ECHO: A Community Video Streaming System with Interactive Visual Overlays

Gene Cheung[a], Wai-tian Tan[b], Bo Shen[b] and Antonio Ortega[c]

[a] HP Labs Japan, 3-8-13, Takaido-higashi, Suginami-ku, Tokyo, Japan 180–0022;
[b] HP Labs Palo Alto, 1501 Page Mill Rd, Palo Alto, CA 94304;
[c] University of Southern California, 3740 McClintock Ave., EEB 436, Los Angeles, CA 90089-2564

ABSTRACT

We describe a networked video application where personalized avatars, controlled by a group of "hecklers", are overlaid on top of a real-time encoded video stream of an Internet game for multicast consumption. Rather than passively observing the streamed content individually, the interactivity of the controllable avatars, along with heckling voice exchange, engenders a sense of community during group viewing. We first describe how the system splits video into independent regions with and without avatars for processing in order to minimize complexity. Observing that the region with avatars is more delay-sensitive due to their interactivity, we then show that the regions can be logically packetized into separable sub-streams, and be transported and buffered with different delay requirements, so that the interactivity of the avatars can be maximized. The utility of our system extends beyond Internet game watching to general community streaming of live or pre-encoded video with visual overlays.

1. INTRODUCTION

Video conferencing and video streaming are two important classes of networked video applications with different purposes. The goal of video conferencing is inter-personal communication, while streaming applications typically involve passive consumption of video content. Individual viewing of video with no interactivity with others is sometimes called *individual streaming*.[1] While communicating to associates and watching video are distinct objectives, they are not mutually exclusive; in cases of live sporting or gaming events, one may prefer to watch the same streaming video with other viewers while communicating with them at the same time. We call such a model *community streaming*. It is similar to watching TV with family members in the living room and sharing comments on the viewed content, except the viewers in community streaming may not be in the same physical location. It is unlike video conferencing in that the primary activity is content viewing.

To enhance the multimedia experience of community streaming, in addition to verbal comments by users captured in audio, one can support *visual overlays* of text, image, or video on the streaming content such that each overlay, controlled by a user and viewable to the streaming group, is reflective of the user's comments, mood or presence. Continuing with the earlier sporting example, we call users of these overlays *hecklers*—viewers of sporting events with frequent and often outrageous comments to the amusement of others—and the overlays *heckles*. Beyond sporting events, another use case of visual overlays is game shows, where viewers can compare their answers with those of the show participants and share them with other group members. Yet another example is a shared "whiteboard" on a background of streaming pre-encoded video, where a viewer can control the playback and visually highlight different parts of the video with an electronic pen.

From the above examples, it is clear that the heckles need to be interactive while the primary streaming presentation is viewed non-interactively. It is well known that the transmission requirements are much more stringent for interactive videos than non-interactive ones. It is therefore a key technical challenge for community streaming with visual overlays to exploit the heterogeneity in transmission requirements for interactive and non-interactive traffic.

One approach to community streaming with visual overlays is to employ multiple video streams: one stream for the main presentation, and one for each visually overlaid heckle. The streams are encoded and delivered independently, and each heckler is responsible for decoding, joint rendering and synchronization of the streams.

Clearly, the non-interactive main presentation can be treated appropriately and differently from the interactive heckles, since they belong to distinct streams. Clients with such capabilities already exist, e.g., SMIL-capable clients. Nonetheless, this multiple-stream solution has several drawbacks. First, it demands complexity that scales linearly with the number of heckles, which may be too expensive for mobile devices such as smartphones. Second, this approach also incurs inefficiency in bandwidth usage, since visual information to be occluded by overlays is also transmitted.

In this paper, we consider a one-stream approach where the interaction among the hecklers *leads to modifications in a shared stream*. In so doing, community streaming can be extended to simpler but ubiquitous clients that are uni-stream capable. This approach also avoids inefficiency in bandwidth usage by the transmission of occluded regions. Nevertheless, treating each video frame in the shared stream with homogeneous delay requirement would require that all video data be transmitted as interactive traffic. Instead, we present algorithms and systems that distinguish different delay requirements for different regions of the *same video frame* in the shared stream. This allows us to exploit the inherent heterogeneity in transmission requirements of community streaming without resorting to the use of multiple streams.

To provide a practical example of community streaming, we describe a streaming system called *ECHO (Enabling Community of Hecklers and Observers)* that can efficiently support multiple communities of hecklers with visual overlays, with the primary streaming video being live Internet games. The heckles are in the form of pre-captured talking heads of hecklers (similar to picture-in-picture) and real-time controllable *avatars*[2] that are overlaid on top of the shared streaming video. ECHO is novel in recognizing that even within a single compressed video stream, the parts with and without heckles have different processing and delay requirements, and can be treated accordingly. Doing so rewards us with both coding and transport benefits: i) the re-encoding of video downstream in the distribution network to overlay heckles can be done using a smaller sub-stream, lowering the stream processing complexity; and ii) exploiting the difference in delay requirements of the two sub-streams, those containing regions of video with and without heckles, allows us to deliver them to the hecklers using different transmission strategies, thus improving heckling interactivity. As a proof of concept, a fully functional prototype of ECHO has been built on top of HP's OpenCall Media Platform (OCMP) version 4.0, using a live game sequence from popular game `Counter-Strike`[3] as the streaming video source. We have tested our prototype with standard SIP[4] video clients on notebook computers running Windows XP and handheld devices running Windows Mobile in real network environments.

Our proposed separable streaming techniques provide considerable gains over simply selecting transport parameters to accommodate interactive delivery of the whole stream. For example, overall channel capacity can be reduced by up to 50% for a given set of delay constraints. Alternatively, for a given channel bandwidth, much lower interactive delay can be achieved (e.g., 0.1 seconds versus close to 10 seconds).

The outline of the paper is as follows. We first discuss related work in Section 2. We then give an overview of the ECHO system in Section 3, which introduces the two main ECHO components, the *Observer View Generator* and the *Heckling Coordinator*. Sections 4 and 5 then define coding and transport strategies, respectively, to support our system architecture. Results that quantify different tradeoffs in our design and conclusions are provided in Sections 6 and 7, respectively.

2. RELATED WORK

Many techniques have been proposed for distribution of streaming media content, from the simple server-client model, to content distribution networks and peer-to-peer networks. However, much of this work has focused on the distribution of the *same* content to different audiences. This traditional usage model has become insufficient and limiting in a number of new contexts.[1,5] To address the heterogenous capability of different handsets and preference of different users, the notion of "collaborative streaming" has been introduced,[1] where a streaming session may involve several heterogenous devices, each receiving the stream in the same playback time but in their own preferred resolution, bit-rate and language. Device discovery, session initialization, and session mobility (including seamless transfer to different devices) are further discussed.[5] ECHO advances a different aspect of the traditional usage model of streaming video by supporting personalized customization via visual overlay on a per group basis.

Recent years have witnessed the growth of networked games like `Counter-Strike` and `Sauerbraten`,[3,6] particularly in Korea and Japan. Along the way, exceptionally skilled and charismatic game players have developed cult followings, and game watching has become a popular if not obsessive pastime for many. One example is Half-Life Television,[7] which allows passive viewing over the Internet of real-time games by a large number of audience. Extending on our previous work,[8] ECHO further enhances the game viewing experience by overlaying interactive avatars or talking heads in real-time to create a heckling community for group communication.

The advantages of variable bit rate (VBR) encoding of video are well known, but come at the cost of additional end-to-end delay and/or bandwidth for delivery over networks.[9] Thus, streams that have tight delay requirements will have to be encoded under low delay constraints (and consequently worse overall video quality) or will require a VBR channel for transmission. Analysis of delay-bandwidth-quality tradeoffs has been considered from a number of different perspectives, including effective bandwidth,[10] smoothing[11] and rate control.[12] For the most part this past work focuses on the properties of a single media source, for which bandwidth usage, quality and end-to-end delay are evaluated. A key novelty of the present work is to consider scenarios where different end-to-end delay characteristics co-exist within a given stream.

Chang and Messerschmitt[13,14] proposed the concept of delay-cognizant video coding (DCVC), where different delay constraints are applied to different parts of an encoded video stream. DCVC is related to our proposed method in that different sub-streams are identified, and also leads to lower effective bandwidth requirements once the delay constraints are loosened on part of the bitstream. There are two fundamental differences with respect to our method. First, DCVC[13,14] sub-streams are separated based on the video characteristics, e.g., high delay blocks in the video are those where video contents change slowly. Instead, our high delay segments will contain areas that are not modified to insert hecklers. Second, DCVC[13,14] data for a given frame can be played back at different times (e.g., high delay packets can be used even after the frame they correspond to has been displayed). Instead, we follow a more traditional delay constraint, where packets for one frame have to be available before the frame is decoded in order to be useful. Thus, in our case, additional end-to-end delay is made possible by starting transmission of the unmodified video early, while the modified video is available for transmission a short time before its scheduled playback time. Both modified and unmodified videos are played synchronously.

3. ARCHITECTURE OVERVIEW

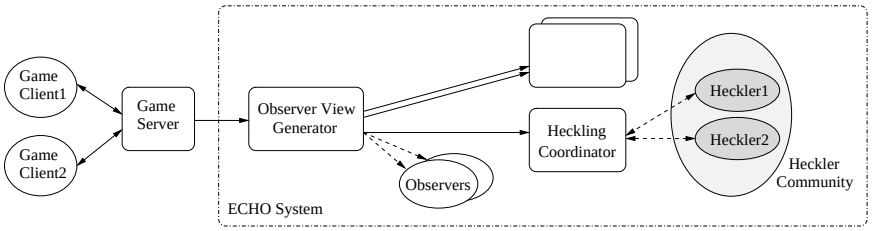
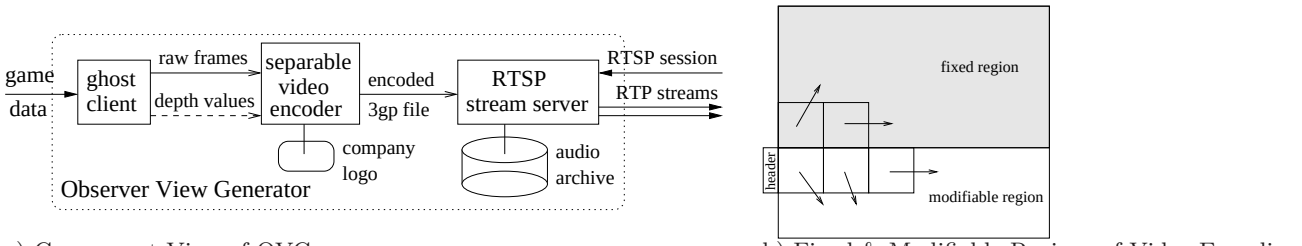


Figure 1. The ECHO system consists of an Observer View Generator to render live game content into streaming video, and Heckling Coordinators (HC) to support customized visual overlays or heckling. A user can be a passive game "Observer" or an active "Heckler".

A system overview of ECHO is shown in Figure 1. A networked game is hosted by a *game server*, and typically involves two or more game players who control the *game clients*. The game server and the game clients typically communicate in proprietary game-specific protocols. To support game viewing for non-players over a wide range of devices, it is desirable to convert the game content into standard-compliant video streams. This is achieved by the *Observer View Generator* (OVG)[8] whose component view is depicted in Figure 2(a). Inside OVG is a *ghost client* that receives real-time gaming data from the game server and draws frames of the game sequence from a chosen perspective. The generated frames are then encoded in real-time by a "separable video encoder" (see Section 4) and subsequently streamed using RTP.[15] Upon request, a RTSP streaming server sets up a RTSP session, and sends two RTP streams, one each for video and audio, to the requesting observer or Heckling Coordinator (HC). The RTSP server can either use a live audio stream from the network game, or use a pre-stored audio file from its audio archive for an alternate multimedia experience. Notice that any processing delay is inconsequential, as neither the observers nor the hecklers interact with the game players.



a) Component View of OVG b) Fixed & Modifiable Regions of Video Encoding

Figure 2. Component View and Video Coding of Observer View Generator

Observers of the game, without community-style interaction, can directly contact the OVG to start a passive viewing session. It is also possible to employ streaming proxies to reduce network resource consumption.[7] Alternatively, a group of *hecklers* can contact a HC to initiate a heckling session. HC in turn requests the live encoded stream from OVG, overlays personalized heckles on top, as shown in Figure 3, and multicasts the enriched media to members of the newly formed heckling community. Each heckler can control his/her heckles by sending commands in real-time.

Figure 3. The Heckling Coordinator overlays controllable avatars and talking heads on game content `Counter-Strike`.

HC is responsible for receiving heckling input from the users and then constructing a new video stream overlaid with corresponding heckles so that participation from simple standard-based SIP clients[4] is possible. A straightforward implementation of HC would employ a video decoder to decode game video from an OVG, overlay heckles in the pixel domain, then encode the resulting video in a standard manner. Two unique characteristics of our particular video content calls for a different implementation. First, the heckles tend to only occupy a small portion of the screen as the primary activity is game observation. This means that re-encoding parts of the video that have not been modified by the heckles can be wasteful in terms of computation resources, and can lead to lower visual quality in general. Instead, we employ *partial transcoding* whereby only the necessary part of a video stream is processed. Second, the game sequence is being watched passively, and similar to television shows, can be buffered or delayed for several seconds at HC without adversely affecting the perceived latency at the hecklers. The heckling action, on the other hand, is interactive. It is desirable to exploit the difference in characteristics of the two "parts" of the content to match characteristics of available delivery options, as will be discussed further in Section 5. To exploit both characteristics, we rely on the production of "separable video" (Section 4.1) by OVG, which allows easy separation of compressed bitstreams into independently decodable regions.

The heckles are controlled by dual-tone multi-frequency (DTMF), which extends usage of the service to generic video-streaming capable cellular phones. As shown in the *heckler overlay* unit of Figure 4, the overlay operation is performed in the pixel domain to a portion of the video frames. Pixel domain operations support a rich set of user actions such as changes in avatar location, speed, size, and opacity. These are generally computationally expensive, but the use of partial encoding helps to reduce overall complexity. The heckler overlay unit is jointly controlled by every user in the same group.

Several characteristics of the system architecture are worth noting. First, encoding of the live game content is performed only once, even with multiple spurred heckling sessions, each coordinated by a different HC. This translates to large computation savings, as encoding of video is much more resource intensive than decoding. Second, though our application of interest relates to game watching, the system is generally applicable to other

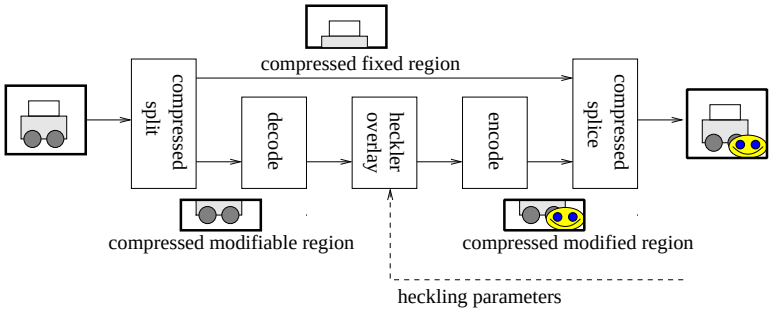


Figure 4. The *Heckling Coordinator* employs partial transcoding in which independent sub-streams are obtained from a separable video source. This allows differential treatment in transport and avoids unnecessary re-encoding.

content. For example, sporting broadcast can replace the game server to feed multimedia data into OVG. Third, to facilitate deployment, observers can be standard RTSP video clients,[16] while hecklers can be standard SIP clients[4] with DTMF capabilities to control avatars. No software download and installation is necessary for client devices. Finally, for wireless cellular network deployment, HC can be implemented as a component of IP Multimedia Subsystem (IMS)[17] inside the wireless core network of 3GPP Release 6 or later.

We now discuss coding (Section 4) and transport (Section 5) strategies to support our system architecture.

4. CODING FOR SEPARABLE VIDEO STREAMS

4.1 Encoding Separable Video

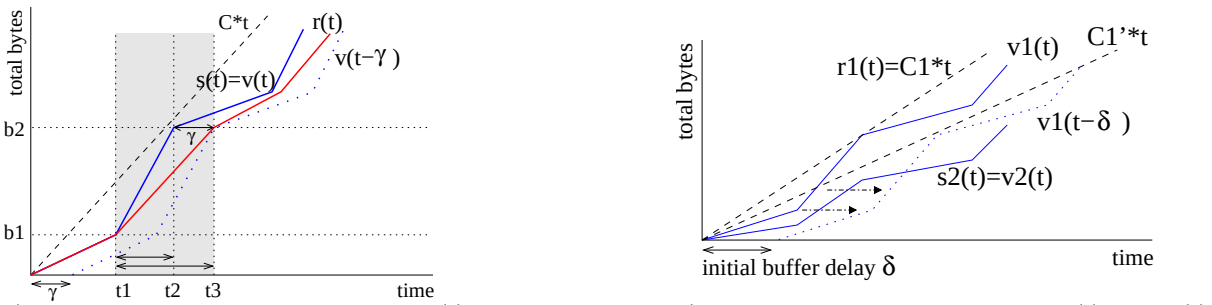
The video format chosen for our current implementation is H.263. This is preferred over more recent standards such as H.264/AVC[18] for two reasons. First, most existing video handsets support H.263 while H.264 is only supported in a small group of high-end devices. Second, the required encoding complexity for H.264/AVC is drastically higher than H.263, limiting the cost effectiveness of an H.264 implementation. Note however that our recent work[19] does provide some promising tools to reduce the H.264/AVC encoding complexity by taking advantage of depth information available as the game video data is being produced.

Since the primary goal of a user is game watching, with heckling being a secondary objective, it is natural to assume that the hecklers will occupy a relatively small portion of the screen, as illustrated in Figure 3. For convenience of discussion, we assume the overlay of hecklers at the HC would only happen on several macroblock (MB) rows at the bottom of a video. The video content is encoded into two independent or *separable regions*: a *fixed region* that is comprised of the top several MB rows, and a *modifiable region* consisting of the remaining MB rows. MBs in the fixed and modifiable regions are encoded in such a way that a motion vector (MV) of a MB does not point outside of the region to which the MB belongs. Specifically, MBs in the bottom MB row of the fixed region are restricted not to have MVs pointing downwards, while MBs in the top MB row of the modifiable region are restricted to have no MVs pointing upwards, as illustrated in Figure 2(b). Doing so allows a downstream HC to only search for and decode the modifiable region, *without* decoding the fixed region. A Group of Block (GOB) header (that is part of H.263 syntax) is inserted at the beginning of the first MB row in the modifiable region to facilitate identification of the region during bitstream parsing, and to ensure the modifiable region is independently decodable.

Note that all modifications done here to enable separable regions in a frame result in a standard-compliant bitstream, so that any observer (i.e., both passive observers and active hecklers) can receive the encoded stream and decode it without any knowledge of separable regions. Note further that such coding restriction of MVs may lead to a decrease in coding performance, which will be addressed in Section 6.

4.2 Partial Transcoding

Given separable video input from OVG, HC performs simple bitstream parsing (without decoding) to split the compressed bitstream into two regions, one that is to remain unchanged (fixed region) and one where heckles are to be added (modifiable region). Decoding, adding of heckles, and subsequent encoding are only performed on the modifiable region. Clearly, this reduces the computing load at HC. Although the use of separable video



a) transmission of non-separable video $v(t)$ b) transmission of separable video $v_1(t)$ and $v_2(t)$

Figure 5. Transmission of Non-separable and Separable Video over CBR Channel

generally incurs a penalty in compression efficiency, avoiding the re-encoding of the fixed region can lead to overall higher video quality at the user (more details in Section 6). An illustration of the steps involved is shown in Figure 4.

5. TRANSPORT AND BUFFERING OF SEPARABLE VIDEO STREAMS

Our application has the unique characteristic of combining video packets with different latency requirements within the same stream. Specifically, since the heckling actions do not affect game play, the region without overlay of hecklers can be transmitted ahead of time in high-latency mode. Even sizeable delay in the order of seconds is feasible, with the only limitation of not introducing excessive start-up latency. Conversely, regions that are being modified in response to commands issued by the hecklers need to be conveyed to all hecklers quickly, in order to provide an interactive community experience.

As mentioned earlier, most of the area of the displayed frames will not be modified to overlay heckling characters. Thus, forcing all packets to follow strict deadlines can be potentially very inefficient. In particular, longer latencies allow for techniques developed for non-interactive streaming to be used, with lesser need for overhead for packet-loss recovery (e.g., lost packets can always be retransmitted so the overhead matches actual network behavior).

Even when packet losses need not be considered, gains are possible by exploiting statistical multiplexing. Note that better video quality can be achieved when operating the encoder in variable bit rate (VBR) mode.[9] In order to transmit VBR encoded video one needs to enable rate averaging by either (i) supporting VBR transmission (i.e., allowing variable bandwidth) or (ii) averaging the rate needs over time (i.e., using buffering). In our proposed system both modalities are supported, leading to improved quality. First, the longer latency enables the unchanged part of the stream to benefit from a large buffering, so that VBR encoded data can be transmitted with constant or near constant bandwidth. Second, while the modified part of the stream is transmitted under a tight deadline, better performance can be achieved by letting the bandwidth vary. Thus, the overall bandwidth can be constant but we allow the low latency part of stream to be transmitted at a variable rate. The high latency part of the stream can absorb these variations (in its own rate and in the bandwidth) thanks to buffering.

5.1 Streaming Non-separable Video Over Constant Bit-rate Channel

In a system with live-encoded content where the transmission channel capacity and receiver buffer delay constraint are known, the encoder can perform *rate control*,[12,20] at a cost of visual distortion due to quantization, so that these constraints can be met when delivering the encoded video over the channel. For the sake of simply illustrating the potential benefits of separable video over non-separable video, we instead assume video is encoded *a priori*, and investigate the corresponding minimum receiver buffer delay required (to be defined) for a lossless, zero-delay, constant bit-rate (CBR) channel. We assume the channel has capacity C at least as large as the average video encoding rate. Further, we assume that a heckler can send a heckling command to HC with zero-delay, and that HC can overlay heckles on video the instant the heckling command is received.

Though a known prior art, let us first consider streaming of interactive, non-separable video to motivate separable video. Suppose we know the *bit-rate profile* $v(t)$ of such a non-separable video as shown in Figure 5(a).

$v(t)$ is the cumulative byte total for the first t seconds of the video. Assuming sender (HC) always attempts to deliver the interactive video as soon as it is encoded, $v(t)$ is then the same as the *sender buffer aggregate* $s(t)$, i.e., the total number of bytes sender has allocated to the sender buffer up till t. This also means sender can send no more than $v(t_o)$ at t_o, since bytes of video $v(t) - v(t_o), t > t_o$ has not been generated yet and are not in the sender buffer.

Clearly, the *receiver buffer aggregate* $r(t)$—the cumulative byte total receiver has received by t—cannot exceed sender buffer aggregate $s(t)$. Moreover, if at times sender buffer is completely drained before channel bandwidth exhaustion, then transmission will not always be at full capacity, and $r(t) < C * t$ for $t \geq t_o$, where t_o is the first occurrence of such sender buffer depletion. Compare this to streaming of stored video, where the sender buffer is always non-empty; in such case, receiver buffer always receives at channel capacity, or $r(t) = C * t$ for $t \geq 0$.

For interactive, non-separable video, we can derive $r(t)$ given $v(t)$ and C as follows. Let T be the frame duration of the video so that the first n video frames, $F_1, \ldots, F_n$, would have cumulative byte size of $v[n] = v(nT)$. After frame F_1 is produced at time T, the sender can send the minimum of unit capacity $C * T$ and F_1's frame size $v[1]$. If frame size $v[1]$ is larger than unit capacity $C * T$, then the *sender buffer* contains $l[1] = v[1] - C * T$ leftover bytes, which need to be sent in future transmission intervals. Generalizing the analysis, we get:

$$r[n] = \begin{cases} 0 & \text{if } n = 0 \\ r[n-1] + \min\{C * T, v[n] - v[n-1] + l[n-1]\} & \text{o.w.} \end{cases} \quad (1)$$

The sender buffer size $l[n]$ is:

$$l[n] = \begin{cases} 0 & \text{if } n = 0 \\ (v[n] - v[n-1] + l[n-1] - C * T)^+ & \text{o.w.} \end{cases} \quad (2)$$

where $(x)^+ = x$ if $x \geq 0$, and 0 otherwise.

Figure 5(a) illustrates $r(t)$ given $v(t)$ and C. Up until t_1, $r(t)$ grows as fast as $v(t)$. From t_1 to t_2, encoded $v(t)$ churns at a faster rate than the channel can handle, and so $r(t)$ takes until t_3 to catch up to the sending of b_2 total bytes. This means that for the receiver to playback at the same rate as the encoder, as shown by $v(t-\gamma)$ in the figure, it will require a *receiver buffer delay* of $\gamma = t_3 - t_2$. In general, the *minimum receiver buffer delay* γ_{int} required at the heckler for playback without receiver buffer underflow is:

$$\gamma_{int} = \max_b \left\{ r^{-1}(b) - v^{-1}(b) \right\} \quad (3)$$

where $r^{-1}(b)$ and $v^{-1}(b)$ are the inverse functions of $r(t)$ and $v(t)$, respectively. The larger γ_{int}, the poorer the interactivity is for the hecklers; we henceforth call γ_{int} the *response lag*. To avoid any response lag, i.e., $\gamma_{int} = 0$, the CBR channel must have capacity C such that:

$$C * T \geq \max_{n \geq 1} \{v[n] - v[n-1]\} \quad (4)$$

In other words, the channel must be large enough to absorb the largest frame in the video sequence. Here we see the fundamental limitation of interactive non-separable VBR video: at low channel capacity, it suffers from large response lag, at high channel capacity, it under-utilizes the channel.

5.2 Streaming Separable Video over CBR Channel: Separate Channel Transmission

Now suppose instead that we use separable video divided into non-interactive and interactive sub-streams, with bit-rate profiles $v_1(t)$ and $v_2(t)$, respectively. Suppose further that we use bandwidth C_1 and C_2, $C_1 + C_2 = C$, to independently transmit $v_1(t)$ and $v_2(t)$.

For the non-interactive sub-stream $v_1(t)$, we assume the sender buffer has all N video frames*, or $s_1(t) = v_1(NT), t \geq 0$. In order to avoid receiver buffer underflow when sending non-interactive sub-stream over a CBR channel, the channel must have bandwidth C_1 such that $C_1 * t \geq v_1(t), t \geq 0$, as shown in Figure 5(b). For non-interactive sub-stream, this is a necessary and sufficient condition, because, as mentioned earlier, streaming of stored video means receiver buffer always receives at channel capacity, or $r_1(t) = C_1 * t$. We see an immediate

*For real-time encoded video, $s_1(t) = v_1(t + \Delta)$, where Δ is the amount of real-time data in time HC caches after receiving video from OVG before starting the heckling session. For practical purposes, we assume Δ is large enough so that the necessary and sufficient condition to-be-discussed still holds.

benefit over interactive non-separable stream: non-interactive sender buffer $s_1(t)$ always has enough data to send at full channel capacity $C_1 * t$ at time t to transport non-interactive sub-stream, which we could not do previously when the entire stream $v(t)$ was interactive. We call this benefit the *capacity-usage gain*.

Moreover, we can introduce an *initial buffer delay* $\delta_{init} >> \gamma_{int}$, in the order of several seconds, which is the maximum amount of time a heckler is willing to wait before the start of the heckling session. During the initial buffer period δ_{init}, the non-interactive sub-stream is sent. In contrast, the interactive sub-stream starts streaming at $\delta_{init} - \gamma_{int}$. Though both are receiver buffer delay, the initial buffer delay and the response lag affect user experience differently: the initial buffer delay makes the user wait before the start of the streaming session, while the response lag causes a delay between the time a heckler inputs a command and the time the command is manifested as visual overlays in the streaming video. Mathematically, δ_{init} relaxes the bandwidth requirement of the non-interactive sub-channel to a less stringent constraint $C_1' * t \geq v_1(t - \delta_{init}), t \geq \delta_{init}$. Since $v_1(t)$ is by definition monotonically non-decreasing, in general a smaller $C_1' < C_1$ would suffice, as shown by $C_1' * t$ in Figure 5(b). We call this gain in bandwidth using initial buffer delay the *asymmetric-delay gain*. We can now adopt the following **Algorithm 1** to divvy up the available bandwidth C among the two sub-streams:

1. Determine the largest tolerable initial buffer delay δ_{init}.
2. Find the smallest C_1' such that $C_1' * t \geq v_1(t - \delta_{init}), t \geq \delta_{init}$. Send non-interactive sub-stream at C_1' for $t \geq 0$.
3. Let $C_2' := C - C_1'$. Find interactive delay γ_{int} using (3). Send interactive sub-stream at C_2' for $t \geq \delta_{init} - \gamma_{int}$.

Having started buffering sub-streams at $t = 0$, the heckler starts the heckling session at $t = \delta_{init}$, viewing video corresponding to $t = 0$. Since $C_1' * t \geq v_1(t - \delta_{init})$, there will never be receiver buffer underflow of the non-interactive sub-stream. Because we devoted the maximum amount of bandwidth to the interactive sub-stream (without causing receiver buffer underflow of the non-interactive sub-stream), we have effectively minimized the response lag of the interactive sub-stream.

Note that the use of initial buffer delay $\delta_{init} >> \gamma_{int}$ alone can benefit separable video even if we assume HC cannot read ahead in the non-interactive sub-stream, or $s_1(t) = v_1(t)$. This would be the case when the observed video is live-encoded and HC performs no content caching before starting a heckling session. In this case, the receiver buffer aggregate $r_1(t)$ at the heckler can be similarly derived using $v_1(t)$ and capacity C_1 as done earlier for interactive stream. Then, the minimum bandwidth C_1' for non-interactive sub-stream must be selected such that:

$$\delta_{init} \geq \max_b \left\{ r_1^{-1}(b) - v_1^{-1}(b) \right\} \tag{5}$$

Because $\delta_{init} >> \gamma_{int}$, the minimum required C_1' will be smaller than when $\delta_{init} = \gamma_{int}$. That means more bandwidth can be allocated to the interactive sub-stream, and as a result the response lag can be made smaller than for the non-separable stream. That means asymmetric-delay gain can be achieved independently from capacity-usage gain.

5.3 Streaming Separable Video over CBR Channel: Shared Channel Transmission

Given that we are using one channel to send both sub-streams, we can actually multiplex the two sub-streams to share the total bandwidth at every time instant t. More precisely, we use the following **Algorithm 2**:

1. Determine the largest tolerable initial buffer delay δ_{init}.
2. Send non-interactive sub-stream at capacity C until $t = \delta_{init} - \gamma_{int}$.
3. For $t \geq \delta_{init} - \gamma_{int}$, send interactive sub-stream until sub-stream exhaustion or until C_2' is expended. Send non-interactive until remaining channel capacity is expended.

We can determine C_2' and γ_{int} to guarantee no receiver buffer underflow as follows. In the separate channel transmission case, interactive sub-stream $v_2(t)$ has unused bandwidth $e[n]$ at time $t = nT$ due to its VBR nature:

$$e[n] = \left(C_2' * T - v_2[n] - v_2[n-1] - l_2[n-1] \right)^+ \tag{6}$$

In the shared channel case when using Algorithm 2, non-interactive sub-stream can use $e[n]$'s for transmission of $v_1(t)$, and the condition to avoid receiver buffer underflow for non-interactive sub-stream is now more lenient:

$$(C - C_2') * nT + \sum_{m=1}^{n} e[m] \geq v_1(nT - \delta_{init}) \qquad \forall n \geq 0 \tag{7}$$

To minimize interactive delay, we find the largest C_2' for interactive sub-stream, such that (7) still holds to avoid non-interactive sub-stream buffer underflow. We can then find γ_{int} using (3).

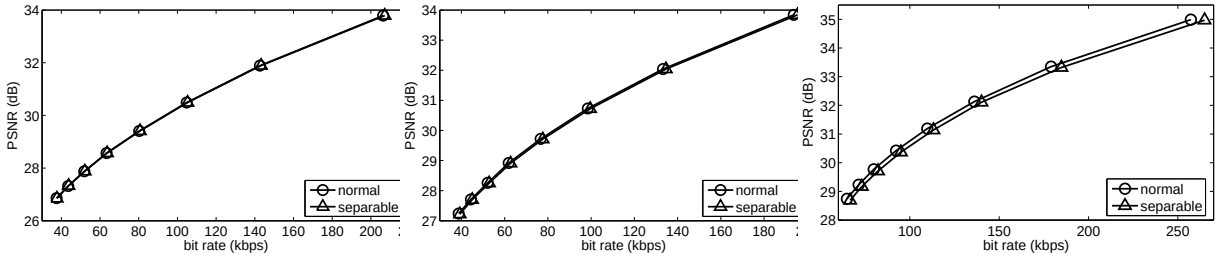
Would Algorithm 2 perform better than Algorithm 1? Clearly, in the separate channel transmission case when interactive sub-stream under-utilizes the channel, in the shared channel transmission case non-interactive sub-stream can capitalize on this unused bandwidth so that its original share of the bandwidth can be reduced while avoiding buffer underflow. That means we can shift more overall capacity to handle the burstiness of the interactive sub-stream to lower response lag. We call the gain of multiplexing interactive and non-interactive sub-streams into the same channel the *stream-multiplexing gain*.

Note, however, that if the interactive sub-stream is itself CBR, then the sub-stream will never under-utilize a sub-channel of the same CBR bandwidth. Although CBR video encoding comes with a hefty cost of variable video quality, this represents an improvement in transport for Algorithm 1 over VBR interactive sub-stream because the sub-channel is now fully utilized. For Algorithm 2, however, because $e(t) = 0$, non-interactive sub-stream has no extra bandwidth to gain by multiplexing. In this case then, Algorithm 2 will perform similarly as with VBR interactive sub-stream.

6. RESULTS

To investigate various aspects of the ECHO system, we discuss in this section the experimental setup and its associated results. We first discuss coding related performance comparing non-separable and separable video in Section 6.1. Thereafter, we investigate the streaming related performances by simulating the interactive streaming of these videos under various application scenarios. All test sequences are QCIF (176x144) and coded at 10 fps, unless otherwise noted.

6.1 Coding and Recoding of Separable Video

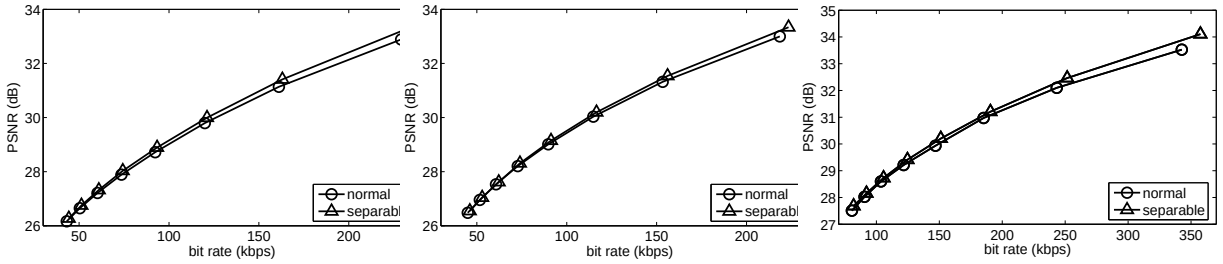


(a) `Counter-Strike 1` (b) `Counter-Strike 2` (c) `Foreman`

Figure 6. Loss in Coding Efficiency in PSNR when Using Separable Video for Different Video Sequences.

Generally, restricting motion vectors to create separable video as discussed in Section 5.2 incurs a loss in coding efficiency. Figure 6 characterizes such loss for several video sequences. We see that such loss is negligible for two sequences of the game `Counter-Strike`, as shown in Figure 6(a) and (b). Sequences `Mother and Daughter`, and `Mobile Calendar` at 30 fps show similar characteristics (not shown) with the rate-distortion curves of normal and separable cases being virtually identical. For the `Foreman` sequence (30 fps), shown in Figure6(c), the PSNR loss is more pronounced at about 0.2dB.

One advantage of using separable video is the ability to selectively process only the modifiable region of the video at the Heckling Coordinator. For video coded using normal methods, it is necessary to re-encode the entire frame even though only parts of the picture changed. This requires additional computation and introduces *regeneration loss* compared to the use of separable video, where the fixed region is not re-encoded. Figure 7 shows the PSNR gain when separable video is employed compared to normal video encoding at the HC. We notice that there is generally a 0.1 to 0.2 dB improvement at usable qualities, such as around 32 dB. Notice that for the `Foreman` sequence, the PSNR gain is achieved over an initial loss in PSNR of Figure 6(c). We thus conclude that using separable video in our ECHO system incurs no loss in compression efficiency.



(a) `Counter-Strike 1` (b) `Counter-Strike 2` (c) `Foreman`

Figure 7. Gain in Coding Efficiency in PSNR when Separable Video are Subsequently Re-encoded.

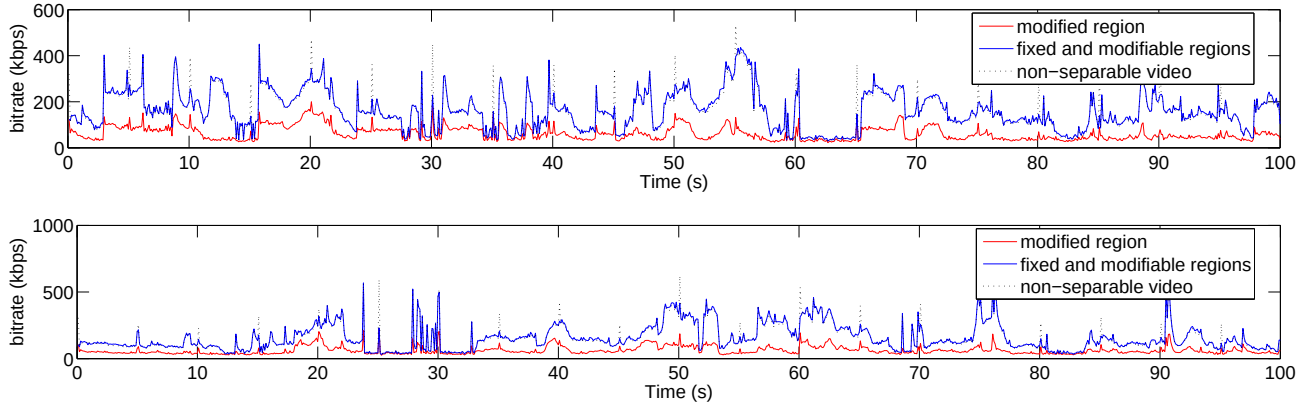


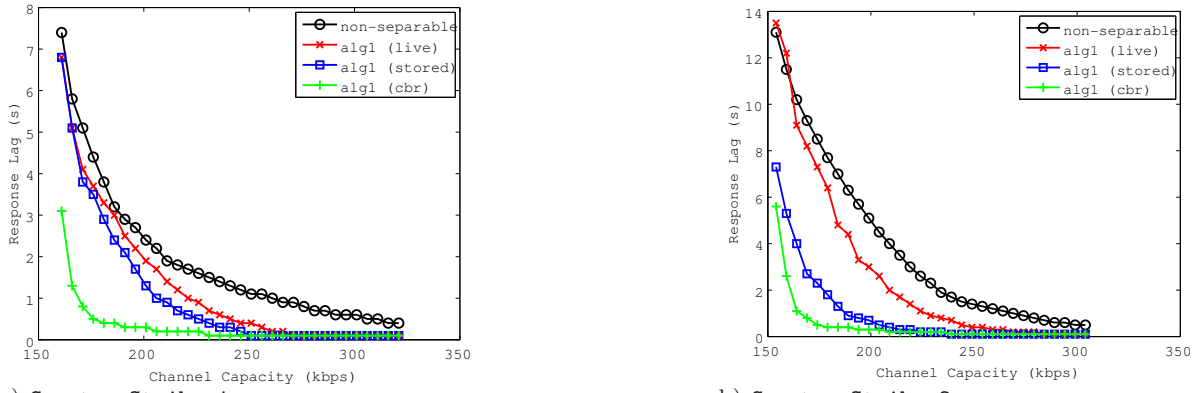
Figure 8. Two Sequences of `Counter-Strike` Exhibit High Variation of Bit-rate as a Function of Time.

Figure 8 shows the bit-rate profiles for two `Counter-Strike` sequences produced by HC. It first shows the bit-rate profile of non-separable video. When separable video is employed, the video is split into fixed and modifiable regions. Bit-rate profile for the modifiable region is also shown. Finally, Figure 8 shows the total bit-rate (sum of two regions) when separable video is employed. The difference between bit-rate profiles for the modifiable region and the sum of two regions represents the fixed region. First, we observe that the bit-rate profiles of the non-separable video and the separable video are very similar. This confirms our earlier observation that the coding cost of using separable video instead of non-separable video is small. Second, we see that all three bit-rate profiles of the same sequence have similar and large variations as a function of time. The largely variable bit-rate (VBR) nature of the encoded video has important consequences in terms of multiplexing, which we will discuss in the following sections.

6.2 Streaming Separable Video: Separate Channel Transmission

In this and subsequent sections, we show experimentally the advantages of separable video over non-separable video under various application scenarios. In all of them, a delivery channel with constant bit-rate (CBR), zero transmission delay and zero transmission loss is assumed. For a given channel capacity, we calculate the *response lag*—the minimum receiver buffer delay γ_{int} required to avoid interactive (sub-)stream playback underflow; this is the metric of evaluation. Plots of response lag vs. capacity for both separable and non-separable videos are drawn for comparison.

We first compare the performance of non-separable video with separable video using Algorithm 1 as discussed in Section 5.2. Recall that Algorithm 1 divides the available channel bandwidth into two bundles and schedules delivery of the interactive and non-interactive sub-streams in their separate bundles with no multiplexing across sub-streams. We assume the receiver's initial buffer delay is no larger than response lag plus five seconds, or $\delta_{init} \leq \gamma_{int} + 5s$. In addition, we assume the non-interactive sub-stream is live-encoded with no video cached at HC; this means that HC cannot read ahead in the non-interactive stream to fill its sub-channel to capacity at



a) `Counter-Strike 1` b) `Counter-Strike 2`

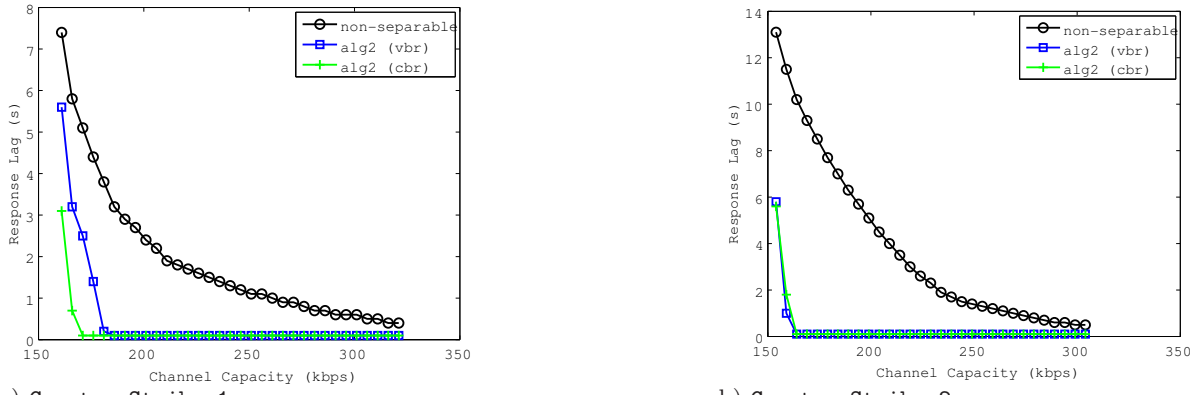
Figure 9. Response Lag vs. Channel Capacity for `Counter-Strike` Sequences for Algorithm 1

all time. Essentially, Algorithm 1 is scheduling two live-encoded sub-streams with different delay requirements. We see in Figure 9 that non-separable video outperformed separable video over most range of channel capacity for both `Counter-Strike` sequences. This shows experimentally that *asymmetric-delay gain*, as described in Section 5.2, can be realized without *capacity-usage gain*.

We now assume that HC can read ahead in the non-interactive sub-stream; i.e., the non-interactive sub-stream is in essence pre-stored at HC. We see in Figure 9 that separable video had further reduced response lag for the two `Counter-Strike` sequences. This shows experimentally that in addition to asymmetric-delay gain, Algorithm 1 can benefit from capacity-usage gain.

Finally, one possible way to further reduce the response lag in is to encode the interactive sub-stream at CBR. As discussed in Section 5.2, this means that traffic associated to this sub-stream is no longer bursty, and a CBR sub-channel of the same bit-rate can support the sub-stream with no receiver buffer underflow. Note that this comes at a cost of variable visual quality over time in the modifiable region contained in the interactive sub-stream. We tested Algorithm 1 using interactive sub-stream encoded at CBR, and we indeed see a noted improvement in the performance of Algorithm 1.

6.3 Streaming Separable Video: Shared Channel Transmission



a) `Counter-Strike 1` b) `Counter-Strike 2`

Figure 10. Response Lag vs. Channel Capacity for `Counter-Strike` Sequences for Algorithm 2.

We next compare the performance of non-separable video with separable video using Algorithm 2 as discussed in Section 5.3. Recall that Algorithm 2 also divides the available channel bandwidth into two bundles, but unused bandwidth for interactive sub-stream can now be used by non-interactive sub-stream. Figure 10 shows the corresponding results for Algorithm 2. Due to additional *stream-multiplexing gain* as described in Section 5.3,

the results are improved over that of Figure 9. In particular, we see that separable video can support interactive streaming with 0.1 seconds of response lag at channel capacity $C = 200kbps$, which is only 20% higher than average source rate of $150kbps$, whereas non-separable video requires C above $300kbps$.

As done previously, we can further reduce the response lag by encoding the interactive sub-stream at CBR. However, for Algorithm 2 we see only minor gain. This is predicted in Section 5.2: the unused bandwidth of the interactive sub-channel due to the VBR nature of interactive sub-stream is already thoroughly exploited by Algorithm 2 via multiplexing of the two sub-streams; forcing the interactive sub-stream to be CBR affords us little additional gain.

7. CONCLUSIONS

In this paper, we have presented a networked video system that supports community streaming with interactive visual overlays. In particular, an Observer View Generator encodes or transcodes live video into a stream containing two independently decodable regions: "fixed" and "modifiable". A downstream Heckler Coordinator then overlays avatars, controlled by a group of hecklers, on top of the modifiable region to sustain a sense of community for the group. We have shown that the separable video approach offers both coding and transport benefits: i) transcoding only the modifiable region of the original stream means re-encoding complexity at the Heckling Coordinator is significantly reduced; in the meantime, ii) the quality of the fixed region is maintained; and iii) multiplexing gain by judicially transporting sub-streams containing fixed and modifiable regions with different delay requirements translates into better interactiveness among hecklers.

8. ACKNOWLEDGMENTS

The authors thank Takashi Sakamoto of HP Labs Japan for software support of the `Counter-Strike` client.

REFERENCES

1. V. Kahmann, J. Brandt, and L. Wolf, "Collaborative streaming and dynamic scenarios," in *Communications of the ACM*, **49, no.11**, November 2006.
2. "Yahoo Avatars." http://avatars.yahoo.com.
3. "Counter-Strike." http://www.counter-strike.net.
4. J. R. et al, "SIP: Session initiation protocol," June 2002. IETF RFC 3261.
5. R. Shacham, H. Schulzrinne, S. Thakolsri, and W. Kellerer, "Ubiquitous device personalization and use: The next generation of IP multimedia communications," in *ACM Transactions on Multimedia Computing, Communications and Applications*, **3, no.2**, May 2007.
6. "sauerbraten." http://sauerbraten.org.
7. "Half-Life TV." http://www.hltv.org.
8. G. Cheung, T. Sakamoto, and W. t. Tan, "Graphics-to-video encoding for 3G mobile game viewer multicast using depth values," in *IEEE International Conference on Image Processing*, (Singapore), October 2004.
9. T. V. Lakhsman, A. Ortega, and A. R. Reibman, "VBR video: Trade-offs and potentials," *Proceedings of the IEEE* **86**, pp. 952–973, May 1998.
10. D. N. C. Tse, R. G. Gallager, and J. N. Tsitsiklis, "Statistical multiplexing of multiple time-scale Markov streams," *IEEE J. on Sel. Areas in Communications* **13**, pp. 1028–1038, Aug. 1995.
11. J. D. Salehi, S.-L. Zhang, J. Kurose, and D. Towsley, "Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing," *IEEE/ACM Transactions on Networking* **6**, pp. 397–410, Aug. 1998.
12. C.-Y. Hsu, A. Ortega, and A. Reibman, "Joint selection of source and channel rate for VBR video transmission under ATM policing constraints," *IEEE J. on Sel. Areas in Communications* **15**, pp. 1016–1028, Aug. 1997.
13. Y.-C. Chang and D. G. Messerschmitt, "Segmentation and compression of video for delay-flow multimedia networks," in *Proc. of ICASSP 1998*, (Seattle, WA), Apr. 1998.
14. Y.-C. Chang and D. G. Messerschmitt, "Improving network video quality with delay cognizant video coding," in *Proc. of ICIP 1998*, **3**, pp. 27–31, (Chicago, IL), Sept. 1998.
15. C. Zhu, "RTP payload format for H.263 video streams," September 1997. IETF RFC 2190.
16. H. Schulzrine, A. Rao, and R. Lanphier, "Real time streaming protcol (RTSP)," April 1998. IETF RFC 2326.
17. "IP multimedia subsystems." http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem.
18. T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," in *IEEE Transactions on Circuits and Systems for Video Technology*, **13, no.7**, pp. 560–576, July 2003.
19. G. Cheung and A. Ortega, "Fast H.264 mode selection using depth information for distributed game viewing," in *Proc. of Visual Communications and Image Processing*, (San Jose, CA), January 2008.
20. A. Ortega, K. Ramchandran, and M. Vetterli, "Optimal trellis-based buffered compression and fast approximations," *IEEE Trans. on Image Proc.* **3**, pp. 26–40, Jan. 1994.