OPTIMAL CONTROL OF MULTIPLE BIT RATES FOR STREAMING MEDIA

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

***Abstract*— Perhaps the major technical problem in streaming media on demand over the Internet is the need to adapt to changing network conditions. Today's commercial streaming media systems rely on Multi Bit Rate coding for adaptation. However, existing algorithms are empirical and often fail in a dynamic network. In this paper, we extend our optimal coding rate control framework and address the unique problems posed by the MBR stream switching constraints. A purely client-driven solution is provided, which achieves three goals: low startup delay (much less than 1 s even 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 robust against data loss and delay.**

I. INTRODUCTION

Perhaps the major technical problem in streaming media on demand over the Internet is the need to maintain a good user experience in the face of time varying network conditions. Users expect that regardless of the network conditions, the startup delay will be low, playback will be continuous, and quality will be as high as possible given the average network bandwidth.

Buffering at the client is the key to meeting these user expectations. Technically, buffering 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 instantaneous bit rate, thus improving overall quality.

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 asymptotically allows high robustness as well as high, nearly constant quality. Thus, client buffer management is a key element affecting the performance of streaming media systems.

The buffer *duration* (i.e., the number of seconds of content in the buffer) tends to increase or decrease depending on the ratio between the *arrival rate* r_a (the number of bits per second of real time that arrive at the client) and the *coding rate* r_c (the number of bits per second of encoded content, on average). If r_a/r_c is greater than the playback speed ν, then the buffer duration increases; otherwise it decreases. The arrival rate r_a is essentially determined by the network capacity. Hence, if the network capacity drops dramatically for a sustained period, reducing the coding rate r_c is the only appropriate way to maintain the buffer duration and prevent an underflow leading to a *rebuffering event*. Adjusting the coding rate in the face of time varying network conditions is the problem of *coding rate control*.

Today's commercial streaming media systems [1], [2] rely on multi bit rate (MBR) coding to perform coding rate control. 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 a set of coding rates $\{r_c\}$, typically using bit stream switching.

However, coding rate control algorithms employed in existing systems are empirical and often fail in dynamic network environments. The failures can be clearly observed as rebuffering events: playback stalls while the buffer "reloads" so that playback can be resumed. However, long startup delays, low perceived quality, and network stress are also symptoms of poor coding rate control. The difficulties faced by coding rate control algorithms include dealing with the stream switching constraints inherent in MBR files, such as widely spaced available coding rates and random, widely spaced available switching times. (The latter may be due to, for example, the need to wait for an *I* frame in a stream before switching to it.) Such difficulties complicate the coding rate control algorithms in the MBR case.

In this paper we extend our framework for optimal coding rate control [3], which is based on linear quadratic optimal control theory and was originally designed for scalable streaming media. Here we address the unique problems posed by the MBR stream switching constraints not present in scalable media. In addition, we provide a purely client-driven solution, which is more likely to be directly applicable to existing systems.

II. OPTIMAL CODING RATE CONTROL

The elements of our control model are illustrated in Figure 1. *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. The *playback deadline*, which indicates the time at which frames are

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

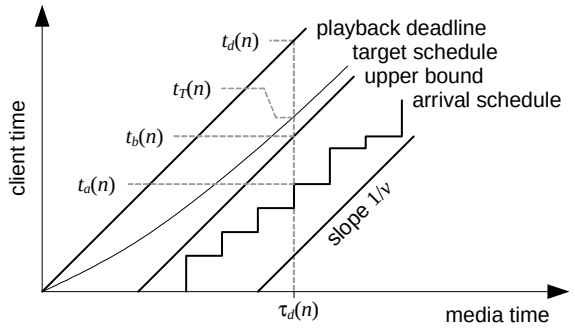


Fig. 1. Coding Rate Control Model

instantaneously decoded and rendered, increases linearly at a rate of $1/\nu$ seconds of client time per second of media time, where ν is the playback speed. (For instance, slow motion sets $\nu = 1/2$, then each second of media will be consumed in 2 seconds of client time.) The *arrival schedule*, which indicates the times at which encoded frames arrive at the client, increases in steps of size $b(n)/r_a$, where $b(n)$ is the size in bits of frame n. The arrival schedule is bounded between a *lower* and an *upper bound* representing a leaky bucket [4] containing the encoded stream. The upper bound increases linearly at slope r_c/r_a. The goal of our control system is to control the upper bound so that it tracks a *target schedule* sufficiently in advance of the playback deadline. It should be clear that the direction of the upper bound can be changed by changing the coding rate r_c, possibly to compensate for a change in the arrival rate r_a (affected by available network bandwidth). It turns out that the upper bound $t_b(n)$ at frame n evolves according to the linear dynamical system

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

where $r_c(n+1)$ is the coding rate of frame $n+1$, $\tilde{r}_a$ is a smoothed estimate of the arrival rate, and f the frame rate. Any deviation caused by using $\tilde{r}_a$ instead of the instantaneous arrival rate r_a in the above equation is captured by the noise term $w(n)$. Thus we can use linear feedback to make the upper bound $t_b(n)$ track a target schedule $t_T(n)$ by controlling the coding rate of a future frame. We also wish to minimize quality variations due to large or frequent changes in the coding rate. This is achieved by designing the feedback gain to minimize the quadratic cost function

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

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. Here, N is the control window size and σ is a Lagrange multiplier or weighting parameter balancing the two terms.

By defining the *error state* $e(n) = t_b(n) - t_T(n)$ and the *control input* $u(n) = \big(r_c(n+2) - r_c(n+1)\big)/\tilde{r}_a$, the linear dynamical system can be expressed in error space as:

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

where $\mathbf{e}(n) = [e(n)\ \ e(n-1)\ \ u(n-1)]^T$ is a state vector in the error space and $d(n) = w(n) - w(n-1)$. Correspondingly, the cost function becomes:

$$I = \sum_{n=0}^{N} \Big\{ \mathbf{e}(n)^T Q \mathbf{e}(n) + u(n-1)^T R u(n-1) \Big\}, \quad (2)$$

where $Q = C^T C$ (with $C = [1\ 0\ 0]$) and $R = \sigma$. Thus, the original coding rate control problem can be converted to a standard regulator problem in error space. After an optimal control feedback gain G^* is obtained by solving the corresponding *discrete algebraic Riccati equation* (DARE) [5] (letting $N \to \infty$), the ideal coding rate for frame $n+2$ can be computed as (refer to [6] for detailed derivation and analysis of stability and robustness)

$$r_c(n+2) = r_c(n+1) - G^* \mathbf{e}(n) \tilde{r}_a.$$

The term $t_b(n)$ within the error vector $\mathbf{e}(n)$ can be estimated as $t_a(n) + g(n)/\tilde{r}_a$, where $g(n)$ is the amount of space left in the leaky bucket after frame n is inserted.

III. Multiple Bit Rate Streaming

The optimal coding rate control framework was originally designed for scalable streaming media. To apply it to MBR streaming, there are several differences that need to be carefully addressed.

First, in MBR streaming there are only a limited number of coding rates (usually 5-7) 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 switch point (e.g., an *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. A purely client-driven solution is desirable.

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

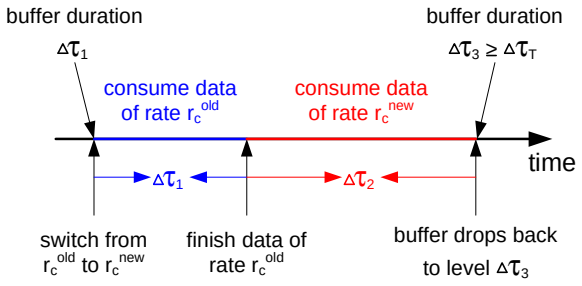


Fig. 2. Conservative rate up-switching.

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

Time (Sec)	AvailBW (Kbps)
0–25	500
25–70	400
70–130	300
130–190	200
190–220	300
220–550	400

TABLE I

LEFT: BIT RATES IN MBR FILE. RIGHT: AVAILABLE BANDWIDTH.

at least a minute. With this technique, rapid down-switching is permitted. In fact, we choose a small value of σ, changing the balance between responsiveness and smoothness of the coding rate in favor of a 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 idea 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. 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 2 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

$$r_c^{new} \leq \frac{r_a\Delta t}{\nu\Delta t - \Delta\tau_1 + \nu\Delta t_T}, \tag{3}$$

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 is more conservative, while a smaller Δt means that up-switching is more prompt. In our implementation, Δt is set to 60 seconds while the target Δt_T is typically about 10 seconds. This improves controller stability for MBR streaming.

B. Buffer Tube Upper Bound Estimation

We now address the problem of reducing server load. As discussed at the end of Section II, the buffer tube upper bound used by the controller is computed as $t_b(n) = t_a(n) + g(n)/\tilde{r}_a$, where $g(n)$ is the amount of space left in the leaky bucket after frame n is inserted (in bits). If the server can perform leaky bucket simulations for each of its streams in parallel, then when switching to a new stream at frame n_0, it can transmit $g(n_0)$ to the client. The client can then maintain the leaky bucket simulation for the new stream, computing $g(n)$ for each $n > n_0$.

If for complexity reasons the server is unable to perform the leaky bucket simulations for all of its streams in parallel, then the client must estimate $g(n_0)$ upon a switch to a new stream at frame n_0. We propose estimating $\hat{g}(n_0) = B - b(n_0)$, where B is the known leaky bucket size for the new stream and $b(n_0)$ is the size of the received frame n_0, in bits. This results in a conservatively large estimate of $g(n_0)$, and hence an upper bound on the upper bound $t_b(n_0)$. However, it can be shown that this upper bound tightens over subsequent frames $n > n_0$ as the leaky bucket simulation progresses [6]. We use this bound in all the experiments in this paper.

IV. PERFORMANCE AND DISCUSSIONS

We evaluate the performance of the controller 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 I (left), using a 5-second leaky bucket for each coding rate. Using the popular network simulator ns-2 [7], we set up a simple network with a 2 Mbps bottleneck and constant bit rate (CBR) cross traffic such that the bandwidth available to the TCP connection between the streaming media server and client varies over time according to the schedule listed in Table I, simulating congestion conditions that cause multiple rebuffering events in Windows Media 9.

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 3 (top) 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

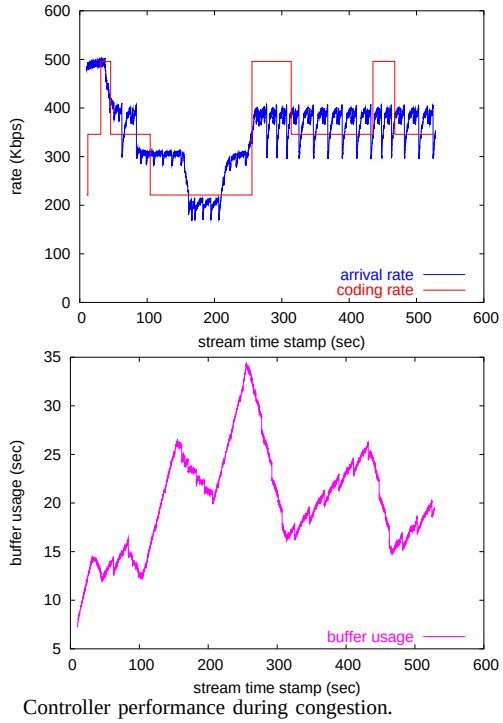
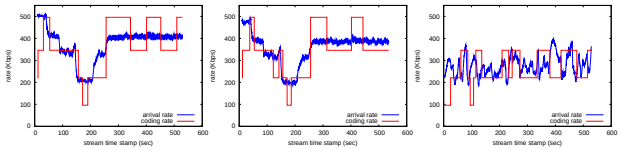


Fig. 3. Controller performance during congestion.

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 3 (bottom) shows that after the initial 15 seconds, the buffer duration hovers between 10 and 35 seconds, and does not underflow.

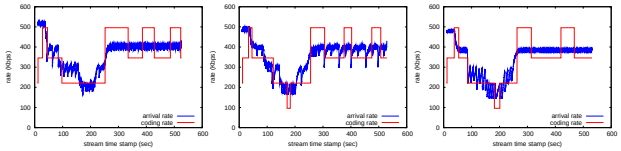
When the TFRC protocol is used to carry data from the server to the client, similar results are obtained when there is no packet loss in the network, as shown in Figure 4(a). When there is 5% packet loss in the network, TFRC reduces the transmission rate accordingly, as shown in Figure 4(c). This makes it difficult to understand the effect of packet loss on the controller. However, when packet loss is only induced within the client application, 5% packet loss (Figure 4(b)) is essentially the same as 0% packet loss (Figure 4(a)). This indicates that the controller is robust to a significant number of frames 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 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 5.



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

Fig. 4. Performance Impact of Data Loss (over TFRC protocol)



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

Fig. 5. Performance Impact of RTT (over TCP protocol)

REFERENCES

[1] 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.

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

[3] C. Huang, P. A. Chou, and A. Klemets. Optimal coding rate control for scalable streaming media. In *Proc. Int'l Packet Video Workshop*, Irvine, CA, December 2004. IEEE.

[4] 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.

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

[6] C. Huang, P. A. Chou, and A. Klemets. Optimal coding rate control for scalable and multi bit rate streaming media. Technical Report MSR-TR-04-XXX, Microsoft Research, Redmond, WA, December 2004. In preparation.

[7] K. Fall and K. Varadhan. The ns manual. The vint project, December 2003. http://www.isi.edu/nsnam/ns/.