Proxy-Assisted Power-Friendly Streaming to Mobile Devices

Prashant Shenoy and **Peter Radkov**
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003
{shenoy,pradkov}@cs.umass.edu

Abstract

Since multimedia applications are known to be resource-hungry and mobile devices are resource-poor, in this paper, we propose techniques to reduce the energy consumption of streaming media applications running on mobile hosts. Our proposed techniques are proxy-based and involve power-friendly transformations on the requested streams so as to limit the energy required for receiving and decoding this data. Additionally, our proxy employs intelligent network transmission techniques to reduce the energy needs for network reception of streaming data. We implement our techniques into a prototype proxy and client and demonstrate their efficacy via an experimental evaluations. Our results show that our power-friendly transformations are effective over a range of bit rates and stream resolutions, while our intelligent transmission techniques can reduce the potential energy wastage during network reception by 65-98%.

1 Introduction

1.1 Motivation

The dramatic increase in the popularity of the Internet over the past decade coupled with the wide availability of networked mobile devices have fueled the growth of a new generation of mobile Internet applications. Today mobile devices such as cellular phones with wireless Internet access and PDAs with wireless LAN interfaces routinely access information on the Internet. Although much of this information is textual in nature, a new generation of mobile devices capable of handling audio, video, and graphic-rich content (e.g., iMode phones, PocketPC PDAs) is becoming popular. Studies have shown that over the next decade networked mobile hosts will vastly outnumber traditional wired hosts such as PCs and workstations. Further, mobile devices significantly differ from traditional wired hosts in their capabilities and characteristics—such devices have intermittent connectivity and are resource-poor. The resource poor nature arises since power (battery) capacities, computational resources, network bandwidth and display resolutions on such devices is significantly smaller than traditional desktops (see Table 1). Due to these key differences, multimedia applications designed for wired hosts are not directly suitable for mobile environments, necessitating fundamental changes in the way mobile multimedia applications are designed and used.

Multimedia applications are known to resource-hungry, while mobile devices are resource-poor, resulting in a fundamental disconnect. In this paper, we attempt to bridge this gap by devising techniques to reduce the energy (and hence, resource) consumption of streaming media applications running on mobile hosts. Our approach to address this challenge is to employ proxies that act as intermediaries between mobile hosts and streaming servers; our proxies use *power-friendly* streaming techniques that are governed by client-specified characteristics and the capabilities of the mobile host. In particular, proxies (i) intelligently transform incoming (or cached) streams and (ii) use intelligent network transmission techniques to reduce the energy consumed at the mobile host during decoding and network reception. By reducing energy needs at the CPU and the network interface card, the overall energy needs of the mobile multimedia application are significantly reduced.

Type	Processor	Display	NIC	Power Consumption	
				Device	NIC
Laptop	Mobile Pentium	1024x768 LCD	11Mb/s 802.11b	50-70 W	50mW (doze), 900-1400mW (recv/xmit)
PocketPC PDA	200 MHz StrongARM	240x320 LCD	11 Mb/s 802.11b	900 mW	same as above
Palm PDA	33 MHz Dragonball	240x320 LCD	11Mb/s 802.11b	900 mW	same as above
Mobile phone	–	Text display	144 kbps GPRS	2.3 W	included in device
Desktop	Pentium 4	1600x1200 CRT/LCD	1 Gb/s ethernet	200-300 W	included in device

Table 1: Characteristics of Typical Mobile Devices and Traditional PCs

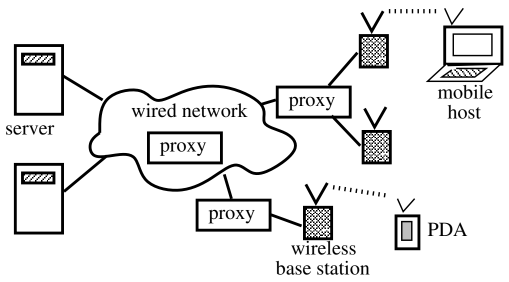


Figure 1: System model for mobile clients accessing streaming media data via proxies.

1.2 Research Contributions

In this paper, we propose two specific techniques to reduce the energy needs of mobile streaming media applications. Our first technique performs power-friendly transformations on client-requested streams to reduce their energy needs. To perform such transformations, we first devise models to estimate the energy needs of a particular stream and use our models to determine the transformations that are necessary to limit the energy needs of the stream to client-specified values. We then propose techniques to transmit the transformed stream to the mobile client in an intelligent fashion. Our transmission techniques enable a client to reduce the energy wastage at its network interface while waiting for data packets. We implement our techniques into a prototype proxy and a mobile client. Our experimental evaluation of the prototype shows that: (i) our power-friendly transformations are effective over a range of bit rates, resolutions and encoding parameters for MPEG-encoded streams, and (ii) our network transmission and reception techniques can reduce the potential energy wastage at the network interface by 65-98%.

The rest of this paper is structured as follows. Section 2 formulates the problem addressed in this paper and specifies our system model. We present our power-friendly transformation techniques in Section 3. Section 4 discusses our power-friendly network transmission and reception schemes. Implementation issues are discussed in Section 5. We present our experimental results in Section 6 and discuss related work in Section 7. Finally, Section 8 presents our conclusions.

2 Problem Definition and System Model

The system model assumed in our research consists of an environment with four types of entities—servers, proxies base-stations, and end-clients (see Figure 1). End-clients are assumed to be mobile devices such as PDAs, laptops or mobile phones with wireless network interfaces. Each mobile host communicates with other entities via a base-station in its geographical proximity; the link between the mobile client and the base station can either be a wireless LAN such as 802.11b or a wireless WAN such as a GSM data network. Each base-station has a proxy associated with it; each proxy is responsible for servicing clients connected to one or more base stations in its proximity.

Given such an environment, applications running on the mobile host send requests for streaming media objects to a proxy. The proxy services a request locally if the requested object is cached or fetches the object from the server in the

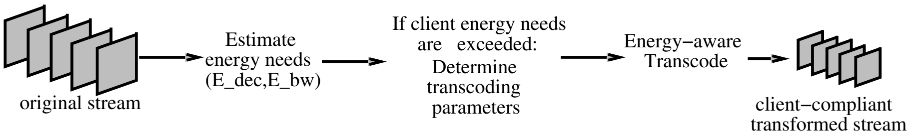


Figure 2: An overview of the the power-friendly transformation process

event of a cache miss. In either case, the object may need to be transformed prior to transmission so as to match the capabilities of the end-client. To enable such transformations, we assume that the client specifies three parameters: the available energy budget for decoding, the available energy budget for network reception and the maximum spatial resolution that it can support (which depends on its screen size). The energy budget for decoding translates to a limit on the amount of CPU time the client is willing to spend on decoding the stream. The energy budget for network reception translates to a limit on the rate at which the client is willing to receive data on its network interface (i.e., a limit on the bandwidth of the stream). In addition, we also assume that the client specifies the type of the CPU and the network interface on the device, so that the proxy can interpret the specified energy budgets relative to these hardware characteristics.

Given these client parameters, the specific problem we wish to address is as follows: how should the proxy transform the original stream such that the resulting stream satisfies the specified constraints on the decode times, the network bandwidth and the spatial resolution? Performing such a transformation helps limit the energy expended in receiving and decoding the stream to the specified values. In addition to performing such power-friendly transformations, the proxy can employ additional techniques to further reduce the energy usage at the client. For instance, most wireless network interfaces have multiple power modes—an active mode, where the card is actively waiting to send or receive data, and a passive (doze) mode, where the card switches to a low power mode; the power consumption in the passive mode is typically significantly smaller than the active mode (see Table 1). This observation provides opportunities for additional energy savings—if the proxy employs an intelligent network transmission technique for sending data, the client can determine the arrival times of individual data packet and selectively switch its interface to the active mode at those instants (and use the passive mode for the remaining time) [4]. The more accurate these estimates, the greater the savings (since less time is wasted in the active mode waiting for data packets). Hence, a second problem that we wish to address is: given the transformed stream, how should the proxy transmit it so that the energy expended in receiving it at the client is minimized (i.e., what transmission technique enables the client to best extract energy savings by selectively powering-up its network interface?)

In what follows, we first present techniques for performing power-friendly transformations and then discuss intelligent network transmission techniques.

3 Power-friendly Transformations to Limit Energy Consumption

Consider a mobile client that requests a video stream from a proxy and specifies constraints on the energy available for decoding $\hat{E}_{dec}$, the energy available for network reception $\hat{E}_{bw}$ and the maximum spatial resolution $\hat{\mathcal{R}} = (\hat{\mathcal{R}}_x, \hat{\mathcal{R}}_y)$. Upon receiving this request, the proxy must first fetch the stream from the its disk cache (in the event of a hit) or from the server (in the event of a miss). The proxy then determines if the original stream satisfies the specified constraints and if not, determines how the original stream should be transformed to produce a stream that meets the client needs. Figure 2 illustrates this process. To precisely define this transformation, let $f_1, f_2, f_3, \ldots f_n$ denote the n frames in the original stream. Let $S(f_i)$ denote the size of frame i and let $D(f_i)$ denote the decode time for the frame at the end-client. Further, let p denote the playback rate of the stream in frames/sec. We assume that the energy needs for decoding or network reception are computed over an interval $\mathcal{I}$; in this paper, we estimate energy needs of the video stream over each 1 second interval (i.e., $\mathcal{I} = 1$). Then, the energy needs of the original stream over an interval

$[t, t+1)$ are defined as follows:

$$E_{dec}(t, t+1) = c_1 \cdot \sum_{i=t \cdot p}^{(t+1) \cdot p - 1} D(f_i) \qquad t = 0, 1, 2, ... \tag{1}$$

$$E_{bw}(t, t+1) = c_2 \cdot \sum_{i=t \cdot p}^{(t+1) \cdot p - 1} S(f_i) \qquad t = 0, 1, 2, ... \tag{2}$$

That is, the energy consumed during decoding over an interval $\mathcal{I}$ is proportional to the sum of the decode times of individual frames and the energy consumed during network reception is proportional to the number of bits received (which is same as the sum of frame sizes). Note that c_1 and c_2 are constants that allow a conversion from decode times and bytes, respectively, to energy units (alternatively, we can express energy usage directly in units of time and bytes, in which case $c_1 = c_2 = 1$).

Having determined the energy requirements (E_{dec} and E_{bw}) and the spatial resolution of the original stream, the proxy must transform the stream *if any of these parameters exceed the client specified constraints*. The transformation is performed over frames in each 1 second interval where the client-specified constraints are exceeded and yields a new stream $\hat{f}_1, \hat{f}_2, \cdots \hat{f}_n$ such that

$$c_1 \cdot \sum_{i=t \cdot p}^{(t+1) \cdot p} D(\hat{f}_i) \quad \leq \quad \hat{E}_{dec} \quad \forall t, t = 0, 1, 2, ... \tag{3}$$

$$c_2 \cdot \sum_{i=t \cdot p}^{(t+1) \cdot p} S(\hat{f}_i) \quad \leq \quad \hat{E}_{bw} \quad \forall t, t = 0, 1, 2, .. \tag{4}$$

Further, the transformation should ensure that the spatial resolution of the resulting stream is no greater than the specified limits of $\hat{\mathcal{R}} = (\hat{\mathcal{R}}_x, \hat{\mathcal{R}}_y)$.

While some of the parameters that govern the above transformation are easy to determine, others are not. For instance, given a video stream, it is typically simple to determine its spatial resolution $\mathcal{R}$ and individual frame sizes $S(f_i)$ (the former is specified in the stream headers, while the latter can be computed by determining the frame boundaries and then the individual frame sizes). In contrast, determining the decode time of a frame $D(f_i)$ without actually decoding the stream is non-trivial. Similarly, if the stream needs to transformed, determining the characteristics of the transformed stream such that the above inequalities are satisfied is non-trivial (i.e., determining the exact transformation that needs to be performed is challenging). In the rest of this section, we first describe how to estimate frame decode times without actually decoding the stream and then discuss how to determine the parameters of the transformation such that client constraints are satisfied.

3.1 Empirical Model for Determining Frame Decode Times

As seen in Equations 1 and 3, determining the energy needs for decoding requires an estimate of the decode times of individual frames. A simple approach to determine decode times is to actually decode the stream at the proxy. However, such an approach is compute-intensive and can be a performance bottleneck at the proxy. An alternate approach is to estimate the decode times of frames using their attributes (frame size, frame type, spatial resolution, etc). Determining such frame attributes is computationally less expensive than a full decode since it only requires a parsing of the media stream (e.g., to determine frame boundaries). The challenge though is to devise a model that can accurate predict the decode times using these frame characteristics. In what follows, we use empirical techniques to devise such a model.

To design our model, we encoded a large number of MPEG-1 video streams (about 90 streams) with spatial resolutions ranging from 240x160 to 720x480 and bit rates from 192 Kbps to 7 Mbps. The contents of these streams

Resolution	Number of Streams	Frame rate	Length (min)	Mean Bit rate
240 x 160	16	30 fps	1-2 min	197 Kbps - 1.1 Mbps
320 x 200	19	30 fps	1-2 min	192 Kbps - 1.15 Mbps
320 x 240	6	24-30 fps	3m18s - 5m23s	1.58 Mbps - 2.44 Mbps
352 x 240	21	30 fps	1-2 min	193 Kbps - 1.15 Mbps
480 x 480	15	30 fps	1-2 min	229 Kbps - 1.4 Mbps
640 x 400	7	24-30 fps	1 min	604 kpbs - 1.6 Mbps
640 x 480	1	30 fps	2 min	6.6 Mbps
720 x 480	6	30 fps	1-2 min	898 Kbps - 7 Mbps
Other assorted	8	24-30 fps	1m44s - 2m10s	194 kbps - 3.4 Mbps

Table 2: Characteristics of MPEG-1 streams used in our study

varied from movies, television sitcoms, newscasts, sports events and chat shows. The characteristics of these streams are summarized in Table 2. We instrumented the publicly-available Berkeley MPEG player [14] to record the CPU time spent in decoding each frame (the time to display the decoded frame on the screen was not taken into account). Using our instrumented player, we decoded each stream on a lightly loaded machine and recorded a trace of frame decode times. Our recorded traces showed the following behavior:

- *Observation 1:* The decode time of a frame was found to directly proportional to the compressed frame size. Not surprisingly, the larger the frame, the larger is the amount of CPU time necessary to decode it.

$$D(f_i) \propto S(f_i) \tag{5}$$

- *Observation 2:* The decode time was also found to be directly proportional to the square root of the spatial resolution—the larger the spatial resolution, the larger is the number of pixels that need to be decoded, and the larger is the decode time.

$$D(f_i) \propto \sqrt{\mathcal{R}_x \times \mathcal{R}_y} \tag{6}$$

Based on our first empirical observation (Equation 5), a simple model to predict frame decode times might be:

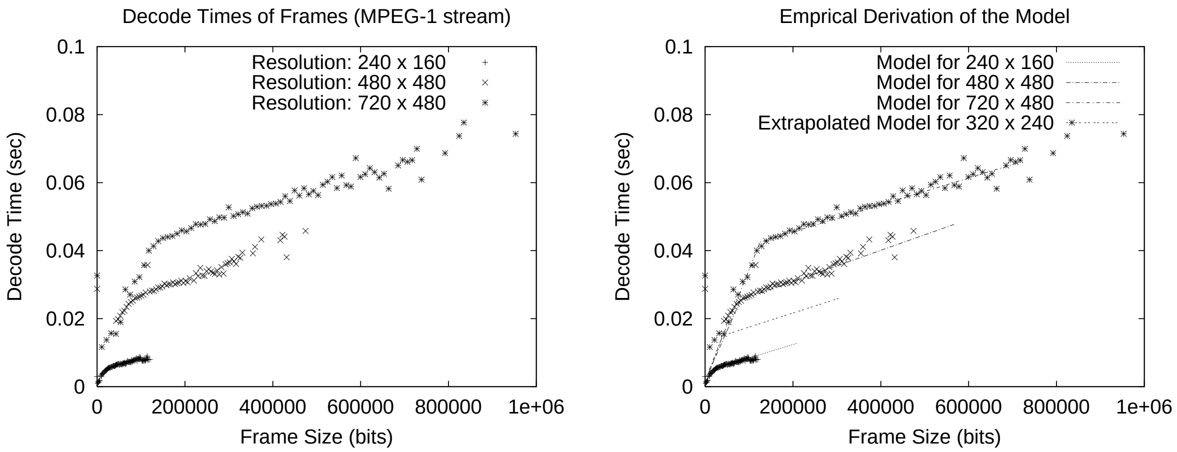
$$D(f_i) = k \cdot S(f_i) \tag{7}$$

where k is a constant.

Figure 3(a) plots the mean decode times for various frames contained in streams with three different spatial resolutions. As can be seen, the decode times are larger for larger frames and larger spatial resolutions, in line with the above observations. However, for a given spatial resolution, we see that the decode times increase sub-linearly with frame size. Consequently, a simple linear relationship between frame size and decode times as in Equation 7 is not a good predictor of decode times; a better approach is to use a *piece-wise linear function* to relate frame sizes to their decode times. While a sophisticated model may use multiple piece-wise linear curves, we found two piece-wise linear functions to be adequate for our purpose. That is,

$$D(f_i) = \begin{cases} k_1 \cdot S(f_i) & \text{if frame size } S(f_i) \leq T \\ k_2 \cdot S(f_i) + k_3 & \text{if frame size } S(f_i) > T \end{cases} \tag{8}$$

where k_1 and k_2 are slopes of the two linear functions and k_3 is the Y-intercept of the latter function. Thus, depending of the frame size, our refined model uses two different estimators to predict the decode times—the first equation is used to determine decode times for *smaller* frames (i.e., frames smaller than threshold T) and the second equation is used to determine decode times for *larger* frames. The values of k_1, k_2 and k_3 are determined empirically for a particular spatial resolution using curve-fitting methods that minimize the mean square error. Figure 3(b) depicts the resulting piece-wise linear predictors for three different spatial resolutions. As can be seen, a piece-wise linear function is a good approximation of the actual decode times.



(a) Frame decode times

(b) Model to predict decode times

Figure 3: Deriving the empirical model.

Since our empirical methods yield predictors for only a small number of spatial resolutions, we need to devise techniques to predict decode times for spatial resolutions that lack a pre-computed empirical model (determining the values of k_1, k_2 and k_3 for all possible spatial resolutions is clearly not feasible). We use our second empirical observation (Equation 6) to extrapolate from a pre-computed model and predict decode times for other spatial resolutions. Since decode times are proportional to the square root of the spatial resolution, we can use the spatial resolution as a scaling factor to devise a predictor for a different resolution. Thus, given values of k_1, k_2, k_3 for a spatial resolution $(\mathcal{R}'_x, \mathcal{R}'_y)$, the model for spatial resolution $(\mathcal{R}_x, \mathcal{R}_y)$ is

$$D(f_i) = \begin{cases} k_1 \cdot S(f_i) \cdot \sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}} & \text{if frame size } S(f_i) \leq T\sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}} \\ (k_2 \cdot S(f_i) + k3) \cdot \sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}} & \text{if frame size } S(f_i) > T\sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}} \end{cases} \tag{9}$$

Figure 3(b) shows a piece-wise linear predictor for a resolution of 320×240 constructed by extrapolating from the (known) predictor for 720×480.

By substituting this model (Equation 9) into Equation 1, we obtain our final model to compute the energy spent in decoding the stream over each 1 second interval:

$$E_{dec}(t, t+1) = c_1 \cdot \sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}} \cdot \sum_{i=t\cdot p}^{(t+1)\cdot p-1} \alpha \cdot S(f_i) + \beta \tag{10}$$

where $\alpha = k_1$ and $\beta = 0$ if the frame size $S(f_i) \leq T\sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}}$ and $\alpha = k_2$ and $\beta = k_3$ if the frame size $S(f_i) > T\sqrt{\frac{\mathcal{R}_x \times \mathcal{R}_y}{\mathcal{R}'_x \times \mathcal{R}'_y}}$. $(\mathcal{R}_x, \mathcal{R}_y)$ is the resolution of the stream and $(\mathcal{R}'_x, \mathcal{R}'_y)$ is the nearest resolution for which an empirical model is available.

3.2 Determining the Transcoding Parameters

Consider a MPEG stream that does not satisfy the client-specified parameters. The proxy must then transform (transcode) the stream such that constraints on energy consumption ($\hat{E}_{dec}, \hat{E}_{bw}$) and the spatial resolution $\hat{\mathcal{R}}$ are satisfied. We assume that the proxy employs a real-time transcoder for this transformation. Techniques for video transcoding have been well-investigated [1, 3, 12, 18] and a number of commercial and public-domain video transcoders are available for various compression formats. Consequently, rather than developing a new transcoder, we assume the availability of a flexible video transcoder and focus on determining the transcoding parameters to produce the transformed stream. Specifically, we use the publicly available TMPGenc transcoder at our proxy [17]. TMPGenc is a flexible MPEG to MPEG transcoder that takes as input a number of parameters such as the target bit rate,the encoding pattern, the target spatial resolution, etc., and produces an output MPEG stream that conforms to these parameters.

Assuming such a transcoder, our objective is to determining transcoding parameters to produce the transformed stream. The transcoding process essentially degrades the *stream quality* such that the resulting stream satisfies client needs. There are two independent dimensions along which the stream quality may be degraded—the spatial resolution (i.e., picture size) and the chroma resolution (i.e., picture quality). Both result in a reduction in the bit rate (and hence, bandwidth and decoding requirements). In general, different combinations of the picture size and picture quality may satisfy the client constraints. For instance, given a certain picture size and picture quality, the proxy may only degrade the picture size (and keep the picture quality fixed) until the decoding and bandwidth constraints are met. An alternate approach is to degrade only the picture quality (and keep the picture size fixed) until the constraints are satisfied. A third approach is to degrade simultaneously along both dimensions. Since degrading along only one dimension may yield unintended results (e.g., a large picture with very poor quality or a thumbnail-sized video with excellent picture quality), a better approach is to successively degrade by small amounts along both dimensions until client needs are met. Consequently, our proxy searches various combinations of spatial resolutions and chroma resolutions (by degrading along each dimension in each step) until it finds an appropriate combination of the two. This ensures that the proxy produces the best quality stream that still meets all client constraints. The search process uses the models derived in the previous section to determine an appropriate picture size and quality. Essentially, for frames in each 1 second interval, the algorithm uses the models to examine the energy requirements for various combinations of spatial and chroma resolutions and picks one that satisfies the client energy needs. The algorithm then invokes the transcoder with these parameters to produce the transformed stream for that interval.

While the basic algorithm outlined above will produce a stream that meets client needs, it suffers from certain limitations. Observe that the algorithm determines transcoding parameters for each 1 sec interval *independently of the remaining stream.* This can yield a transformed stream where the spatial resolution differs across each 1 second interval, resulting in a playback where the picture size fluctuates every so often. Frequent changes in picture sizes can be unsettling to the end-user. Since human perception is more tolerant to changes in picture quality than the picture size, we can bias the basic algorithm to prefer a degradation in picture quality over that in the picture size. Two possible enhancements that introduce such a bias are possible depending on whether the stream is cached at the proxy or whether it is being fetched in real-time from the server. In the former scenario, the proxy can quickly examine the cached stream and determine the amounts by which the picture size and picture quality needs to be degraded for each 1 second interval. It then picks the minimum spatial resolution over all intervals and uses this for the entire video. This ensures that the spatial resolution remains fixed over the duration of the video and only the picture quality varies across each interval.

While such a technique is feasible for cached stream, a different technique is necessary when the stream is being fetched in real-time from the server (since it is not possible to pre-process the entire stream before transcoding it). In such an event, we can bias the algorithm by degrading the picture quality multiple times before considering a degradation in the picture size (i.e., the algorithm degrades the picture quality i times, $i > 1$, before considering a degradation in picture size). Further, the algorithm can remember the spatial resolution used for the previous 1 second interval and use it for transcoding in subsequent intervals, thereby ensuring that the resolution does not vary across each such interval. Together, these enhancements minimize the variations in the spatial resolution by biasing the algorithm towards picture quality degradations.

4 Power-friendly Network Transmission and Reception

In the previous section, we discussed techniques for transforming a stream to meet client-specified constraints. The transformed stream is then transmitted by the proxy over a wireless network. In this section, we examine intelligent

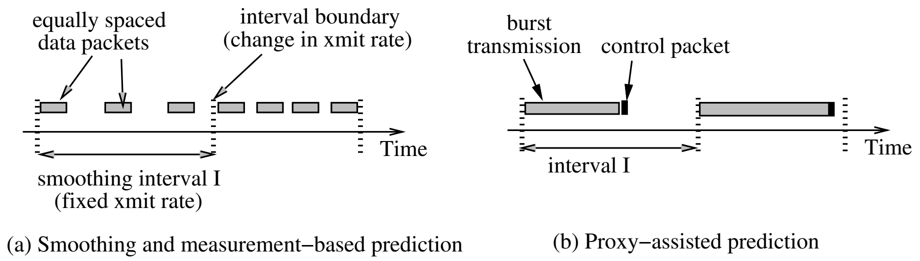


Figure 4: Power-friendly network transmission techniques.

network transmission techniques that allow a client to minimize the energy expended in receiving the transformed stream. Our techniques are motivated by observation that the power consumption of a wireless interface in the passive mode is significantly smaller than the active mode, and consequently, switching the interface to the passive mode in idle periods can result in significant energy savings [4]. The effectiveness of such an approach, however, depends on the accuracy with which a client can predict packet arrival times. The better the prediction, the greater are the savings. In what follows, we present two techniques for intelligent network transmission and reception that extract such energy savings. We compare these two approaches with other recent work in the area [4, 5] in Section 7.

4.1 Proxy-based Smoothing and Measurement-based Prediction

Video streams that employ compression formats such as MPEG can have a variable bit rate nature. Variations in the stream bit rate can result in corresponding variations in inter-arrival times of packets at a client, making the task of predicting packet arrival times more complex. One possible approach for handling the VBR nature of video streams is to employ smoothing techniques at the proxy. A smoothing algorithm converts a VBR stream into a piece-wise constant bit rate (CBR) stream and allows the proxy to transmit each CBR segment at a constant bit rate. Consequently, smoothing can alleviate the problems in predicting packet arrival times of a VBR stream, since data is transmitted at a fixed rate within each piece-wise CBR segment. A number of smoothing algorithms have been proposed recently [7, 11, 16] and any such algorithm may be employed by the proxy. For the purpose of this paper, we assume a simple smoothing algorithm that smooths the stream data in each interval $\mathcal{I}$ and transmits it at a fixed rate within that interval. That is, within each smoothing interval $\mathcal{I}$, the technique sets the transmission rate to $\sum_{i \in \mathcal{I}} S(f_i)/\mathcal{I}$. Figure 4(a) illustrates this technique.

Assuming such a smoothing technique, the client needs to estimate the arrival time of each packet and switch on its interface before the first bit of each packet arrives. One technique to predict the arrival time is to maintain a history of packet inter-arrival times and use the mean inter-arrival time of the previous i packets to predict the arrival time on the next packet. Thus, if t denotes the time at which the previous packet was completely received and A denotes the mean inter-arrival time of the previous i packets, then the predicted arrival of the next packet is $t + \gamma \cdot A$, where γ is a constant. The value of γ can chosen between 0 and 1 depending on how conservative or aggressive the client wants to be. If $\gamma = 1$, the card is powered on at precisely the predicted arrival time. Smaller values of γ allow the client to power-up the card *before* the predicted arrival time to account for variations in the link delay, proxy load, etc. The more the anticipated variability in packet-arrival times, the smaller should the value of γ be to prevent packet losses (a packet is lost if the interface is not in the active mode when the first bit of the packet arrives).

4.2 Proxy-assisted Prediction

The approach outlined in the previous section required the client to predict packet arrival times based on past history. While such an approach works well when both the number and magnitude of bit rate changes are small, it can result in packet losses whenever the stream exhibits increased variability in the bit rate (note that, this is true even in the presence of smoothing where bit-rate changes occur at the boundaries of the piece-wise CBR segments). Typically, a client can address this problem by choosing conservative estimates of packet arrival times (i.e., small values of γ), which in turn reduce energy savings (since it increases the time spent by the network interface in the active mode). An orthogonal limitation of employing smoothing techniques at the proxy is that it increases the playback startup delay (since additional data must be buffered at the client to prevent buffer underflows in the presence of smoothing).

Both of these problems can be addressing by eliminating smoothing at the proxy and instead having the proxy provide explicit control information to the mobile client—the client can then utilize this control information to accurately estimate packet arrival times (instead of resorting to past observations for making such predictions). Consequently, in our second approach, the proxy does not smooth the video stream; instead it sends out the entire set of frames within each interval $\mathcal{I}$ in a single burst. At the end of each burst, the proxy sends a control packet that indicates the transmission time of the next set of frames. The client uses this control information to switch its interface to the active mode at the specified instant (and uses the passive mode in the interim). Specifically, if t denotes the instant at which the proxy will begin transmitting the next set of frames and d denotes the link delay, then the client switches its interface to the active mode at $t + \gamma \cdot d$, where the parameter γ is used to used to account for potential variations in the link delay. Like in the previous scheme, γ can be chosen between 0 and 1 depending on the accuracy with which the client can estimate the link delay. Fig 4(b) illustrates this technique. Note that, due to the availability of explicit transmission times, the inaccuracy inherent in using past history to predict future packet arrivals is eliminated. This in turn enables the client to aggressively extract energy savings at the network interface,

As a final caveat, we note that hybrid approaches are possible, where a combination of proxy-based smoothing and providing explicit control information to the client may be used. In such an approach, the proxy transmits the stream as piece-wise CBR segments and transmits control information that explicitly specifies the transmission rate of each CBR segment prior to its transmission. The client can use the transmission rate to accurately estimate the packet arrival times within each segment. Further, previous history may be used to account for second-order variations such as those in link delay. Such a hybrid approach results in less bursty network transmissions than the above technique at the potential expense of increased client buffers (and delay).

5 Implementation Issues

We have implemented a prototype proxy and a mobile client that incorporate the techniques described in the previous sections. Our proxy runs on the Windows platform and has two key components: a transformation module and a network transmission module. Upon receiving a client request, the proxy fetches the requested stream from the server if it is not already cached; we use UDP to transmit the stream from the server to the proxy (in the future, we plan to support protocols such as RTP and RTSP at the proxy; a simple UDP scheme suffices for our present work). The proxy then examines the stream to see if it complies with client-specified constraints. If not, the transformation module determines the transcoding parameters for each 1 sec of the video and invokes the transcoder to transform the stream. As indicated earlier, our proxy uses the public TMPGEnc transcoder to transform the stream. From our experiments, we observed that TMPGEnc, while very flexible, can not transcode high bit-rate streams in real-time even on fast Pentium processors (low bit rate streams can, however, be transcoded in real-time). Consequently, in our current proxy implementation, we first transcode the entire stream and then transmit it to the end-client (we are also examining other transcoders to see if they provide more efficient real-time transcoding support) . The output of the transformation module at the proxy is fed to the network transmission module. Both the smoothing technique and the proxy-assisted prediction are supported by the network transmission module.

Our client is a Sony PictureBook Laptop with a 633 MHz Transmeta Crusoe processor running Windows 2000. The laptop has 256MB RAM and is equipped with a 11 Mb/s Lucent Orinoco wireless card. The network reception module at the client implements both measurement-based prediction and the proxy-assisted prediction. In the former approach, the client records packet arrival times and uses the past measurements to predict the arrival time of the next packet. In the latter approach, control information provided by the proxy is used to compute packet arrival times. In both cases, the parameter γ is used to control the aggressiveness of the estimates; γ is currently chosen by the user. In both schemes, the network reception module emulates the toggling of the network interface between active and passive modes based on packet arrival times. Since we did not have access to the source code for the Windows driver, we could not actually toggle the card between the two modes and our prototype currently emulates these functions; we plan on implementing future versions of our client in Linux, which will allow us to modify the driver to instantiate this functionality. The data received by the network reception module is fed to a modified (and instrumented) Berkeley MPEG decoder. Our decoder can record various stream statistics (packet loss, frame sizes, bit rate, etc) as well as frame decode times.

6 Experimental Evaluation

In this section, we experimentally evaluate the effectiveness of our techniques using our prototype. We first present an evaluation of our techniques for power-friendly transformations and then demonstrate the benefits of our intelligent transmission techniques.

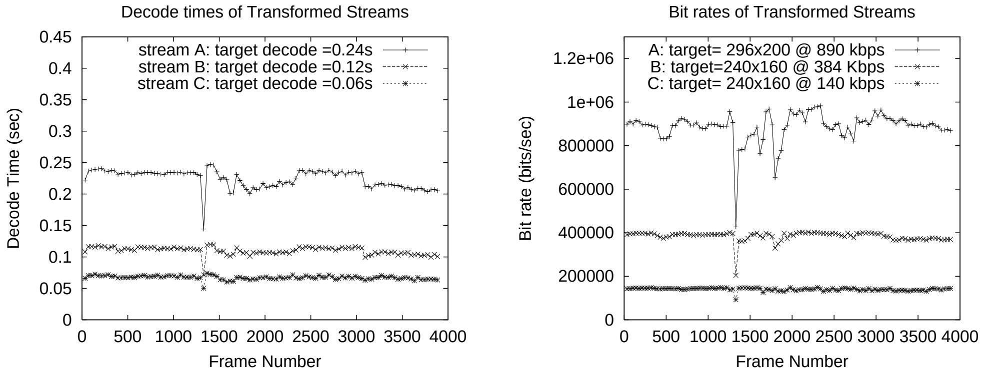
6.1 Efficacy of the Power-friendly Transformation Techniques

To demonstrate the efficacy of our techniques, we conducted an experiment where the client requested various MPEG-1 streams with different constraints on the decode times, bandwidth and spatial resolutions. The streams generated by the proxy were then examined at the client to determine if they adhered to the specified constraints. Specifically, we decoded each stream at the client and measured the frame decode times. We also computed the bandwidth of the transformed streams over each 1 second interval. Due to space constraints, we present results for only a few representative streams from the ones we considered in this experiment. Specifically, we present results for two MPEG-1 streams encoded at resolutions 720x480 and 640x480 and bit rates of 7.1 Mbps and 6.6 Mpbs, respectively.

We derived five different streams using these two streams. Specifically, using the 720x480 stream, we derived three streams: (i) stream A with decoding constraint $\hat{E}_{dec}$=0.24s, bandwidth constraint $\hat{E}_{bw}$=890 Kbps and resolution $\hat{\mathcal{R}}$=296x200, (ii) stream B with $\hat{E}_{dec}$=0.12s, $\hat{E}_{bw}$=384 Kbps, and $\hat{\mathcal{R}}$=240x160, and (iii) stream C with $\hat{E}_{dec}$=0.06s, $\hat{E}_{bw}$=140 Kbps and $\hat{\mathcal{R}}$=240x160. Using the 640x480 stream, we derived two streams: (i) stream D with $\hat{E}_{dec}$=0.9s, $\hat{E}_{bw}$=3 Mbps and $\hat{\mathcal{R}}$=544x408 and (ii) stream E with $\hat{E}_{dec}$=0.6s, $\hat{E}_{bw}$=2 Mbps and $\hat{\mathcal{R}}$=520x344. As can be seen, the derived streams span a range of bit rates, decode requirements and spatial resolutions. Figure 5 and 6 depict the decode times and bit rates of these five streams as measured at the client. Figure 7 summarize these results by plotting the mean bit rate and decode times of the transformed streams along with their 95% confidence intervals. As can be seen, the transformed streams closely match the specified client constraints over a range of bit rates, decode times and resolutions (the small errors that are seen are primarily due to the inaccuracies of our models in predicting decode times). Thus, our experiment demonstrates the feasibility and effectiveness of power-friendly transformation techniques in reducing the energy needs at the mobile host.

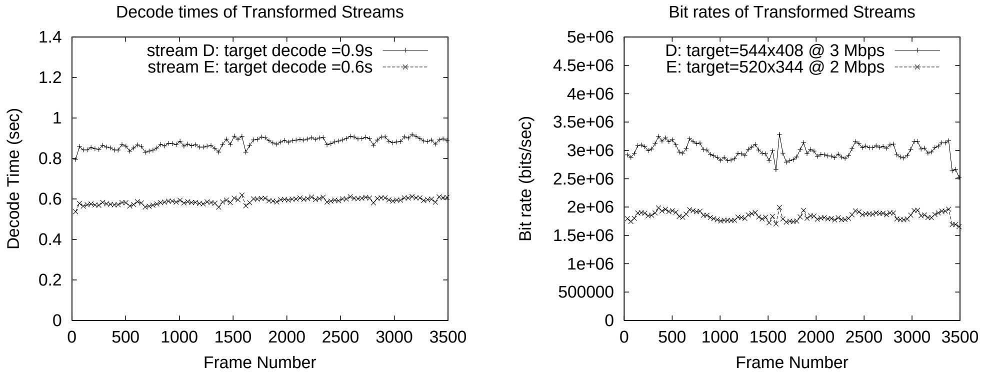
6.2 Effectiveness of our Intelligent Network Transmission Techniques

To examine the effectiveness of our intelligent network transmission schemes, we conducted an experiment where we produced VBR MPEG streams with bit rates ranging from 190 Kbps to 2 Mbps (using the transformation module) and transmitted these streams to the client using our smoothing and bursty transmissions techniques. For each stream, we varied the parameter γ from 0.7 to 1 at the client to examine the efficacy using using different levels of aggressiveness in estimating packet arrival times. In each case, we measured two metrics: the percentage idle uptime and the percentage packet loss. The percentage idle uptime is defined to the fraction of the time the network interface is idle (not receiving any data) and yet is in the active mode. Thus, this metric measures the energy wasted in waiting for data. The ideal energy saving scheme will have a percentage idle uptime of 0, since it will always switch the interface to the passive mode when no data is being received. The packet loss percentage measures the packets lost due to inaccuracies in the packet arrival estimates—a packet is lost if the interface is not active when it arrives. Clearly, a good network reception technique should minimize packet losses while attempting to extract energy savings.



(a) Decode times (b) Bit rates

Figure 5: Decode times and bit rates of the transformed streams derived from the 720x480 7.1 Mbps stream.



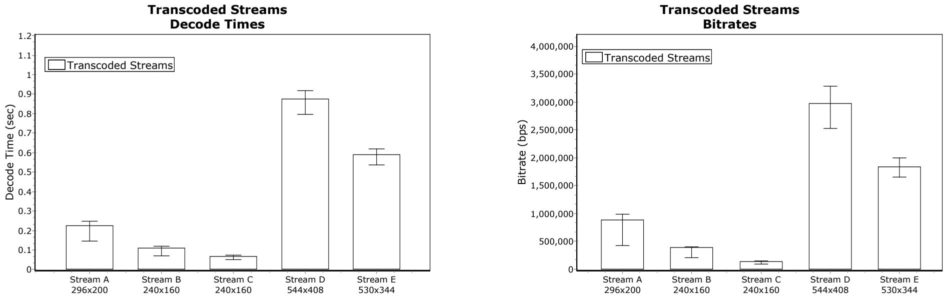
(a) Decode times (b) Bit rates

Figure 6: Decode times and bit rates of the transformed streams derived from the 640x480 6.6 Mbps stream.

Figure 8(a) and (b) depict the idle uptime and packet loss for the two approaches. Although we experimented with a large number of streams, due to space constraints, we only present results for two representative streams: a 644 Kbps stream and a 1.57 Mbps stream. We note that while both streams are VBR streams, the latter stream exhibited more variability in the bit rate than the former. As shown in the figures, more aggressive estimates yields greater energy savings but at the potential expense of larger packet loss. Further, we find that the proxy-assisted prediction scheme yields higher energy savings at negligible (or very small) loss over a range of γ. Specifically, we observe the following behavior:

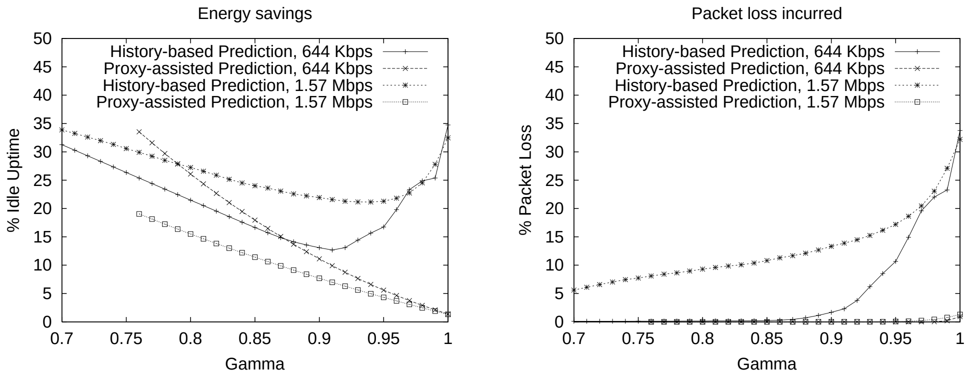
- The measurement-based approach yields idle uptimes ranging from 15-35% depending on the value of γ. In contrast, the proxy-assisted prediction approach yields idle uptimes of only 2-20%, thereby yielding better energy savings. In general, more aggressive estimates of packet arrival times (larger γ) yield larger energy savings (subject to the caveat noted below). We note that a naive reception approach that is not energy-aware has an idle uptime of 100% (since the interface is always active waiting for network packets). Compared to such an approach, our schemes yield an 65-98% increase in energy savings (we also note that, unfortunately, many interface cards ship with power management features turned off by default, resulting in an "always-on" interface and wastage of energy).

- The idle uptime of measurement-based approach shows an interesting behavior. We find that the idle uptime



(a) Variation in decode times (b) Variation in bit rates

Figure 7: Summary of bit rates requirements and decode times of transcoded streams.



(a) % Idle Uptime (b) % Packet Loss

Figure 8: Energy savings and packet loss incurred by our measurement-based and proxy-assisted prediction approaches.

decreases initially with increasing γ and then increases again with increasing γ (Figure 8(a)). This is because very aggressive estimates result in increasing packet loss, which causes the measurement-based approach to become more conservative in its estimates so as to to reduce the loss. Conservative estimates in turn increase the idle uptime. This adaptive behavior is the result of recording the arrival time of a lost packet as $t - \Delta$, where t is the time when the interface was switched on and a packet loss was detected and Δ is a constant. Since arrival times of lost packets are conservatively estimated, future packet arrival estimates also become conservative, resulting in increased idle uptime (and a U-shaped curve for the idle uptime).

- Figure 8(b) shows that the proxy-assisted prediction yields lower losses than the measurement-based approach. We also see the the increased variability of the 1.57 Mbps stream increases the inaccuracy of the estimates in the measurement-based approach, thereby increasing loss. In contrast, proxy-assisted prediction yields negligible or no loss for a range of γ values (except for values close to 1).

The behavior of these two streams were representative of our observations for a variety of streams used in our experiments. A potential limitation of our experiments is that we assumed a single streaming client environment. Although our experiments were carried out on a production wireless network with other background traffic (and account for the resulting jitter), we expect streaming to multiple concurrent client to introduce additional jitter (resulting in reduced savings or increased loss). A thorough study of such effects is the subject of ongoing research.

7 Related Work

Recently, power-aware resource management for mobile hosts has received increasing research attention and a variety of techniques targeting various system components such as the hardware, the operating system, the compiler and the application itself have been proposed. For instance, at the hardware level, processors can employ intelligent power management techniques that monitor the CPU usage and dynamically vary the CPU clock frequency based on the observed load [9, 15]. A recent study has also proposed offline processing of MPEG video streams to determine the CPU voltage settings for each frame during decode time—the techniques lowers energy usage by choosing the smallest feasible CPU voltage setting for each frame so as to meet real-time decoding constraints [13]. At the operating system level, recent studies have investigated intelligent CPU scheduling techniques that lower the energy required to schedule a set a tasks [2, 6]. Formal methods for adapting multimedia applications using energy-aware CPU scheduling has been studied in [19]. At the programming language level, compiler research has investigated techniques for generating executables that reduce energy consumption [10].

From the perspective of multimedia applications, a recent work on energy-aware adaptation assumed a server that stores streaming media data encoded at different bit rates and resolutions and showed that requesting an appropriate resolution version from the server can yield energy savings[8]. Rather than assuming such pre-encoded streams, a particular focus of our work is to produce such streams on-the-fly at the proxy. The design of transcoding techniques to derive new streams on-the-fly has been studied extensively [1, 3, 12, 18]. Our work differs from past work in that we do not propose a new transcoding technique per se, rather we apply existing transcoding techniques to perform power-friendly transformations.

The idea of detecting idle periods and switching the client network interface to passive mode during such periods was first proposed in [4]. The study examined off-the-shelf servers for popular streaming formats Windows Media and Real Media and showed that network transmission of Windows Media streams tends to be more predictable than Real Media. They used a history-based mechanism to predict arrival times and showed that such a mechanisms can reduce energy consumption by over 50%. A follow-on work that will appear at an upcoming conference examines the use of traffic-shaping transmission mechanisms in conjunction with such history-based mechanisms to extract further energy savings [5]. We note that this enhanced scheme is similar to our first approach that uses proxy-side smoothing with client-side history-based prediction. Although the specific techniques for smoothing and prediction are somewhat different, the resemblance is intentional—one of the goals of our work was to examine how such an approach would compare to one where the proxy supplied the client with explicit control information. Our experiments demonstrated that such control information can indeed help the client to extract further savings; in fact, we believe that a hybrid approach that combines smoothing, measurement-based estimates and proxy-supplied control information may offer the best tradeoffs in heterogeneous environments and our ongoing work focuses on the design such techniques. We also note that there are important philosophical differences between the two approaches. The approach taken by Chandra et. al. focuses on popular, though proprietary, streaming formats such as Windows Media and Real, and consequently, they are constrained by the proprietary nature of the signaling and transmission protocols used by these formats. Due to such constraints, application-level techniques such as traffic shaping must be carefully designed (e.g., the proprietary flow-control techniques used by these protocols might mistake the delays introduced by traffic shaping to be an indication of network congestion); hence, energy-saving transmission schemes must be carefully designed to be avoid unintended side-effects. In contrast, our research assumes no restrictions on the transmission or signaling protocols and attempts to devise the most appropriate proxy-side and client-side technique to extract energy savings.

8 Concluding Remarks

In this paper, we argued that the resource-hungry nature of multimedia applications necessitates the use of energy-saving techniques to enable them to run on resource-poor mobile devices. To address this issue, we proposed two proxy-based techniques to reduce the energy needs of mobile streaming media applications. Our first technique performed power-friendly transformations on client-requested streams to reduce their energy needs. To perform such transformations, we devised models to estimate the energy needs of a particular stream and used our models to determine the transformations that are necessary to limit the energy needs of the stream to client-specified values. We then proposed techniques to transmit the transformed stream to the mobile client in an intelligent fashion. Our transmission techniques enable a client to reduce the energy wastage at its network interface while waiting for data packets. We implemented our techniques into a prototype proxy and a mobile client. Our experimental evaluation of the prototype showed that: (i) our power-friendly transformations are effective over a range of bit rates, resolutions and encoding parameters for MPEG-encoded streams, and (ii) our network transmission and reception techniques can reduce the potential energy wastage at the network interface by 65-98%.

References

[1] S. Acharya and B. C. Smith. Compressed Domain Transcoding of MPEG. In *Proceeding of IEEE Conference on Multimedia Computing Systems (ICMCS), Austin TX*, 1998.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS), London, England*, December 2001.

[3] N. Bjork and C. Christopoulos. Video transcoding for universal multimedia access. In *Proceedings of the Eighth ACM Conference on Multimedia, Los Angeles, CA*, November 2000.

[4] S. Chandra. Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference (MMCN)*, pages 85–99, January 2002.

[5] S. Chandra and A. Vahdat. Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats. In *Proceedings of the Usenix Annual Technical Conference*, June 2002.

[6] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, October 2001.

[7] W. Feng and J. Rexford. Performance Evaluation of Smoothing Algorithms for Transmitting Prerecorded Variable-Bit-Rate Video. *IEEE Transactions on Multimedia*, September 1999.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th SOSP, Kiawah Island Resort, SC*, December 1999.

[9] C. Hughes, J. Srinivasan, and S. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO-34)*, December 2001.

[10] U. Kremer, J. Hicks, and J. Rehg. A Compilation Framework for Power and Energy Management on Mobile Computers. In *Proceedings of the 14th International Workshop on Parallel Computing (LCPC'01)*, August 2001.

[11] S.S. Lam, S. Chow, and D.K.Y. Yau. An Algorithm for Lossless Smoothing of MPEG Video. In *Proceedings of ACM SIGCOMM'94, London*, 1994.

[12] K. Mayer-Patel and L. Rowe. Design and Performance of the Berkeley Continuous Media Toolkit. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, pages 194–206, January 1997.

[13] M. Mesarina and Y. Turner. Reduced Energy Decoding of MPEG Streams. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference (MMCN)*, pages 73–84, January 2002.

[14] Berkeley MPEG Player Home Page. http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html, 2001.

[15] P. Pillai and K. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 89–102, October 2001.

[16] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *Proceedings of ACM SIGMETRICS, Philadelphia, PA*, May 1996.

[17] TMPGEnc Home Page. http://www.tmpgenc.net/e_main.html, 2002.

[18] P. Yin, M. Wu, and B. Liu. Video Transcoding By Reducing Spatial Resolution. In *Proceedings of the IEEE Int'l Conf. on Image Processing (ICIP), Vancouver, Canada*, September 2000.

[19] W. Yuan, K. Nahrstedt, and X. Gu. Coordinating Energy-Aware Adaptation of Multimedia Applications and Hardware. In *Proceedings of 9th ACM Multimedia Middleware Workshop, Ottawa, Canada*, October 2001.