSF grants CCR-0208975 and ANI-0322615.

References

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

[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] G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik. Video coding for streaming media delivery on the Internet. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):269–281, March 2001. special issue on Streaming Video.

[4] T. Wiegand and G. Sullivan. Joint video specification rec. h.264 & 14496-10 avc. Non-Final Draft of Final Draft International Standard (FDIS) JVT-G050, ITU-T & ISO/IEC, Pattaya, Thailand, March 2003.

[5] B. G. Haskell and A. Puri and. *Digital Video: An Introduction to MPEG-2*. Chapman & Hall, New York, 1997.

[6] B.-J. Kim, Z. Xiong, , and W. A. Pearlman. Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3-D SPIHT). *IEEE Trans. Circuits and Systems for Video Technology*, 10(8):1374–1387, December 2000.

[7] 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*, 11(3):301–317, March 2001.

[8] J. Li. Embedded audio coding (eac) with implicit psychoacoustic masking. In *Proc. Int'l Conf. Multimedia*, pages 592–601, Nice, France, December 2002. ACM.

[9] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. Technical Report MSR-TR-2001-35, Microsoft Research, Redmond, WA, February 2001.

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

[11] S. Nelakuditi, R. R. Harinath, E. Kusmierek, and Z.-L. Zhang, "Providing smoother quality layered video stream," in *Proc. Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, June 2000.

[12] T. Kim and M. H. Ammar, "Optimal quality adaptation for mpeg-4 fine-grained scalable video," in *Proc. Conf. Computer Communications (INFOCOM)*, San Francisco, CA, Apr. 2003.

[13] D. Saparilla and K. W. Ross, "Optimal streaming of layered video," in *Proc. Conf. Computer Communications (INFOCOM)*, Tel-Aviv, Israel, Mar. 2000.

[14] 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.

[15] 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, August 2000. ACM.

[16] 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*, 13(7), July 2003.

[17] 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*, 15(5):1016–1028, August 1997.

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

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

[20] S. Keshav. Packet-pair flow control. http://www.cs.cornell.edu/skeshav/-papers.html.

[21] V. J. Ribeiro, R. H. Riedi, J. Navratil, L. Cottrell, and R. G. Baraniuk. pathchirp: efficient available bandwidth estimation for network paths. In *Proc. Passive and Active Measurement Workshop (PAM)*, La Jolla, CA, April 2003.

[22] K. Fall and eds. K. Varadhan. The *ns* manual. Technical report, The VINT Project, December 2003. http://www.isi.edu/nsnam/ns/.

[23] 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*, 13(2):141–148, February 2003.

[24] A. Ortega, K. Ramchandran, and M. Vetterli. Optimal trellis-based buffered compression and fast approximation. *IEEE Trans. Image Processing*, 3:26–40, January 1994.

[25] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for Internet streaming video. *IEEE J. Selected Areas in Communications*, 18(12):2530–2543, December 2000.

[26] Q. Zhang, Y.-Q. Zhang, and W. Zhu. Resource allocation for multimedia streaming over the Internet. *IEEE Trans. Multimedia*, 3(3):339–355, September 2001.