A Generalized Hypothetical Reference Decoder for H.264/AVC

Jordi Ribas-Corbera, *Member, IEEE*, Philip A. Chou, *Senior Member, IEEE*, and Shankar L. Regunathan

***Abstract*—In video coding standards, a compliant bit stream must be decoded by a hypothetical decoder that is conceptually connected to the output of an encoder and consists of a decoder buffer, a decoder, and a display unit. This virtual decoder is known as the hypothetical reference decoder (HRD) in H.263 and the video buffering verifier in MPEG. The encoder must create a bit stream so that the hypothetical decoder buffer does not overflow or underflow. These previous decoder models assume that a given bit stream will be transmitted through a channel of a known bit rate and will be decoded (after a given buffering delay) by a device of some given buffer size. Therefore, these models are quite rigid and do not address the requirements of many of today's important video applications such as broadcasting video live or streaming pre-encoded video on demand over network paths with various peak bit rates to devices with various buffer sizes. In this paper, we present a new HRD for H.264/AVC that is more general and flexible than those defined in prior standards and provides significant additional benefits.**

***Index Terms*—Hypothetical reference decoder (HRD), video buffering verifier (VBV).**

I. Introduction

When compressing digital video, the encoder usually attempts to maintain the image quality nearly uniform throughout the video sequence, since drops or changes in video quality result in poor viewing experiences. To achieve this, the encoder must assign more bits to video frames or segments that are more difficult to compress (e.g., those that contain more textured regions or faster motion) and fewer bits to easier video segments, and as a result the encoding bit rate may vary significantly over time. Since compressed digital video is often transmitted through channels of (nearly) constant bit rate, the bit-rate variations need to be smoothed using buffering mechanisms at the encoder and decoder. The sizes of the physical buffers are finite, and hence the encoder must constrain the bit-rate variations to fit within the buffer limitations.

Video coding standards do not mandate specific encoder or decoder buffering mechanisms, but they require encoders to control bit-rate fluctuations so that a hypothetical reference decoder (HRD) of a given buffer size would decode the video bit stream without suffering from buffer overflow or underflow [1], [2]. This hypothetical decoder is based on an idealized decoder model that decoder manufacturers can use as a reference for their implementations, but its main goal is to impose basic buffering constraints on the bit-rate variations of compliant bit streams.

In previous HRDs [1], [2], the video bit stream is received at a known bit rate (usually the average rate in bits per second of the stream) and is stored into the decoder buffer until the buffer fullness reaches a desired level. This level is denoted the initial decoder buffer fullness and is directly proportional to the start-up (buffer) delay. When the initial buffer fullness is reached, the decoder instantaneously removes the bits for the first video frame of the sequence, decodes the bits, and displays the frame. The bits for the following frames are also removed, decoded, and displayed instantaneously at subsequent time intervals.

An HRD usually assumes that the decoding and display times preserve some pre-defined constraints, such as a fixed frame rate, and the system's end-to-end delay is constant (e.g., for broadcast applications). This constant-delay mode of operation is the focus of our contribution. The hypothetical decoder can also operate in a low-delay mode (e.g., for video conferencing). Low-delay operation can reduce average end-to-end delay by building a nominal delay into the system that is lower than the worst-case delay necessary to send "big pictures" (pictures that require an unusually large number of bits to encode with adequate fidelity). However, low-delay mode operation produces incorrect motion rendition caused by the variation in end-to-end delay in the neighborhood of big pictures, and is therefore undesirable for applications in which achieving good video quality is more important than reducing average end-to-end delay.

In the constant-delay mode, prior HRDs operate with a fixed peak bit rate, buffer size, and initial delay. However, in many of today's video applications (e.g., video streaming through the Internet) the peak transmission bit rate varies according to the network path (e.g., how the user connects to the network: by modem, ISDN, DSL, cable, etc.) and also fluctuates in time according to network conditions (e.g., congestion, the number of users connected, etc.) [3, Ch. 1–2]. In addition, the video bit streams are delivered to a variety of devices with different buffer capabilities (e.g., hand-sets, PDAs, PCs, set-top-boxes, DVD-like players, etc.) and are created for scenarios with different delay requirements (e.g., low-delay streaming, progressive download or pseudo-streaming, etc.) [4, Ch. 8]. As a result, these applications require a more flexible HRD that can decode a bit stream at different peak transmission bit rates, and with different buffer sizes and start-up delays.

We present a new HRD that operates according to N sets of transmission rate and buffer size parameters for a given bit stream. Each set characterizes what is known as a leaky bucket model [5], [6] and contains three values (R, B, F), where R is the peak transmission bit rate, B is the buffer size, and F is

Manuscript received December 12, 2001; revised May 9, 2003.

The authors are with Microsoft Corporation, Redmond, WA 98052 USA (e-mail: jordir@microsoft.com; pachou@microsoft.com; shanre@microsoft.com).

Digital Object Identifier 10.1109/TCSVT.2003.814965

1051-8215/03$17.00 © 2003 IEEE

TCL-1026
TTE Technology Inc. d/b/a TCL North America v. InterDigital, Inc. IPR2026-00417
AMAZON-1026
Amazon v. InterDigital, IPR2026-00195

the initial decoder buffer fullness. (F/R is the start-up or initial buffer delay.) An encoder can create a video bit stream that is contained by some desired N leaky buckets, or it can simply compute the N sets of parameters after the bit stream has been generated. Our new HRD intelligently interpolates among the leaky bucket parameters and can operate at any desired peak transmission bit rate, buffer size, or delay. To be more concrete, given a desired peak transmission bit rate R', our reference decoder will select the smallest buffer size and delay (according to the available leaky bucket data) that will be able to decode the bit stream without suffering from buffer underflow or overflow. Conversely, for a given buffer size B', the hypothetical decoder will select and operate at the minimum required peak transmission bit rate.

There are multiple benefits of this generalized HRD (GHRD). For example, a content provider can create a bit stream once, and a server can deliver it to multiple devices of different capabilities, using a variety of channels having different peak transmission bit rates. Or a server and a terminal can negotiate the best leaky bucket for the given networking conditions, e.g., the one that will produce the lowest start-up (buffer) delay, or the one that will require the lowest peak transmission bit rate for the given buffer size of the device. In Section IV, we quantify these benefits for the standard MPEG test sequences encoded with a recent test model of the H.264/AVC standard (i.e., JM version 3.2 [7]). We find that in realistic scenarios, the buffer size and the delay can be reduced by a factor of 14 to 45 at the same peak transmission bit rate, or the peak transmission bit rate can be reduced by almost a factor of four at the same physical buffer size. Alternatively, the buffer size can be increased by a factor of 45 without increasing the delay at the peak transmission bit rate, thereby potentially improving the signal-to-noise ratio (SNR) without increasing the average encoding rate.

In Section II, we describe the standard leaky bucket model and its associated parameters in more detail. Following that, in the context of the standard leaky bucket model we review the reference decoders used in H.263 (HRD) [1] and MPEG (VBV) [2], and discuss the limitations of these buffer models. Next, we present our GHRD, which uses N leaky buckets, and recommend minor syntax changes in the bit stream for implementing the GHRD. Finally, we present specific examples of the benefits over previous reference decoders using H.264/AVC bit streams.

This new HRD has been adopted as part of the H.264/AVC video coding specification [9]. Other aspects of the HRD in H.264/AVC are covered in Annex C of the standard [9], or in [10], [11]. The H.264/AVC video coding specification has been developed jointly by video codec experts from ITU and ISO, and the standardization effort is also known by other names, such as Joint Video Team (JVT), ITU-T H.26L, or ISO MPEG-4 part 10.

II. Leaky Bucket Model

We first define a leaky bucket model, since it is the basis of all the hypothetical reference decoders that we will discuss later.

A leaky bucket is a direct metaphor for the encoder's output buffer, i.e., the queue between the encoder and the communication channel. At frame time s_i, the encoder instantaneously

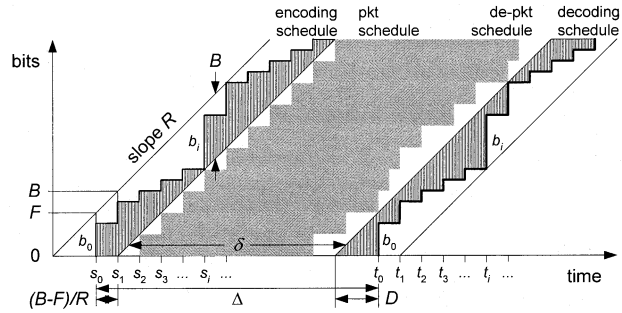

Fig. 1. Leaky bucket bounds in the CBR case. On the left: encoding schedule (staircase) and its leaky bucket bounds (straight lines). The encoder buffer fullness is the shaded area between the encoding schedule and the later/lower leaky bucket bound. On the right: decoding schedule (staircase) and its leaky bucket bounds. These are simply shifts of the encoding schedule and its bounds by a constant delay Δ. The decoder buffer fullness is the shaded area between the earlier/upper leaky bucket bound and the decoding schedule. Note that the decoder buffer fullness is the complement of the encoder buffer fullness in the CBR case. In the center: schedule for packet transmission and reception. The necessary additional encoder buffering and delay (for packetization) is the white area between the encoder's leaky bucket bound and the packet transmission schedule. The necessary additional decoder buffering and delay (for de-packetization and jitter) is the white area between the packet reception schedule and the decoder's leaky bucket bound.

encodes frame i into b_i bits and pours these bits into the leaky bucket.[1] In the constant bit rate (CBR) case, the leaky bucket drains its accumulated bits into the communication channel at a fixed bit rate R, and the encoder must add enough bits to the leaky bucket often enough so that the leaky bucket does not underflow (i.e., become empty) in any interval of time. On the other hand, the encoder must not add too many bits to the leaky bucket too frequently, or else the leaky bucket, which has capacity B, will overflow. Thus, the leaky bucket, which may begin at an arbitrary initial state F (with $0 \leq F \leq B$), constrains the encoding sequence (s_i, b_i), $i = 0, 1, 2, \ldots$ Graphically, the encoding sequence, or encoding *schedule*, can be represented by the cumulative number of bits encoded by time s, as illustrated in the left half of Fig. 1. Furthermore, the leaky bucket constraint can be represented by the two parallel lines bounding the encoding schedule. The later/lower line represents the schedule on which bits drain from the leaky bucket, and the earlier/upper line represents the capacity constraint of the leaky bucket, that is, an upward shift of the later/lower line by B bits.

Although a leaky bucket is a metaphor for the encoder buffer, it also characterizes the decoder buffer, i.e., the queue between the communication channel and the decoder. In the CBR case, after the encoded bits traverse the channel, they enter the decoder buffer at a fixed bit rate R. Then, at frame time $t_i = s_i + \Delta$, where Δ is a constant encoding-to-decoding delay, the decoder instantaneously extracts b_i bits from the decoder buffer and decompresses frame i, $i = 0, 1, 2, \ldots$. This decoding schedule, (t_i, b_i), $i = 0, 1, 2, \ldots$, is illustrated in the right half of Fig. 1. If, after the first bit enters the decoder buffer, the decoder delays at least $D = F/R$ seconds before decoding the first frame, then the decoding schedule is guaranteed not to un-

[1]Note that the frames in the bit stream are ordered in decoding order. Any re-ordering of pictures can be done for the purposes of displaying (e.g., a given P frame could be decoded before a given B frame, but the latter could be displayed earlier). The decoding time for a frame must not be later than its display time.

derflow the decoder buffer, due to the leaky bucket bounds inherited from the parallel encoding schedule. Furthermore, with delay $D = F/R$, if the capacity of the decoder buffer is at least B, then the decoding schedule is guaranteed not to overflow the decoder buffer, again due to the leaky bucket bounds inherited from the parallel encoding schedule. In fact, observe that the fullnesses of the encoder and decoder buffers are complements of each other in the CBR case [8]. Thus, the leaky bucket model determines both the minimum decoder buffer size and the minimum decoder buffer delay using three parameters, R, B, and F, by succinctly summarizing with upper and lower bounds the encoded sequence (t_i, b_i), $i = 0, 1, 2, \ldots$.

The leaky bucket model can also be used with variable bit rate (VBR) channels, such as packet networks. If the VBR channel has a long-term average bit rate that equals the long-term average bit rate of the encoded sequence, then it is often convenient to continue to use the above CBR leaky bucket bounds. At the decoder, the buffering B and the delay D due to the leaky bucket can be augmented by additional buffering and delay to accommodate both de-packetization and packet network delivery jitter. Likewise, at the encoder, the buffering and delay can be augmented by additional buffering and delay to accommodate packetization. The additional buffering and delay at both the encoder and decoder are illustrated in Fig. 1. The resulting total amount of buffering and delay are sufficient to guarantee continuous media playback without stalling due to decoder buffer underflow and without loss due to decoder buffer overflow. In essence, at the decoder, the leaky bucket provides a deadline by which packets must be available for decoding, or risk being late. Similarly, at the encoder, the leaky bucket provides a deadline by which the encoded bits will be available for packetization.

For VBR channels that have a *sustainable* peak bit rate R' *greater* than the long-term average bit rate R of the encoded sequence, then it is beneficial to characterize the encoded sequence $\{(t_i, b_i)\}$ using a leaky bucket with the higher leak rate R', yet allowing the leaky bucket to underflow (become empty) when the channel drains the bucket faster than the encoder can fill it. To illustrate, Fig. 2 shows the *same* encoding and decoding schedules as in Fig. 1, using a leaky bucket with the higher leak rate R'. As in Fig. 1, the later/lower bound on the encoding schedule is the schedule by which bits drain from the leaky bucket and are transmitted or packetized. In Fig. 2, however, flat spots in this transmission schedule indicate intervals in which the bucket is empty because there is nothing to transmit. The earlier/upper bound on the encoding schedule represents the capacity constraint of the leaky bucket: an upward shift of the transmission schedule by B' bits. Since the bucket drains at a rate R' greater than R, the bucket can have a capacity B' smaller than B, while still not overflowing, and/or can start with an initial state F' smaller than F. The transmitted bits enter the decoder buffer after a constant transmission delay δ. If, after the first bit enters the decoder buffer, the decoder delays at least $D' = F'/R'$ seconds before decoding the first frame, then the decoding schedule is guaranteed not to underflow the decoder buffer. Furthermore, with delay $D' = F'/R'$, if the capacity of the decoder buffer is at least B', then the decoding schedule is guaranteed not to overflow the decoder buffer. In this VBR case, the encoder buffer fullness and the decoder buffer fullness are

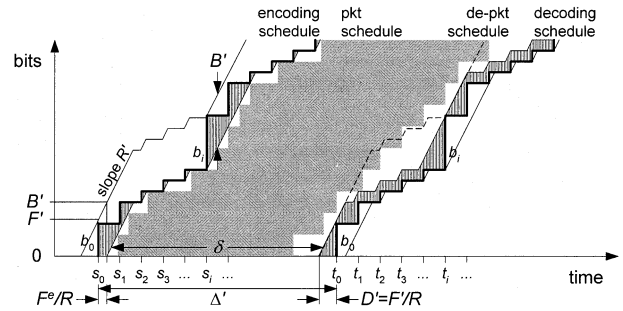

Fig. 2. Leaky bucket bounds in the VBR case. The encoding and decoding schedules are the same as in the CBR case in Fig. 1 (thus they have the same average bit rate), but the peak transmission bit rate R' is higher, resulting in a lower buffer size B', a lower initial delay $D' = F'/R'$, and a lower encoding-to-decoding delay Δ'. On the left: the encoding schedule is bounded below/right by the schedule by which bits leave the encoder bucket and are nominally transmitted. Flat spots indicate intervals in which the bucket is empty because there is nothing to transmit. The encoding schedule is bounded above/left by an upward shift of the transmission schedule by B' bits. This is the constraint imposed on the encoding schedule by the leaky bucket capacity. On the right: the decoding schedule is bounded above/left and below/right by rightward shifts of the transmission schedule. The earlier of these shifts is the schedule at which bits nominally arrive in the decoder buffer. Note that the encoder and decoder buffer fullnesses are no longer complementary. However, the decoder buffer fullness is bounded by the complement of the encoder buffer fullness (dotted).

no longer complementary. However, the complement of the encoder buffer fullness provides a tight (achievable) upper bound to the decoder buffer fullness. The advantage of using a leaky bucket with a peak rate greater than the average rate is that it allows a smaller decoder buffer size and delay for the same encoded sequence.

Clearly, a VBR channel is a generalization of a CBR channel, and in either case a single leaky bucket is specifiable by three parameters (R, B, F), where:

- R is the peak transmission bit rate (in bits per second) at which bits may leave the encoder buffer and enter the decoder buffer after a constant delay;
- B is the capacity (in bits) of the encoder or decoder buffer;
- F is the initial decoder buffer fullness (in bits) before the decoder can start removing bits from its buffer.[2] F and R determine the initial or start-up delay D, where $D = F/R$ seconds.

A leaky bucket with parameters (R, B, F) is said to *contain* an encoded bit stream if there is no overflow of the encoder buffer (i.e., the leaky bucket) or equivalently, if there is no underflow of the decoder buffer. Thus, a leaky bucket that contains an encoded bit stream provides constraints on the stream's encoding and decoding schedules so that the resources necessary to decode it are predictable.

A leaky bucket characterization of an encoded bit stream is especially important when the bit stream is encoded off line (for applications such as playback from a local disk, or streaming video on demand over a packet network from a remote server). When an entire bit stream is recorded and encoded before playback, it is essential that the decoder buffer and delay require-

[2]A leaky bucket can also be specified by parameters (R, B, F^e), where F^e is the initial encoder buffer fullness. In this paper, we have chosen to use the initial decoder buffer fullness $F = F^d$, unless otherwise indicated. They are related by $F^e + F^d = B$.

ments be stored along with the encoded bit stream, so that it will be possible for a decoder to know whether it will be able to decode a bit stream and what start-up delay to give it. The leaky bucket model provides a way to parameterize these requirements.

When the bit stream is encoded online, that is, when the encoder is connected directly to the transmission channel (for applications such as point-to-point video telephony), then in some circumstances, e.g., communication over a variable rate packet network with very low jitter, the encoder can control or at least know the instantaneous bit rate $R(t)$ at which bits arrive at the receiver. In this case, rather than use a leaky bucket model, it may be better to use a low-delay mode, in which the receiver may decode each frame of data as soon as it is fully received. In such a mode, the decoder buffer underflow is impossible (by definition) and decoder buffer overflow can be prevented by the encoder either by reducing $R(t)$ to match the number of bits per frame, or by increasing the number of bits per frame to match $R(t)$. Unfortunately, low-delay mode does not preserve presentation timing; constant end-to-end delay is sacrificed. Thus, the leaky bucket model is also important for on-line applications when precise presentation timing is important (e.g., for broadcasting high quality video live), or when the encoder cannot control or know the instantaneous rate $R(t)$ at which bits arrive at the receiver, or when $R(t)$ is anyway either a positive constant or zero (e.g., for transmission in a multiplex over an isochronous channel).

In this paper, we focus on generalizing the above single leaky bucket model to multiple leaky buckets for H.264/AVC. As mentioned earlier, other aspects of the HRD in H.264/AVC, such as low-delay mode, are not addressed here, but are explained in [9]–[11].

Before exploring multiple leaky buckets, let us be more specific about how a single leaky bucket with parameters (R, B, F) works in H.264/AVC. As indicated above, let $\{(s_i, b_i)\}$ be the encoder schedule and let $\{(t_i, b_i)\}$ be the corresponding decoder schedule. The leaky bucket starts with initial fullness F^e, which means that if frame 0 is inserted into the leaky bucket at time s_0, then transmission of the initial bit of frame 0 begins (after a delay of F^e/R) at time

$$s_0^{\text{initial}} = s_0 + \frac{F^e}{R}.$$

Transmission of the final bit of frame 0, indeed the final bit of any frame i for $i \geq 0$, ends at time

$$s_i^{\text{final}} = s_i^{\text{initial}} + \frac{b_i}{R}.$$

Hence, transmission of the initial bit of the subsequent frame begins at time

$$s_{i+1}^{\text{initial}} = \max\left(s_i^{\text{final}}, s_{i+1}\right)$$

that is, when frame $i+1$ is inserted into the leaky bucket or when frame i completes transmission, whichever is later. Now, assuming a constant transmission delay δ, the initial bit of frame 0 begins its arrival in the decoder buffer at time

$$t_0^{\text{initial}} = s_0^{\text{initial}} + d = s_0 + \frac{F^e}{R} + \delta.$$

Similarly, the final bit of frame 0, or indeed the final bit of any frame i for $i \geq 0$, ends its arrival in the decoder buffer at time

$$t_i^{\text{final}} = s_i^{\text{final}} + \delta = s_i^{\text{initial}} + \frac{bi}{R} + \delta = t_i^{\text{initial}} + \frac{b_i}{R}.$$

The initial bit of the subsequent frame will begin its arrival in the decoder buffer at time

$$\begin{aligned} t_{i+1}^{\text{initial}} &= s_{i+1}^{\text{initial}} + \delta = \max\left(s_i^{\text{final}} + \delta, s_{i+1} + \delta\right) \\ &= \max\left(t_i^{\text{final}}, t_{i+1} - \left(\frac{F^e}{R} + \frac{Fd}{R}\right)\right) \end{aligned}$$

which follows since $t_i = s_i + F^e/R + \delta + F^d/R$ for all i. The last three displayed equations form the basis of the HRD model in H.264/AVC. (In the specification, F^d/R is known as the `initial_cpb_removal delay` and F^e/R is known as the `initial_cpb_removal_delay_offset`.) The decoder buffer will not underflow if and only if for all i, frame i arrives in the decoder buffer before it is due for removal, that is,

$$t_i \geq t_i^{\text{final}} = \max\left(t_{i-1}^{\text{final}}, t_i - \left(\frac{Fd}{R} + \frac{Fe}{R}\right)\right) + \frac{bi}{R}.$$

This will be true if $F^e + F^d \geq b_i$ for all i, which will in turn be true if $F^e + F^d \geq B \geq b_i$ (which in turn will be true if the encoder buffer does not overflow). Hence, the decoder buffer capacity $B^d = F^e + F^d$ is usually set equal to the encoder buffer (leaky bucket) capacity $B^e = B$ in order to minimize both the decoder delay F^d/R and the decoder buffer size B^d. For this reason, we do not ordinarily distinguish between B^d and B^e and we set $B^d = B^e = B$.

Evolution of the encoder and decoder buffers can also be described in terms of buffer fullness. If the encoder buffer (the leaky bucket) starts with initial fullness F^e, then when frame 0 is inserted into the encoder buffer, the fullness of the encoder buffer rises to

$$B_0^e = F^e + b_0.$$

Thereafter, the fullness of the encoder buffer after inserting frame i, $i > 0$ is

$$B_i^e = \min(0, B_{i-1}^e - R(s_i - s_{i-1})) + b_i. \qquad (1.0)$$

The encoder buffer will not overflow if and only if $B_i^e \leq B$ for all $i \geq 0$. The complement of these equations describes an upper bound on the decoder buffer fullness after removing frame i for $i \geq 0$. If the decoder waits until the decoder buffer fullness reaches F^d before removing and decoding the frame 0, then the fullness of the decoder buffer falls to

$$B_0^d = F_d - b_0.$$

Thereafter, the upper bound on the fullness of the decoder buffer after removing frame i, $i \geq 0$, can be expressed as

$$B_i^d = \max(B, B_{i-1}^d + R(s_i - s_{i-1})) - b_i. \qquad (1.1)$$

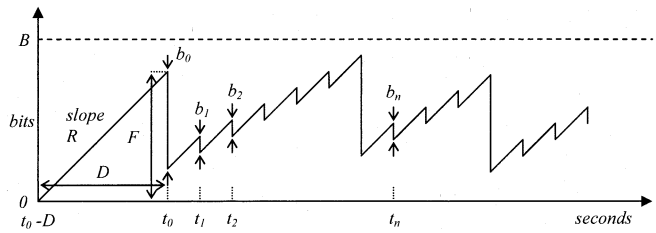

Fig. 3. Decoder buffer fullness when decoding a generic video bit stream that is contained in a leaky bucket having parameters (R, B, F). R is the peak incoming (or channel) bit rate in bits per second, B is the buffer size in bits, and F is the initial decoder buffer fullness in bits. $D = F/R$ is the initial or start-up (buffer) delay in seconds. The number of bits for the ith frame is b_i. The coded video frames are removed from the buffer (typically according to the video frame rate), as shown by the drops in buffer fullness.

It is simple to prove by induction that $B_i^e + B_i^d = B$ for all i; that is, (1.0) and (1.1) are complements. That the complement of the encoder buffer fullness is indeed an upper bound on the fullness of the decoder buffer can be seen from the fact that in any time interval of length B/R, at most B bits can enter the decoder buffer. Hence, as illustrated in Fig. 2, a leftward shift by B/R of the later/lower bound on the decoding schedule must lie below its upward shift by B. In other words, the actual decoder buffer fullness is bounded above by the complement of the encoder buffer fullness. Furthermore, the bound is tight, for if the encoder ever fills up the leaky bucket then the actual decoder buffer fullness achieves the bound. Thus, if a decoder buffer size $B_{\min}$ is minimally sufficient to contain the upper bound on the decoder buffer fullness, then it is also minimally sufficient to contain the actual decoder buffer fullness. Hence, in the remainder of the paper, we safely use (1.1) as a surrogate for the actual decoder buffer fullness, and refer to it simply as the "decoder buffer fullness." Since this is now complementary to the encoder buffer fullness, we are able to focus exclusively on the decoder buffer fullness, and we drop the superscript "d" from our notation. Fig. 3 illustrates the decoder buffer fullness as a function of time for a bit stream that is contained in a leaky bucket having parameters (R, B, F).

We now make the following observation. A given video stream can be contained in many leaky buckets. For example, if a video stream is contained in a leaky bucket with parameters (R, B, F), it will also be contained in a leaky bucket with a larger buffer (R, B', F), $B' > B$, or in a leaky bucket with a higher peak transmission bit rate (R', B, F), $R' > R$, or in a leaky bucket with larger start-up delay (R, B, F'), $F' > F$, $F \leq B$. Moreover, it can also be contained in a leaky bucket with a lower peak transmission bit rate (R', B, F), $R' < R$, if the video is time-limited. In the worst case, as R' approaches 0, the buffer size and initial buffer fullness will need to be as large as the bit stream itself. Put another way, a video bit stream can be transmitted at any peak transmission bit rate (regardless of the average bit rate of the clip) as long as the buffer size and delay are large enough.

Thus, for any value of the peak transmission bit rate R, we can find the minimum buffer size $B_{\min}$ and the minimum initial buffer fullness $F_{\min}$ that will contain the bit stream. This can be done by iterating (1.1), as illustrated by the Matlab code in Appendix B. Surprisingly, both minima can be achieved simultaneously, as we prove in Appendix A (Proposition 2), even though in general the buffer size required to contain the bit stream stays the same or increases as the initial buffer fullness decreases.

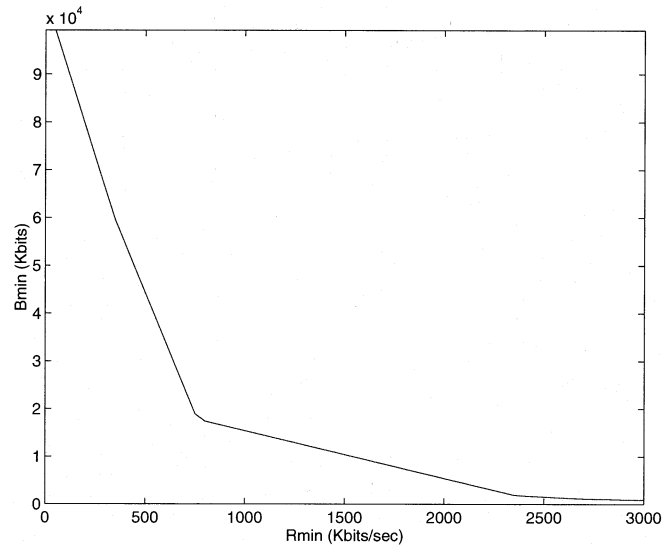

Fig. 4. Illustration of peak bit rate $R_{\min}$ and buffer size $B_{\min}$ values for a given bit stream. This curve indicates that in order to transmit the stream at a peak bit rate R, the decoder needs to buffer at least $B_{\min}(R)$ bits. Observe that higher peak rates require smaller buffer sizes. Alternatively, if the size of the decoder buffer is B, the minimum peak rate required for transmitting the bit stream is the associated $R_{\min}(B)$.

By computing $B_{\min}$ for each R, we can plot a curve of $R-B$ values such as the one in Fig. 4.[3]

A key observation is that the curve of $(R_{\min}, B_{\min})$ pairs for any bit stream (such as the one in Fig. 4) is piecewise linear and convex. Let us illustrate this interesting property using a simple, intuitive example. Consider the three plots of decoder buffer fullness in Fig. 5. These plots are generated when decoding a video bit stream that contains five frames, where $b_0, b_1, \ldots, b_4$ are the number of bits per frame $0, 1, \ldots, 4$, respectively. The peak transmission bit rate is R (top), R' (middle), and R'' (bottom), and $R < R' < R''$.

For clarity, the buffer (in each of the three cases) is initially filled to the maximum value B (which here represents the physical buffer size), i.e., $F = B$, although other initial values could have been selected as well. As expected, the minimum buffer size required to decode the bitstream decreases with larger peak transmission bit rate, i.e., $B_{\min}(R) > B_{\min}(R') > B_{\min}(R'')$. One can easily show that

$$B_{\min}(R) = \sum_{i=0}^{3} b_i - 3R\tau, \quad B_{\min}(R') = \sum_{i=0}^{3} b_i - 3R'\tau$$
$$B_{\min}(R'') = \sum_{i=2}^{3} b_i - R''\tau \tag{2}$$

where τ is the time interval between frames, which in this example is assumed to be constant (i.e., the decoding frame rate is $1/\tau$ frames/s). Observe that the formulas for $B_{\min}(R)$ and $B_{\min}(R')$ correspond to two different points on the same

[3]This curve is actually computed from a real video bit stream using (1.1) in the Matlab program of Appendix B.

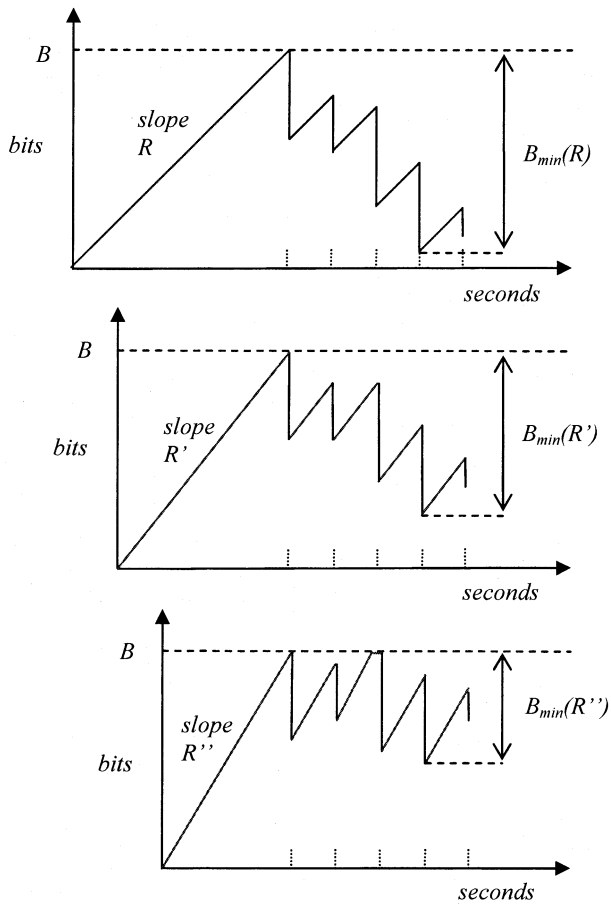

Fig. 5. The plots illustrate the decoder buffer fullness when decoding a video bit stream transmitted at bit rate R (top), R' (middle), and R'' (bottom), where $R < R' < R''$. Observe that the minimum buffer size required to decode the bitstreams is $B_{\min}(R) > B_{\min}(R') > B_{\min}(R'')$.

straight line of slope -3τ, and that the values of $B_{\min}$ for all bit rates between R and R' will lie on that line as well, because the respective decoder buffer plots would fall in between the top and middle plots in Fig. 5. For transmission bit rates slightly larger than R', the value of $B_{\min}$ will still lie on that same line, but at some point, the buffer will fill up within some frame interval (here between frames 1 and 2, as seen in the bottom plot of Fig. 5) and the formula for $B_{\min}$ will change to that of a straight line of slope $-\tau$. The value of $B_{\min}(R'')$ derived in (2) falls on this second straight line. Fig. 6 illustrates the curve of values $(R_{\min}, B_{\min})$ in this example.

Although this is a specific example, one can intuitively see that the decoder buffer plot for any video bit stream (of any number of frames) can be analyzed using the same process. As the peak transmission bit rate increases, the value of $B_{\min}$ will decrease following straight line formulas of the same form as those in (2), i.e., a summation of the number of bits for L frames and a slope — $(L-1)\tau$. The next line in the sequence will have fewer frames in the summation and hence a less negative slope, and as a result a piecewise linear curve that is monotonically decreasing such as that in Fig. 6 can be plotted. Since each new segment in the sequence has a less negative slope, such a piecewise linear curve is convex. For completeness, a rigorous proof of the piecewise linear and convex properties of the curve of $(R_{\min}, B_{\min})$ pairs (using induction) is provided in

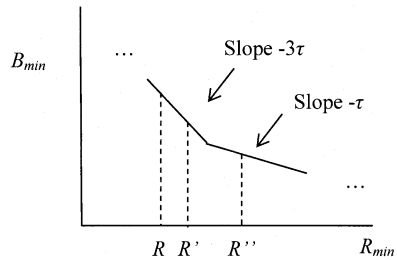

Fig. 6. Illustration of $(R_{\min}, B_{\min})$ pairs for the bit stream used in the decoder buffer plots of Fig. 5. As the channel rate increases, $B_{\min}$ decreases following a piecewise linear curve. Observe that $B_{\min}(R)$ and $B_{\min}(R')$ lie on the same straight line of slope -3τ, while $B_{\min}(R'')$ lies on the line of slope $-\tau$, as indicated in (2).

Appendix A. Not surprisingly, such linear and convex properties apply to $(R_{\min}, F_{\min})$ as well, which is also shown in Appendix A.

Because of the convexity, if N points of the curve are provided, the decoder can linearly interpolate the values to arrive at some points $(R_{\text{interp}}, B_{\text{interp}}, F_{\text{interp}})$ that are slightly but safely larger than $(R_{\min}, B_{\min}, F_{\min})$. In this way, as we quantify in Section V, one is able to safely reduce the buffer size, and the delay, by well over an order of magnitude, relative to a single leaky bucket containing the bit stream at its average rate. Alternatively, for the same delay, one is able to reduce the peak transmission rate by almost a factor of four, or possibly even improve the SNR.

III. Previous Work

We next explain the hypothetical reference decoders in MPEG and H.263 in the context of leaky bucket models.

A. MPEG's Video Buffering Verifier (VBV)

The VBV [2] can operate in two modes: CBR and VBR. MPEG-1 and MPEG-4 only support the CBR mode, while MPEG-2 supports both modes.

The VBV operates in CBR mode when the bit stream is contained in a leaky bucket having parameters (R, B, F) and the following conditions hold:

- $R = R_{\max} =$ the average bit rate of the stream;
- the value of B is stored in the syntax parameter vbv_buffer_size using a special size unit (i.e., 16×1024 bit units);
- the value of F/R is stored in the syntax element vbv_delay associated to the first video frame in the sequence using a special time unit (i.e., number of periods of a 90 KHz clock);
- the decoder buffer fullness follows the following equations:

$$\begin{aligned} B_0 &= F \\ B_{i+1} &= B_i - b_i + \frac{R_{\max}}{M}, \quad i = 0, 1, 2, \ldots. \end{aligned} \tag{3}$$

The encoder must ensure that $B_i - b_i$ is always greater than or equal to zero while B_i is always less than or equal to B. In other words, the encoder must ensure that the decoder buffer does not underflow or overflow.

The VBV operates in VBR mode when the bit stream is contained in a leaky bucket having parameters (R, B, F) and the following conditions hold:

- $R = R_{\max}$ = the peak or maximum rate. $R_{\max}$ is higher than the average rate of the bit stream;
- $F = B$, i.e., the buffer fills up initially;
- the value of B is represented in the syntax parameter vbv_buffer_size, as in the CBR case;
- the value of F is not stored and vbv_delay is set to FFFF (in hex);
- the decoder buffer fullness follows the following equations:

$$\begin{aligned} B_0 &= B \\ B_{i+1} &= \min\left(B, B_i - b_i + \frac{R_{\max}}{M}\right), \quad i = 0, 1, 2, \ldots. \end{aligned} \tag{4}$$

The encoder must ensure that $B_i - b_i$ is always greater than or equal to zero. That is, the encoder must ensure that the decoder buffer does not underflow. However, in this VBR case, the encoder does not need to ensure that the decoder buffer does not overflow. If the decoder buffer becomes full, then it is assumed that the encoder buffer is empty and hence no further bits are transmitted from the encoder buffer to the decoder buffer.

The VBR mode is useful for devices that can read data up to the peak rate $R_{\max}$. For example, a DVD includes VBR clips where $R_{\max}$ is about 10 Mbps, which corresponds to the maximum reading speed of the disk drive, even though usually the average rate of the DVD video stream is only about 4 to 6 Mbps.

Fig. 7 illustrates plots of decoder buffer fullness for some bit streams operating in CBR and VBR mode, respectively.

There are some aspects of VBV that we do not address here because they are either not relevant with respect to this contribution, or are simply special cases of the leaky bucket model. For example, the VBV includes a low-delay mode that tolerates frame skipping.

B. H.263's HRD

The HRD model for H.263 [1] (which is equivalent to that in H.261) operates in low-delay mode. It resembles the CBR mode of MPEG's VBV, except for the following.

- The decoder inspects the buffer fullness at some time intervals and decodes a frame as soon as all the bits for the frame are available. This approach results in a couple of benefits: 1) the start-up delay is minimized because F is usually just slightly larger than the number of bits for the first frame and 2) if frame skipping is common, the decoder simply waits until the next available frame. As mentioned above, the latter is enabled in the low-delay mode of MPEG's VBV as well.
- The check for decoder buffer overflow is done after the bits for a frame are removed from the buffer (i.e., $B_{i+1} = B_i - b_i + R_{\max}/M$ must be less than or equal to B, for all i). This relaxes the constraint for sending large I frames once in a while, but there is a maximum value for the largest frame.

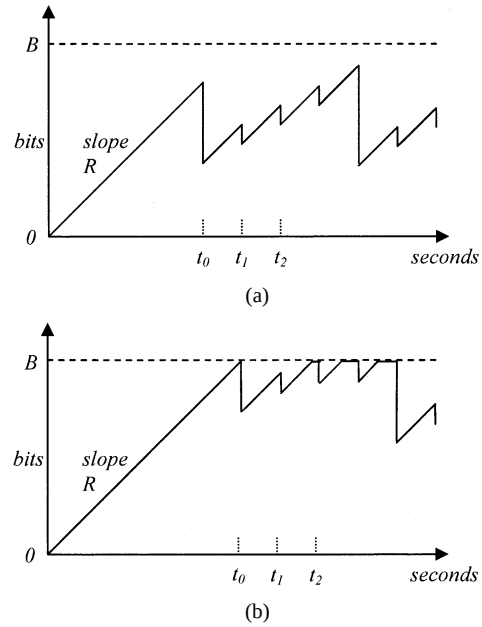

Fig. 7. Examples of typical plots of decoder buffer fullness for (a) CBR and (b) VBR for MPEG-2's VBV model.

- The transmission bit rate varies along time, and it is assumed that both encoder and decoder know the bit rate at each time instant.

The HRD in H.263 is suitable for low-delay communications (e.g., video conferencing). Observe that in such HRD, the decoding and display times vary significantly across frames, and hence this approach is not appropriate for constant-delay time-preserving applications.

C. Limitations of Previous Models

Observe that all previous hypothetical reference decoders operate at only one point (R, B) of the curve in Fig. 4. As a result, these decoders have the following drawbacks.

- If the bit rate available in the channel R' is lower than R (e.g., this is common for internet streaming and progressive download, or when an MPEG VBR clip needs to be transmitted at a rate lower than the peak), strictly speaking, the hypothetical decoder would not be able to decode the bit stream.
  - In practice, a decoder will not know the size of the buffer required for this lower rate, and generally it will run into buffer problems while decoding the bit stream. The best that a smart decoder could do would be to find a tight upper-bound for the buffer size. One can easily show that an almost tight[4] upper bound is $B' = B + (R - R')T$, where T is the time length or duration in seconds of the video sequence (to be more specific, T is the time difference between the decoding times of the last and first frames in the

[4]A tight upper bound, which depends on the initial decoder buffer fullness F of the original buffer, is $B' = (B - F)(R'/R) + F + (R - R')T$. This is because the initial buffer fullness F' of the new decoder buffer must be at least $F + (R - R')T$ to guarantee that the new decoder buffer will not underflow by time T when it receives bits at rate only $R' < R$, and because the headroom $B' - F'$ must be at least $(B - F)(R'/R)$ to guarantee that the new decoder buffer will not overflow by time $(B - F)/R$ when it receives bits at rate R'.

coded bit stream). Clearly, this bound increases the buffer size and delay requirements very rapidly and may not be very useful in practice (especially if T is large and R' is significantly lower than R).

- If the available bandwidth R' is larger than R (e.g., this is also common for internet streaming, as well as for local playback), the previous hypothetical decoders could operate in the VBR mode and decode the bit stream. However, as no additional information on the Rate-Buffer curve is available, the buffer size required to decode the bit stream cannot be reduced (as we will see later in the examples).
- If the physical buffer size in a decoder device is smaller than B, the device will not be able to decode that bit stream.
- If the buffer size is larger than B, the device will be able to decode the bit stream but the start-up delay will be the same.
- More generally, if a bit stream is generated according to a leaky bucket (R, B, F) and no other leaky bucket parameters are known, one will not usually be able to be distribute such bit stream through different networks having peak transmission bit rates smaller than R, and to a variety of devices with buffer sizes smaller than B. Also, the start-up delay will not be minimized.

IV. A Generalized GHRD

We present a GHRD that can operate given the information of N leaky bucket models

$$(R_1, B_1, F_1), (R_2, B_2, F_2), \ldots, (R_N, B_N, F_N) \tag{5}$$

each of which contains the bit stream. Without loss of generality, let us assume that these leaky buckets are ordered from smallest to largest bit rate, i.e., $R_i < R_{i+1}$. Let us also assume that the encoder computes these leaky bucket models correctly, and hence $B_i > B_{i+1}$.

The desired value of N can be selected by the encoder. (If $N = 1$, the GHRD is essentially equivalent to the VBR mode of MPEG's VBV). The encoder can choose to:

1) pre-select the leaky bucket values and encode the bit stream with a rate control that makes sure that all of the leaky bucket constraints are met;
2) encode the bit stream and then use (1.1) to compute a set of leaky buckets containing the bit stream at N different values of R;
3) do both.

The first approach can be applied to live or on-demand transmission, while 2) and 3) only apply to on-demand.

The number of leaky buckets N and the leaky bucket parameters (5) are inserted into the bit stream. In this way, the decoder can determine which leaky bucket it wishes to use, knowing the peak bit rate available to it and/or its physical buffer size. The leaky bucket models in (5), as well as all the linearly interpolated or extrapolated models, are available for use. Fig. 8 illustrates a set of N leaky bucket models and their interpolated or extrapolated (R, B) values.

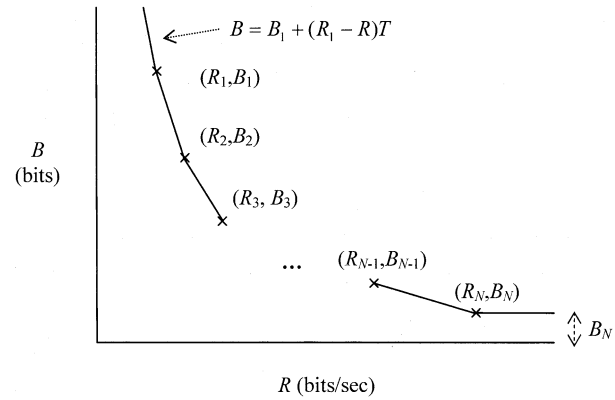

Fig. 8. Example of (R, B) values available for the GHRD, all of which are guaranteed to contain the bit stream. T is the time length or duration of the encoded video sequence.

The interpolated buffer size B between points k and $k+1$ follow the straight line

$$B = \frac{R_{k+1} - R}{R_{k+1} - R_k} B_k + \frac{R - R_k}{R_{k+1} - R_k} B_{k+1}, \quad R_k < R < R_{k+1}. \tag{6}$$

Likewise, the initial decoder buffer fullness F can be linearly interpolated

$$F = \frac{R_{k+1} - R}{R_{k+1} - R_k} F_k + \frac{R - R_k}{R_{k+1} - R_k} F_{k+1}, \quad R_k < R < R_{k+1}. \tag{7}$$

The resulting leaky bucket with parameters (R, B, F) is guaranteed to contain the bit stream, because, as we prove in Proposition 1 of Appendix A, the minimum buffer size $B_{\min}$ is convex in both R and F, that is, the minimum buffer size $B_{\min}$ corresponding to any convex combination $(R, F) = a(R_k, F_k) + (1-a)(R_{k+1}, F_{k+1})$, $0 < a < 1$, is less than or equal to $B = aB_k + (1-a)B_{k+1}$.

As discussed earlier, observe that if R is larger than R_N, the leaky bucket (R, B_N, F_N) will also contain the bit stream, and hence B_N and F_N are the buffer size and initial decoder buffer fullness recommended when $R \geq R_N$. If R is smaller than R_1, then the upper bound $B = B_1 + (R_1 - R)T$ can be used (and one can set $F = B$), where T is the time length of the video sequence in seconds. These (R, B) values outside the range of the N points are also shown in Fig. 8.

The number of bits required for the leaky bucket parameters is minimal. If R_i, B_i, and F_i were encoded in the same units as $R\max$, vbv_buffer_size, and vbv_buffer_delay (in the MPEG-2 bit stream syntax), then the number of bits required for these elements would be 30, 18, and 16, respectively, for a total of 64 bits per leaky bucket. The number of leaky buckets N could be specified with 8 bits. This information would be succinct enough that it could be re-specified at any point in the bitstream, such as at all random access points.

In the H.264/AVC specification, the number of leaky buckets is specified with a variable-length code in the range 1 to 32. Each bit rate in bits per second is specified with a variable-length mantissa in the range 1 to $2^{32} - 1$ and a fixed-length base-2 exponent in the range 6 to 21. Each buffer size in bits is specified

with a variable-length mantissa in the range 1 to $2^{32}-1$ and a fixed-length base-2 exponent in the range 4 to 19. All of these are in the HRD Parameters section of the video usability information (VUI). The initial decoder buffer delay in 90-kHz ticks is specified in the Buffering Period section of the supplemental enhancement information with a fixed-length code. The number of bits in this code is specified in the HRD parameters section of the VUI by a fixed-length code in the range 1 to 32.

V. Experimental Results: Evaluation of the GHRD

To evaluate the benefits of the GHRD, we encoded a 130-s video clip (which contained a sequence of 13 MPEG clips combined, i.e., "Stefan", "Akiyo", "Mother and Daughter", "Fun Fair", "Foreman", "Bream", "News", "Sean", "Children", "Mobile & Calendar", "Weather", "Container", and "Hall") with the Test Model JM version 3.2 [7] with the default parameters (i.e., five reference frames, 1/8th motion accuracy, arithmetic coding ON, hadamard transform ON, error robustness OFF, B frames OFF, and RD optimization ON, and a motion search range of 16 pixels). We set $F = B$ and used the formula (1.1) to provide the $(R_{\min}, B_{\min})$ plot in Fig. 9. To be more concrete, we ran the simple Matlab program in Appendix B which makes use of (1.1).

The bit stream in Fig. 9 was produced with $\mathrm{QP} = 26$ and yielded an average bit rate of 600 kbits/s. As shown in the figure, at a constant transmission bit rate of 600 kbits/s, the decoder needs a buffer size of about 16 500 kbits. With an initial decoder buffer fullness equal to 16 500 kbits, the start-up delay is about 27 s. Thus, this VBR encoding (produced with no rate control) shifts bits by up to 27 s in order to achieve constant quality over its encoded length.

The figure also shows that at a peak transmission bit rate of 2400 kbits/s (e.g., the video bit-rate portion of a $2\times$ CD), the decoder needs a buffer size of only 370 kbits, sufficiently small for a consumer hardware device. With an initial buffer fullness equal to 370 kbits, the start-up delay is only about 0.15 s.

Thus, for this encoding, two leaky bucket models might typically be useful.

1) ($R = 600\ \mathrm{kbits/s}$, $B = 16500\ \mathrm{kbits}$, $F = 16500\ \mathrm{kbits}$). This leaky bucket permits transmission of the video over a CBR channel, with a delay of about 27 s. While this delay may be too large for many scenarios, it is probably acceptable for streaming or progressive download of movies over the Internet, for example.
2) ($R = 2400\ \mathrm{kbits/s}$, $B = 370\ \mathrm{kbits}$, $F = 370\ \mathrm{kbits}$). This leaky bucket permits transmission of the video over a shared network with peak rate 2400 kbits/s, or permits local playback from a $2\times$ CD, with a delay of about 0.15 s. This subsecond delay is acceptable for random access playback with VCR-like functionality.

If only the first leaky bucket is specified in the bit stream, but not the second, then even when playing back over a channel with peak bit-rate 2400 kbits/s, the decoder would use a buffer of size 16 500 kbits and thus the delay would be $F/R = 16500\ \mathrm{kbits}/2400\ \mathrm{kbits/s} = 6.9\ \mathrm{s}$. This is too large for random access playback with VCR-like functionality. However, if the second leaky bucket is specified as well, then

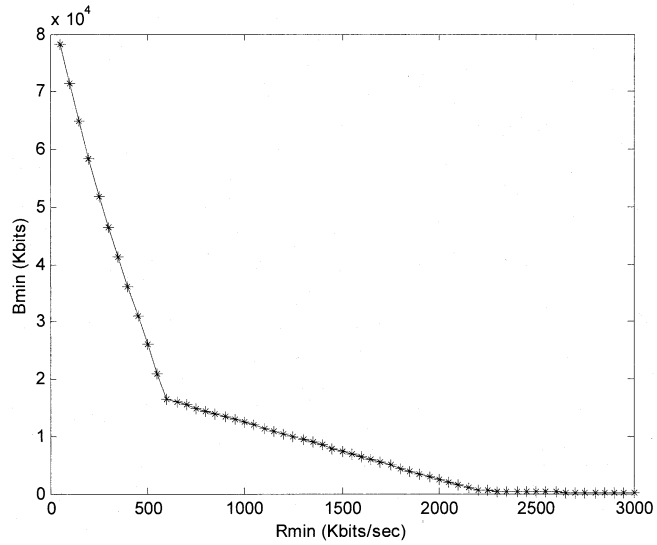

Fig. 9. Plot of leaky bucket parameters (R, B) for an H.264/AVC compressed video clip with $\mathrm{QP} = 26$. The points labeled with "$*$" correspond to the minimum buffer size $B_{\min}$ needed to contain a bit stream with the associated rate $R_{\min}$. These points were computed from the bit stream using the Matlab program in Appendix B, which uses the formulae in (1.1) and scans bit rates from 50 kbits/s to 3 Mbps in increments of 50 kbits/s. The other points between the "$*$" are linearly interpolated. Observe that, in practice, only a small subset of points (e.g., 4 or 5) may characterize the curve fairly well.

at peak bit-rate 2400 kbits/s the buffer size drops to 370 kbits and the delay drops to 0.15 s, as we have seen.

On the other hand, if only the second leaky bucket is specified, but not the first, then at a constant transmission rate of 600 kbits/s, even a smart decoder would be forced to use a buffer that is far larger than necessary, to ensure that the buffer will not overflow: $B' = B + (R - R')T = 370\ \mathrm{kbits} + (2400\ \mathrm{kbits/s} - 600\ \mathrm{kbits/s}) \times 130\ \mathrm{s} = 234370\ \mathrm{kbits}$. This corresponds to an initial delay of 391 s, or about 6.5 min, over three times the length of the original clip, which is far from acceptable. However, if the first leaky bucket is specified as well, then at rate 600 kbits/s the buffer size drops to 16 500 kbits and the delay drops to 27 s, as we have seen.

Moreover, if both leaky buckets are specified, then the decoder can linearly interpolate between them [using (6) and (7)], for any bit rate R between 600 and 2400 kbits/s, thereby achieving near-minimal buffer size and delay at that rate. Extrapolation is also more efficient both below 600 kbits/s and above 2400 kbits/s, compared to extrapolation with only a single leaky bucket anywhere between 600 and 2400 kbits/s.

As the above example shows, even just two leaky buckets can provide well over an order of magnitude reduction in buffer size (e.g., a factor of 234 370 kbits/16 500 kbits = 14 in one case and 16 500 kbits/370 kbits = 45 in another), and a corresponding reduction in delay (e.g., 391 s/27 s = 14 s in one case and 6.9 s/0.15 s = 45 s in another) at a given peak transmission bit rate.

Conversely, it is also possible to reduce the peak transmission bit rate for a given decoder buffer size. Indeed, as is clear from Fig. 9, if the $R-B$ curve can be obtained by interpolating and/or extrapolating multiple leaky buckets, then it is possible for a decoder with a fixed physical buffer to choose the minimum peak transmission bit rate needed to safely decode the bit stream

without decoder buffer overflow. For example, we know from the figure that if the decoder has a fixed buffer of size 16 500 kbits, then the peak transmission bit rate for the encoding can be as low as 600 kbits/s. However, if only the second leaky bucket is specified, but not the first, then the decoder can reduce the bit rate to no less than $R' = R - (B' - B)/T = 2400\ \mathrm{kbits/s} - (16500\ \mathrm{kbits} - 370\ \mathrm{kbits})/130\ \mathrm{s} = 2276\ \mathrm{kbits/s}$. In this case, compared to using a single leaky bucket, using just two leaky buckets reduces the peak transmission rate by almost a factor of four, for the same decoder buffer size.

It is quite likely that having multiple leaky buckets can also improve the quality of the reconstructed video, *at the same average encoding rate*, in the following sense. Suppose both leaky buckets are available for the encoding described above. Then, as we have seen, it is possible to play back the encoding with a delay of 27 s if the peak transmission rate is 600 kbits/s, and with a delay of 0.15 s if the peak transmission rate is 2400 kbits/s. However, if the second leaky bucket is unavailable, then the delay increases from 0.15 to 6.9 s at 2400 kbits/s. To reduce the delay back to 0.15 s without the benefit of the second leaky bucket, it would suffice to re-encode the clip *with rate control* by reducing the buffer size (of the first leaky bucket) from 16 500 kbits to $(0.15\ \mathrm{s}) \times (2400\ \mathrm{kbits/s}) = 370\ \mathrm{kbits}$, a factor of 45. This would ensure that the delay is only 0.15 s if the peak transmission rate is 2400 kbits/s. However, the quality of this rate-controlled stream would vary over time, and it is quite likely that the average quality (SNR) would be lower than that of the original constant-quality stream at the same average bit rate. Unfortunately we cannot yet evaluate this decrease in SNR with objective tests, because as of this writing there is no rate control in the test model of H.264/AVC. Thus we can only conjecture that specifying a second leaky bucket can increase the SNR with no change in the average bit rate (except for the additional bits per clip to specify the second leaky bucket). This increase in SNR would be visible on playback for every peak transmission rate.

The benefits of specifying multiple leaky buckets in a generalized hypothetical reference decoder appear, of course, only in heterogeneous situations, where a single encoding is transmitted over channels with different peak bit rates, or to devices with different physical buffer sizes. However, this is increasingly the case. Content that is encoded offline and stored on a disk is often played back locally as well as streamed over networks with different peak rates. Even for local playback, different drives speeds (e.g., $1\times$ CD through $8\times$ DVD and beyond) affect the peak transfer rate. And of course the peak transmission bit rates through network connections also vary dramatically according to the speed of the limiting link, which is typically near the end user (e.g., 100 or 10 baseT Ethernet, T1, DSL, ISDN, modems, etc.). Buffer capacities of playback devices also vary significantly, from desktop computers with gigabytes of buffer space to small consumer electronic devices with buffer space that is smaller by several orders of magnitude. Typically, content providers spend a significant amount of effort creating a single bit stream (e.g., top studios may spend over 80 h to create a DVD), and they wish to reach the largest audience, with the best user experience. The multiple leaky buckets in the proposed generalized hypothetical reference decoder make it possible for the same bit stream to be transmitted over a variety of channels with the minimum startup delay, minimum decoder buffer requirements, and maximum possible quality. This applies not only to video that is encoded off-line, but also to live video that is broadcast simultaneously through different channels to different devices. In short, the proposed GHRD adds significant flexibility to the transmission of existing bit streams.

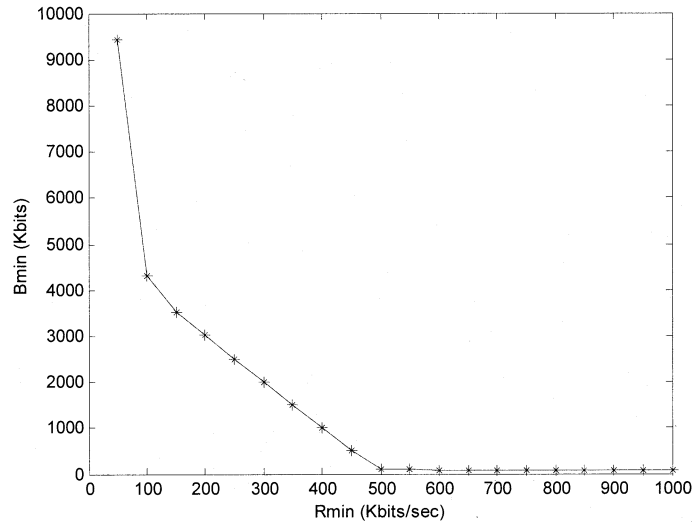

Fig. 10. Same plot as Fig. 9 but when $\mathrm{QP} = 38$. Once again, observe that only a subset of these points (e.g., 4 or 5) already characterize this curve well. In this case, the average bit rate of the clip is 150 kbits/s and the related buffer size and delay are 3500 kbits and 23 s, respectively. If the clip were streamed in a LAN or DSL channel with higher bit rate, say 500 kbits/s, our new approach would reduce the buffer size requirement to only 116 kbits, and the new start-up delay would be only 0.23 s.

On the other hand, it must be remembered that although different leaky buckets may describe transmission of an encoded bit stream at different peak transmission bit rates, the different leaky buckets do not alter the average bit rate of the encoded bit stream. Thus, although the different leaky buckets for an encoded bit stream may help the decoder to choose the minimal buffer size and startup-delay to play back the stream flawlessly under the prevailing network conditions, once the stream begins to play, the stream's multiple leaky buckets do nothing to adapt the stream's encoded bit rate to fluctuating network conditions. However, the leaky bucket bounds can be used by the decoder to decide, for example, whether its buffer is in danger of underflowing or overflowing under the prevailing network conditions. If so, then the decoder can switch to a new stream with a different average bit rate, and a whole new set of associated leaky buckets. As with any stream switch (e.g., after a seek), the new set of leaky buckets can be used to bound the decoding schedule of the new stream, to ensure that the decoder buffer will not underflow or overflow after the stream switch, given the desired start-up delay.

Fig. 10 shows a further example of an H.264/AVC clip computed with a higher QP. Some of the benefits of the proposed approach in this case are quantified in the caption of the figure.

VI. Conclusions

We have presented a hypothetical reference decoder which is a generalization of those in prior standards. This new GHRD re-

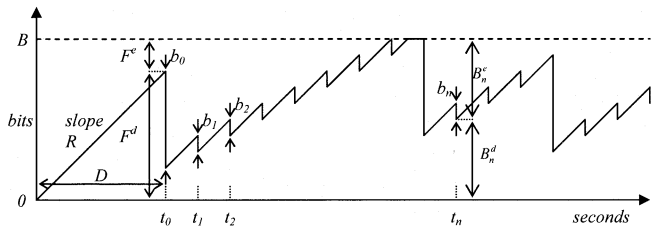

Fig. 11. Decoder buffer state as a function of time.

quires only a few syntax elements at random access points of the bit stream, and provides much higher flexibility for bit stream delivery through today's emerging networks where bandwidth is variable and terminals have a variety of bit rate and buffering capabilities. This new reference decoder enables these new scenarios, while reducing the transmission delay to a minimum for the available bandwidth. In addition, it minimizes the channel bit-rate requirement for delivery to devices with given physical buffer size limitations. In our tests using H.264/AVC video bit streams, we found that the buffer size and the start-up delay for some terminals can be reduced by a factor of 14 to 45, or the peak transmission bit-rate requirement can be reduced by almost a factor of four. The GHRD is applicable to any multimedia (e.g., video, audio) data compression and communication system. It has recently been adopted as part of the H.264/AVC video codec standard specification.

Appendix I

A. Proof That the $R_{\min} - B_{\min}$ Curve is Decreasing and Convex

In this Appendix, we prove that both the minimum decoder buffer size $B_{\min}$ and the minimum decoder buffer delay $D_{\min}$ are decreasing, convex functions of the bit rate R. Therefore, knowing the values of one of these functions at several bit rates, say $B_{\min}(R_1), \ldots, B_{\min}(R_N)$, allows the decoder to linearly interpolate the values to arrive at a value $B_{\text{interp}}(R)$ that is slightly but safely larger than $B_{\min}(R)$.

Our first goal is to derive expressions for $B_{\min}$ and $D_{\min}$ as functions of the bit rate R and an initial buffer state. Fig. 11 illustrates the decoder buffer state as a function of time, where B is the size of the decoder buffer in bits, R is the rate at which bits arrive into the decoder buffer in bits per second, and F^d is the number of bits that the decoder accumulates before extracting and decoding the first frame. Define $F^e = B - F^d$, and let $\{b_i, t_i\}$, $i = 0, \ldots, n-1$, denote the number of bits b_i coded for frame i at time t_i (relative to $t_0 = 0$), where n is the number of frames in the sequence. Then the number of bits B_i^d in the decoder buffer immediately *after* frame i is extracted can be expressed recursively as

$$B_i^d = \begin{cases} B - F^e - b_0, & i = 0 \\ \min\{B, B_{i-1}^d + R(t_i - t_{i-1})\} - b_i, & i > 0. \end{cases}$$

This can be made nonnegative for all $i = 0, \ldots, n-1$ (for fixed F^e and R) by making B sufficiently large. $B_{\min}$ is the smallest such sufficiently large B. That is, for any fixed B

$$B_{\min} = B - \min_i B_i^d.$$

The decoder buffer delay is $D = F^d/R = (B - F^e)/R$. Hence, for fixed F^e and R, the minimum decoder buffer delay is

$$d_{\min} = \frac{B_{\min} - F^e}{R}.$$

To show that these are convex in R and F^e (and various other properties), it simplifies matters slightly to consider the state of the *encoder* buffer as a function of time. The encoder buffer can be considered as a leaky bucket of size B bits that leaks bits at a constant rate R bits per second into the channel (and from there, after a fixed delay, into the decoder buffer). The encoder inserts b_i bits into the bucket at each time t_i corresponding to frame i. The bucket may drain completely before another frame is inserted, in which case no bits are transmitted while the bucket is empty. The bucket begins with initial fullness F^e bits. These initial F^e "dummy" bits are not transmitted. However, they restrict the size of the first few frames i to at most $b_i \le B - F^e + (t_i - t_0)R$ bits, thereby serving to limit the decoding delay. (In a constant bit-rate system, in which the encoder buffer is not allowed to underflow, the initial F^e "dummy" bits would also serve to delay transmission of the first real encoded bits, thereby preventing buffer underflow. In a VBR system, this would not be necessary (see Figs. 1 and 2.) If the bucket size B is sufficiently large, then the number of bits B_n^e in the bucket immediately after frame i is inserted can be expressed recursively as

$$B_i^e = \begin{cases} F^e + b_0, & i = 0 \\ \max\{0, B_{i-1}^e - R(t_i - t_{i-1})\} + b_i, & i > 0. \end{cases}$$

Note that this expression does not depend on the bucket size B.

Lemma 1: For all n, $B_i^d + B_i^e = B$. That is, B_i^d and B_i^e are complementary.

Proof: By induction, using the formulas for B_i^d and B_i^e: Clearly, $B_0^d + B_0^e = B$. For $i > 0$, assume $B_{i-1}^d + B_{i-1}^e = B$. Then it is not difficult to see that $B_i^d + B_i^e = B$. □

Lemma 2: $B_{\min} = \max_i B_i^e$. That is, $B_{\min}$ is the maximum number of bits in the leaky bucket.

Proof: Using Lemma 1, $B_{\min} = B - \min_i B_i^d = \max_i (B - B_i^d) = \max_i B_i^e$. □

Lemma 3: $B_{\min}$ is **monotonically nonincreasing** as a function of R, for fixed F^e.

Proof: Let $R' > R$. We show that $B_{\min}(R') \le B_{\min}(R)$ by first showing that $B_i^e(R') \le B_i^e(R)$ for each i, then invoking Lemma 2. By induction: Clearly, $B_0^e(R') \le B_0^e(R)$. For $i > 0$, assuming $B_{i-1}^e(R') \le B_{i-1}^e(R)$, it is easy to see that $B_{i-1}^e(R') - R'(t_i - t_{i-1}) \le B_{i-1}^e(R) - R(t_i - t_{i-1})$, hence the conclusion follows. □

Lemma 4: $B_{\min}$ is **continuous** as a function of R, for fixed F^e. □

Proof: By induction, $B_i^e(R)$ is continuous in R, for each i. Apply Lemma 2. □

Lemma 5: $B_{\min}$ is **piecewise linear** as a function of R, for fixed F^e.

Proof: By induction, $B_i^e(R)$ is piecewise linear in R, for each i. Apply Lemma 2. □

Lemma 6: $B_{\min}$ is **convex** as a function of R, for fixed F^e.

Proof: By induction, $B_i^e(R)$ is convex in R, for each i. Apply Lemma 2. □

The following lemmas are similarly proved.

Lemma 7: $B_{\min}$ is **monotonically nondecreasing** as a function of F^e, for fixed R.

Lemma 8: $B_{\min}$ is **continuous** as a function of F^e, for fixed R.

Lemma 9: $B_{\min}$ is **piecewise linear** as a function of F^e, for fixed R.

Lemma 10: $B_{\min}$ is **convex** as a function of F^e, for fixed R.

A fortiori, the following can be similarly proved.

Proposition 1: $B_{\min}$ is **convex** as a bi-variate function of R and F^e.

As a bi-variate function of R and F^e, denote $B_{\min} = B_{\min}(R, F^e)$.

Definition 1: $B_{\min}(R) = \min_{F^e} B_{\min}(R, F^e)$. This is the minimum possible buffer size given R.

Definition 2: $F^d_{\min}(R, F^e) = B_{\min}(R, F^e) - F^e$. This is the initial decoder buffer fullness given R and F^e.

Definition 3: $F^d_{\min}(R) = \min_{F^e} F^d_{\min}(R, F^e)$. This is the minimum possible initial decoder buffer fullness given R.

Definition 4: $F^e_{\min}(R) = B_{\min}(R) - F^d_{\min}(R)$. This is the corresponding initial encoder buffer fullness.

Lemma 11: For some $F^e_0(R) \geq 0$,

$$B_{\min}(R, F^e) = \begin{cases} B_{\min}(R), & \text{for } F^e \leq F^e_0(R) \\ B_{\min}(R) + (F^e - F^e_0(R)), & \text{for } F^e \geq F^e_0(R). \end{cases}$$

That is, for fixed R, $B_{\min}$ is a constant as a function of F^e until F^e reaches a breakpoint $F^e_0(R)$, after which point $B_{\min}$ increases as a constant plus F^e.

Proof: By Lemma 7, $B_{\min}(R, 0) = B_{\min}(R)$. Furthermore, by Lemmas 7, 9, and 10, the (right) derivative $\partial B_{\min}/\partial F^e$ is nonnegative and is stepwise increasing. Hence, it suffices to show that $\partial B_{\min}/\partial F^e$ must equal either 0 or 1. In turn, by Lemma 2, it suffices to show that for all i, the (right) derivative $\partial B_i/\partial F^e$ must equal either 0 or 1. Let $j(R, F^e)$ be the first index for which $B^e_{i-1} - R(t_i - t_{i-1}) < 0$. Then by induction, $\partial B_i/\partial F^e = 1$ for $i < j(R, F^e)$, and $\partial B_i/\partial F^e = 0$ otherwise. □

Lemma 12: For some $F^e_0(R) \geq 0$,

$$F^d_{\min}(R, F^e) = \begin{cases} B_{\min}(R) - F^e, & \text{for } F^e \leq F^e_0(R) \\ B_{\min}(R) - F^e_0(R), & \text{for } F^e \geq F^e_0(R). \end{cases}$$

That is, for fixed R, $F^d_{\min}$ decreases as a constant minus F^e until F^e reaches a breakpoint $F^e_0(R)$, after which point $F^d_{\min}$ is constant.

Proof: Follows from Lemma 11 and Definition 2.

Proposition 2: For any R, $B_{\min}(R)$ and $F^d_{\min}(R)$ are simultaneously achieved by $F^e = F^e_0(R) = F^e_{\min}(R)$, and this is the only value of F^e at which both minima are achieved.

Proof: Follows from Lemmas 11 and 12 and Definition 4.

We now turn our attention to $D_{\min} = (B_{\min} - F^e)/R$.

Lemma 13: $D_{\min}$ is **monotonically decreasing** as a function of R, for fixed F^e.

Proof: For fixed F^e, the numerator is monotonically nonincreasing (by Lemma 3), while the denominator is strictly increasing. □

Lemma 14: $D_{\min}$ is **continuous** as a function of R, for fixed F^e.

Proof: Follows from the continuity of $B_{\min}$ (Lemma 4). □

Lemma 15: $D_{\min}$ is **convex** as a function of R, for fixed F^e.

Proof: Except at the finite number of points where $\partial B_{\min}/\partial R$ does not exist, $\partial B_{\min}/\partial R = (\partial B_{\min}/\partial R)/R - (B_{\min} - F^e)/R^2$, by the chain rule. Both terms are negative and increase monotonically to zero, by Lemmas 3 and 6. □

The following are corollaries of Lemma 12.

Lemma 16: $D_{\min}$ is **monotonically nonincreasing** as a function of F^e, for fixed R.

Lemma 17: $D_{\min}$ is **continuous** as a function of F^e, for fixed R.

Lemma 18: $D_{\min}$ is **piecewise linear** as a function of F^e, for fixed R.

Lemma 19: $D_{\min}$ is **convex** as a function of F^e, for fixed R.

APPENDIX II

A. *Matlab Program for Computing the $R\min - B\min$ Curve for a Bit Stream*

```
clear all;
clf;
% bits(i) is number of bits per frame i.
% Frame rate in frames/s of the given stream.
FrameRate  = 30;
% Number of frames.
N = max(size(bits)).
% Calculate data for R − B plot
j = 0;
% Test bit rates from 50 kbps/s to 3 Mbps.
% Assume an initial (dummy) buffer sizeB = R*20..
% buff1 is buffer state before frame removal.
% buff2 is buffer state after frame removal.
% Assume that initially buffer is full :
% F = buff1(1) = B.
for R = 50000 : 50000 : 3000000
    j = j + 1;;
    B = R*20;;
    buff1 = zeros(1, N + 1);
    buff1 = zeros(1, N);
    buff1(1) = B;;
    minbuff = buff1(1);
    for i = 1:1:N,
          buff2(i) = buff1(i) − bits(i);
    if (buff2(i) < minbuff)minbuff = buff2(i);
    end
          buff1(i + 1) = buff2(i) + R/FrameRate;
          if (buff1(i + 1) > B) buff1(i + 1) = B;
          end
    end
 % Minimum buffer size in bits.
 Bmin = B − minbuff;
 % Peak bit rate in kbps.
 X(j) = R/1000;
 % Minimum buffer size in kbits.
 Y(j) = (Bmin)/1000;
 % Simulate leaky bucket to find Fmin.
 % Set buffer size to be its minimum value.
```

```
% Initially assume Fmin is zero.
% Whenever underflow occurs do (1) and (2).
% (1) increase Fmin by underflow amount.
% (2) reset buffer.
B = Bmin;
Fmin = 0;
Buff1(1) = Fmin;
for i = 1:1:N
      buff2(i) = buff1(i) - bits(i);
      if (buff2(i) < 0)
          Fmin = Fmin + (0 - buff2(i))
          buff2(i) = 0;
      end
      buff1(i + 1) = buff2(i) + R/FrameRate;
      if (buff1(i + 1) > B) buff1(i + 1) = B;
      End of loop for R
end
holdoff;
plot(X,Y,' - *');
ylabel('Bmin (kbits)')
xlabel('Rmin (kbits/s)');
hold
axis([0 max(X) 0 max(Y)]);
```

Explanation of the Matlab program follows.

- We initially set the buffer size B to any arbitrary value (in this case the bit rate times 20 s, but we could have chosen any other value).
- We then compute the value of buffer fullness along time using (1.1).
- Next, we determine the smallest buffer size needed to contain the bit stream "$B\min = B - \text{minbuff}$", where minbuff is the minimum value of buffer fullness. Observe that minbuff could be negative, which simply says that the original arbitrary buffer size was too small to contain the bit stream, i.e., $B\min > B$.
- Finally, we compute the minimum initial buffer fullness $F\min$. We start setting $F\min$ to 0 and increase it as much as needed to prevent decoder buffer underflow. $F\min$ will take the smallest value in $[0, B\min]$ for which underflow does not occur

Observe that one could use a very small rate increment in the program above, compute the derivatives of the $B\min(R\min)$ curve at each point, and then determine the critical points at which the derivative does not exist (i.e.,the points where the segments of the piece-wise lines of different slopes join).

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable suggestions. They are also grateful to Dr. G. Sullivan for providing many insightful comments and revisions.

REFERENCES

[1] "Annex B, hypothetical reference decoder," in Video Coding for Low Bit Rate Communication, ITU-T Recommendation H.263, Jan. 1998.

[2] "Annex C, video buffering verifier," in *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video (MPEG-2/H.262)*, 2000, ISO/IEC 138 180-2.

[3] J. Crowcroft, M. Handley, and I. Wakeman, *Internetworking Multimedia*. San Mateo, CA: Morgan Kaufmann, 1999.

[4] J. Keyes, *Webcasting*. New York: McGraw-Hill, 1997.

[5] C.-Y. Hsu, A. Ortega, and A. R. Reibman, "Joint selection of source and channel rate for VBR video transmission under ATM policing constraints," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 1016–1028, Aug. 1997.

[6] A. R. Reibman and B. G. Haskell, "Constraints on variable bit-rate video for ATM networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 361–372, Dec. 1992.

[7] "JVT test model JM," in Joint Video Team (JVT) of ITU-T SG16/Q15 (VCEG) and ISO/IEC JTC1/SC29/WG11 (MPEG), Klagenfurt, Austria, July 2002, Doc. JVT-D147.

[8] H.-M. Hang and J. J. Chen, "Source model for transform video coder and its application—Part II: Variable frame rate coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 299–311, Apr. 1997.

[9] T. Wiegand and G. Sullivan, "Final draft international standard (FDIS) of joint video specification (ITU-T rec. H.264 ISO/IEC 14 496-10 AVC)," in *Joint Video Team (JVT) of ITU-T SG16/Q15 (VCEG) and ISO/IEC JTC1/SC29/WG1, Annex C*, Pattaya, Thailand, Mar. 2003, Doc. JVT-G050, pp. 195–202.

[10] E. Viscito, "HRD and related issues," in *Joint Video Team (JVT) of ITU-T SG16/Q15 (VCEG) and ISO/IEC JTC1/SC29/WG11 (MPEG)*, Klagenfurt, Austria, July 2002, Doc. JVT-D131.

[11] N. Peterfreund, "Time-shift causality constraint on the CAT-LB HRD," in *Joint Video Team (JVT) of ITU-T SG16/Q15 (VCEG) and ISO/IEC JTC1/SC29/WG11 (MPEG)*, Geneva, Switzerland, Oct. 2002, Doc. JVT-E1333.

Jordi Ribas-Corbera (S'95–M'96) received the Enginyer Tècnic de Telecomunicacions degree from the Escola d'Enginyeria La Salle, Barcelona, Spain, in 1990, the M.S. degree in electrical engineering from the University of California, Irvine, in 1992, and the Ph.D. degree in electrical engineering (systems) from the University of Michigan, Ann Arbor, in 1996.

During 1994, he was with the Advanced Video Processing Laboratory, NTT Human Interface Labs, Yokosuka, Japan. From 1996 to 2000, he was with the Digital Video Department, Sharp Labs of America, Camas, WA. He joined Microsoft Corporation in February 2000, where he is currently the Group Program Manager for the Windows Media Codec team in the Digital Media Division. His team develops the core compression and signal processing components for Windows media, such as the Windows Media Audio (WMA) and Windows Media Video (WMV) codecs. He has actively participated in the development of compression standards such as ISO MPEG-4, ITU-T/H.263+, and ITU-T/H.264, and in industry consortia such as the DVD Forum or MPEG-LA's MPEG-4 Visual Patent Holders group. He is the author of numerous contributions to standards and peer-reviewed technical papers in academic conferences and journals, and has been an invited speaker for a number of industrial conferences and seminars, such as at Cable Labs, EBU, NAB, and SMPTE. He has been awarded seven patents and has eight pending.

Dr. Ribas received the Young Investigator Award in the 1997 IS&T/SPIE International Conference on Visual Communications and Image Processing, and the Sharp Labs President's Award in 1999.

Philip A. Chou (SM'87–M'87–SM'00) was born in Stamford, CT, in 1958. He received the B.S.E. degree from Princeton University, Princeton, NJ, in 1980, the M.S. degree from the University of California, Berkeley, in 1983, both in electrical engineering and computer science, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1988.

Since 1977, he has worked for IBM, AT&T Bell Laboratories, Princeton Plasma Physics Lab, Telesensory Systems, Speech Plus, Hughes, Xerox, VXtreme, and Microsoft, where he was variously involved in office automation, motion estimation, character recognition, speech compression, LPC and text-to-speech synthesis, compression of digitized terrain, speech and document recognition, and multimedia network communication. His research interests are data compression, pattern recognition, and multimedia processing and communication. During 1994-1995, he was a consulting Associate Professor at Stanford University. Since 1998, he has been an Affiliate Professor at the University of Washington, Seattle. Currently, he is with Microsoft Corporation, Redmond, WA.

Dr. Chou serves on the IEEE Technical Committee for Image and Multidimensional Signal Processing (IMDSP). From 1998 to 2001, he served on the Editorial Board of the IEEE TRANSACTIONS ON INFORMATION THEORY as an Associate Editor for Source Coding. He is the recipient (with Tom Lookabaugh) of the 1993 Signal Processing Society Paper award. He is a member of Phi Beta Kappa, Tau Beta Pi, Sigma Xi, and the IEEE Computer, Information Theory, Signal Processing, and Communications societies, and was an active member of the MPEG committee.

Shankar L. Regunathan received the B.Tech degree in electronics and communication from the Indian Institute of Technology, Madras, in 1994, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Santa Barbara, in 1996 and 2001, respectively.

Currently, he is with Microsoft Corporation, Redmond, WA.