DYNAMO: A Cross-Layer Framework for End-to-End QoS and Energy Optimization in Mobile Handheld Devices

Shivajit Mohapatra, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian

***Abstract*— In this paper, we present the design and implementation of a cross-layer framework for evaluating power and performance tradeoffs for video streaming to mobile handheld systems. We utilize a distributed middleware layer to perform joint adaptations at all levels of system hierarchy - applications, middleware, OS, network and hardware for optimized performance and energy benefits. Our framework utilizes an intermediate server in close proximity of the mobile device to perform end-to-end adaptations such as admission control, intelligent network transmission and dynamic video transcoding. The knowledge of these adaptations are then used to drive "on-device" adaptations, which include CPU voltage scaling through OS based soft realtime scheduling, LCD backlight intensity adaptation and network card power management. We first present and evaluate each of these adaptations individually and subsequently report the performance of the joint adaptations. We have implemented our cross-layer framework (called DYNAMO) and evaluated it on Compaq iPaq running Linux using streaming video applications. Our experimental results show that such joint adaptations can result in energy savings as high as 54% over the case where no optimization are used while substantially enhancing the user experience on hand-held systems.**

***Index Terms*— Cross layer design, distributed middleware, power/energy optimization**

I. Introduction

RAPID advances in processor and wireless networking technology have rendered most modern handheld devices multimedia capable. While the integration of such devices into distributed environments have come as a boon to application designers (video streaming, gaming, virtual reality applications etc.), the short battery lifetimes of these devices remain a scourge to the widespread acceptance of these "cool" applications. For example, multimedia applications have distinctive quality of service(QoS) and processing requirements which tend to make them extremely resource-hungry and energy intensive. Consequently delivering high quality multimedia content to mobile handheld devices while preserving their service lifetimes presents competing design requirements.

Modern mobile devices also present several opportunities to achieve power savings by providing the capability of operating various hardware components in multiple low duty-cycle operating modes. Consequently, several interesting solutions for energy optimization have been proposed at various computational levels - system cache and external memory access optimization [10], [21], [20], dynamic voltage scaling (DVS) [27], [23], dynamic power management of disks and network interfaces [12], [7], [8], efficient compilers and application/middleware based adaptations [17], [25], [26], [16].

However, we observe an interesting disconnect in the research initiatives undertaken at each system level. Power optimization techniques developed at each level have remained seemingly incognizant of the other abstraction hierarchy levels, potentially missing opportunities for substantial improvements achievable through cross-level integration. The cumulative power gains achievable by joint adaptations at each level can be potentially significant; but it also requires a study of the trade-offs involved and the customizations required for unified operation.

To successfully achieve such joint end-to-end and cross-layer adaptations, we (i) identify techniques within each system level that can benefit from such adaptations (ii) determine parameters available at specific system levels that can be exposed to other levels (iii) design techniques at each level to integrate these "identified" parameters into their adaptation algorithms (iv) use adaptations at a machine located in close proximity to the mobile device (also referred to as a "proxy" server in the rest of the paper) to drive on-device adaptations and (v) provide a continuous feedback loop for exchange of local and distributed state. We have designed an API interface that can successfully enable a continuous dialogue between the system levels for communicating state related information. The advantage of such a framework is that it presents an opportunity at each system level to expose its state information to other levels, as well as incorporate information from other levels within its own adaptations.

We present our system in the context of video streaming to mobile handheld devices. Our local (or "on-device") adaptation schemes include dynamic power management techniques for the LCD backlight, the wireless NIC and the CPU. For our global (between a proxy and the mobile device) adaptations, we employ three specific techniques: (i) video transcoding (ii) energy-aware admission control and (iii) adaptive network traffic shaping. We employ a distributed middleware (called DYNAMO) to control and coordinate these joint adaptations.

II. System Model

We assume the system model shown in Fig. 1. The system includes multimedia servers, intermediate proxies and mobile

Manuscript received May 25, 2006; revised December 1, 2006. This work was partially supported by ONR MURI Grant N00014-02-1-0715 and NSF award ACI-0204028. Some of the techniques presented in this journal version are based on the first author's earlier reported work in [1], [24] and [9].

Shivajit Mohapatra is with the Applications Research Center, Motorola Labs, Schaumburg, Illinois (e-mail: mopy@labs.mot.com).

Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian are with the School of Information & Computer Sciences, University of California, Irvine, CA 92697-3425 (e-mail: {dutt,nicolau,nalini}@ics.uci.edu).

Digital Object Identifier 10.1109/JSAC.2007.070509.

client devices. The model includes a multimedia server, a proxy server close to the wireless clients and a set of mobile clients requesting content from the media server. All communication between the handheld client and the media servers are routed through the proxy server, that can transcode the video stream in realtime.

The mobile device is represented as having four system levels - application, middleware, OS and hardware. The energy consuming components (e.g. CPU, NIC, LCD display) are at the hardware level. The next higher level is the operating system, which has access to the physical devices through well defined driver interfaces. We have enhanced the operating system to support cross-layer energy management. Next in the hierarchy is the distributed middleware level. Our framework adds this layer for coordinating external and "on-device" adaptations. The middleware layer in our system can be considered to have three abstract components to it - a "system" component that resides within the OS (with OS level code integration), a "network" component that implements a communication protocol to talk with a proxy server and a "user level" component that performs the various middleware based adaptations using the information gathered from the network, the OS and the user (or application). Finally, the application layer provides user profiles (specific user preferences) and application specific context to the middleware and also performs dynamic QoS negotiations with the middleware.

The distributed middleware executes on both the mobile device and the proxy. On the device, the middleware makes the executing modules both "*local state aware*" and "*global state aware*". By *local state awareness* we refer to everything that resides on the physical device the application is executing on - for example residual battery information, backlight settings, operating CPU frequency, memory information, as well as information regarding other executing applications. We also utilize the middleware to store and expose state information available at various system layers to other layers. On the other hand, *global awareness* refers to information that is typically not available on the local device (e.g. bandwidth availability, network congestion, mobility information). The distributed part of the middleware (running on the proxy) is responsible for determining these parameters. While most current power management strategies exhibit local awareness in their approaches, very few power management approaches have tried to exploit the information available in both global and local contexts. The distributed middleware plays the role of both exposing global and local contexts to applications and proxies.

A. System Utility and Video Quality Modeling

System utility defines the overall value of the system to the end user and therefore a higher system utility is desirable. One of the key challenges we faced was to define the notion of system utility for our end-to-end system. To achieve the tradeoffs between energy and QoS for multimedia applications, the system utility must relate energy to video quality in some quantitative manner. This requires that the utility contain both subjective (quality of video on handheld) and objective elements (energy consumption which can be measured accurately using a data acquisition board). In order to design an effective utility function for the system, we conducted an extensive survey to understand human perception of video quality on handheld devices. We also studied the impact of video quality on the energy consumption a handheld system. In this section, we summarize the results of our study. Note that we leverage this study to design several of our distributed middleware adaptations described later.

System Utility: In order to achieve an optimal balance between power and performance, we introduce a notion of "*Utility Factor* U_F" for a system, and try to optimize the U_F for the system. This approach precludes our system from aggressively optimizing for power at the expense of performance and vice-versa. Under this strategy, we try to utilize all the available energy to maximize the user experience ($\equiv$ video quality). Let U_F be a measure of "user satisfaction" and we define it as follows: given the residual energy E_{res} on a handheld device, a threshold video quality level Q_a ($Q_{MAX} > Q_a > Q_{MIN}$), and the duration of the video playback T, the U_F of the system is non-negative, if the system can stream the highest admissible quality of video to the user such that the time, quality and the power constraints are satisfied; otherwise U_F is negative. Let P_{VID} denote the average power consumption rate of the video playback at the handheld and Q_{PLAY} be the quality of video streamed to the user by the system. Using the above notation, we define U_F as follows:

$$U_F = \begin{cases} Q_{PLAY} - Q_a & IFF\ P_{VID} * T < E_{res} \quad \& \\ & Q_{PLAY} \geq Q_a \\ -1 & Otherwise \end{cases}$$

Note that the above characterization can form the basis of dynamic video transcoding assuming that the various energy, video quality and time parameters are available.

Energy-Aware Video Transcoding: In our system, we employ dynamic transcoding to change the video application characteristics at runtime. Two key questions arise when performing dynamic transcoding: (i) what video qualities should the original video be transcoded to and (ii) what is energy requirement of the transcoded video? In [24], we extensively performed a survey to relate user perception to video quality. We present the following interesting observations and conclusions from that survey:[1]

(i) Almost 90% of the subjects were able to differentiate between close video quality levels on laptop/desktop systems; however only about 20% were able to differentiate between close quality levels on a handheld computer.

(ii) It was hard to programmatically identify video quality parameters(a combination of *bit rate, frame rate and video resolution*) that produced a user perceptible change in video quality and/or a noticeable shift in power consumption.

(iii) For all the video streams to handheld devices, it was enough to use just three standard intermediate formats (e.g SIF 320x 240), Half SIF(240x160) and Quarter SIF(160x120)) for frame resolution values. Other resolutions did not produce a perceptible quality change or power uptake compared to the nearest SIF encoded video with similar bit and frame rates.

Based on these conclusions and our extensive experiments, we identify the eight dynamic video stream transformation

[1]Note that with newer iPaq models the user perception might change

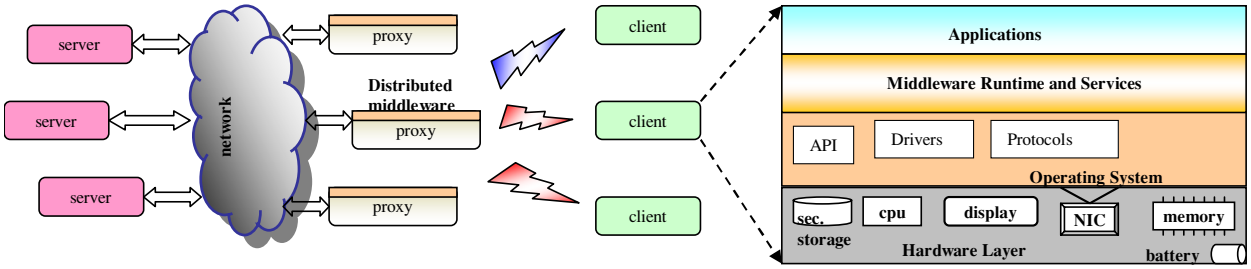

Fig. 1. System Model.

TABLE I

ENERGY-AWARE TRANSFORMATIONS FOR COMPAQ IPAQ 3650 WITH BRIGHT BACKLIGHT, CISCO 350 SERIES AIRONET WNIC CARD, FOR THE GRAND THEFT AUTO VIDEO(ENCODED USING MPEG-1), USING POCKET VIDEO PLAYER(CE, HTTP STREAMING) AND VIDEOLAN CLIENT(LINUX, UDP STREAMING).

Quality	Transformation Parameters	Avg. Power (Windows CE)	Avg. Power (Linux)
Like Original (No improvement required)(Q8)	SIF, 30fps, 650Kbps	4.42 W	6.07 W
Excellent (Q7)	SIF, 25fps, 450Kbps	4.37 W	5.99 W
Very Good (Q6)	SIF, 25fps, 350Kbps	4.31 W	5.86 W
Good (Q5)	HSIF, 24fps, 350Kbps	4.24 W	5.81 W
Fair (Q4)	HSIF, 24fps, 200Kbps	4.15 W	5.73 W
Poor (Q3)	HSIF, 24fps, 150Kbps	4.06 W	5.63 W
Bad (Q2)	QSIF, 20fps, 150Kbps	3.95 W	5.5 W
Terrible (poorer quality not acceptable)(Q1)	QSIF, 20fps,100kbps	3.88 W	5.38 W

parameters (Table I) for our proxy-based realtime transcoding and profile their average power consumption values. Note that similar device specific transformations can be made for other portable computers. This approach provides us with two significant advantages: (a) real time stream quality transformations can be performed with no overheads of dynamically determining quality degradation parameters. The system now only needs to optimize for only eight different video quality levels as opposed to potentially infinite possibilities when using a dynamic transcoder. (b) A simple feedback loop between the device and the proxy is sufficient to control the adaptations at the proxy.

III. END-TO-END APPROACH TO CROSS LAYER ADAPTATION

Fig. 2 presents the architecture of our system. On the proxy server, we employ our distributed middleware to perform two primary adaptations. As shown in the figure, the "Remote adaptation manager" uses feedback from the device, the end-to-end state (e.g network noise, mobility) and a device specific rule base (profiled information) to determine the quality parameters of the video stream that still maintains the highest system utility within the available energy budget of the device. It then transcodes the video stream into an appropriate quality and performs intelligent traffic shaping (discussed in the next section). Simultaneously, it informs the device on a control channel about the remote changes so that the mobile device can adapt accordingly.

On the mobile system, the middleware has two primary components: (i) the global adaptor is responsible for receiving control information about remote adaptations and taking appropriate steps to jointly adapt various local techniques, as well as sending device specific feedback to the proxy. (ii) the local coordinator contains a set of modules that provide access to operating system and hardware modules through adaptors implemented at that operating system level. These adaptors are responsible for directly accessing hardware power management functions and coordinating local variations in resource requirements. Each of these local coordinator modules use information available at the global adaptor to adapt the power states of the power consuming components (cpu, NIC and backlight).

Clearly, in our system the large scale adaptations happen when the residual energy of the mobile system changes enough to warrant a transition in the quality of the video stream. A change in the video stream can possibly trigger local adaptations to change the CPU frequency settings, the network power management settings and the backlight settings. However, these are adaptations are infrequent compared to the local adaptations necessary to manage temporary variations on the local device. While the power settings for the LCD backlight and the network interface card are largely dependent on the video stream adaptations controlled remotely at the proxy, CPU frequency adaptations need to account for the local application load on the system. We present our proxy-based and local adaptations in the subsequent sections.

A. End-to-End Adaptation

The goal of end-to-end adaptation is to utilize the knowledge of end-to-end system state (e.g. other mobile devices sharing the network, network noise, mobility, proxy server load) along with the feedback received from a mobile device

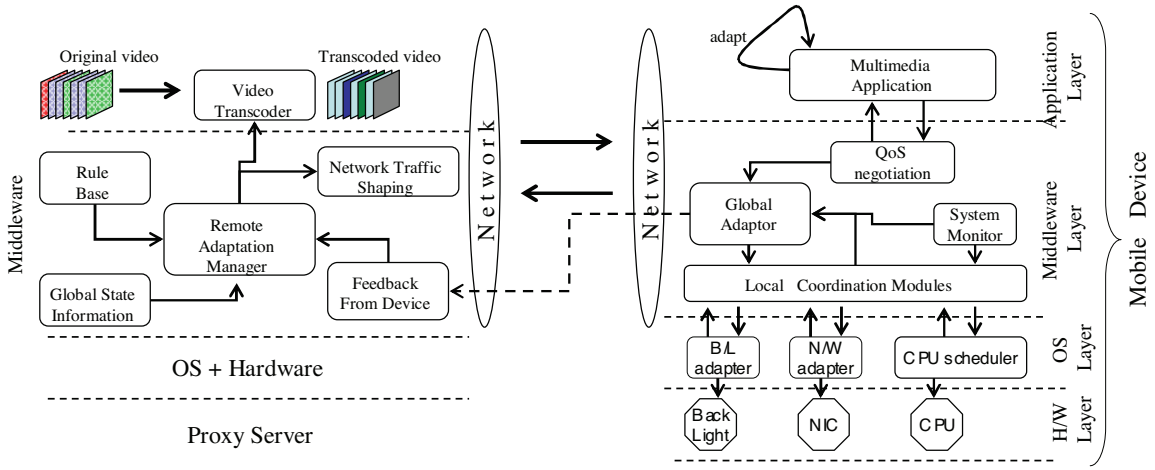

Fig. 2. Architecture of the End-to-End Cross-Layer Adaptation Framework.

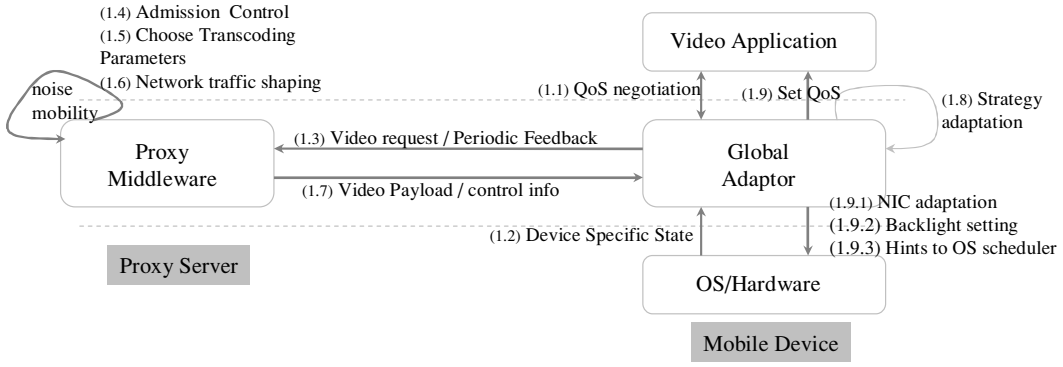

Fig. 3. Protocol for End-to-End Adaptation.

to achieve the following: (i) determine whether a video request can be satisfied with the current energy budget of the device (energy-aware admission control). The proxy middleware performs two important tasks: (a) energy-aware video transcoding and (b) intelligent network traffic shaping. The adaptations may result in a change in the overall application characteristics and may require the "on-device" strategies to adapt to the changes. We achieve this by proactively sending such information over to the device as control information and use our cross-layer framework to effect the dynamic adaptations at the device. While the transcoding ensures that the highest quality stream is being delivered to the device within the available energy budget, the intelligent network transmission can facilitate for effective power management of the network card at the device. We now introduce these middleware techniques.

1) Energy-Aware Admission Control and Stream Transformations at the Proxy:

The middleware on the proxy utilizes the feedback from the device, to continuously monitor the *utility factor* (U_f) of the system. It performs an energy aware admission control initially to identify whether a request can be scheduled. Subsequently, it monitors the residual energy at the device and streams the highest allowable quality video (performing realtime video transcoding) that meets the energy budget at the device and maintains an acceptable U_f (≥ 0). We characterize the admission control and stream transformations using the following parameters:

- T_{start},T_{cur}: The start time of the video streaming and the current time respectively.
- T : duration of the entire video.
- E_{res} : Residual energy of the device.
- Q_1 .. Q_N : The 'N' video quality levels with the associated video transformation(Table I) and architectural optimization parameters.
- P_1 .. P_N : The average power consumption(e.g. Table I) of the corresponding quality levels of the video.
- I_f : Interval between successive feedbacks. This is decided by administrative policies.
- Q_a : Lowest user acceptable video quality for the request.
- R_i, F : The initial request and the feedback from the device respectively. The initial request contains E_{res}, Q_a and device specific details such as (NIC model, CPU arch etc). The feedback(F) simply contains values for E_{res} and Q_a. Note that R_i is used for initial admission and F is used for maintaining an acceptable U_f.

Fig 4 outlines the high level admission control algorithm employed by the proxy middleware. An initial admission control is performed to check whether the video can be streamed at the user requested quality level for the entire length of the video. Subsequently, the stream quality is adjusted using the periodic feedbacks from the device.

```
WHILE (TRUE)
{
    IF ( R_i OR F received)
    {
        Determine i, such that Q_i = Q_a;
        IF ( (T – (T_cur – T_start)) * P_i <= E_res && U_f == acceptable)
            Find Highest Q_i that satisfies the above
            inequality. Set the corresponding transformation
            parameters for the transcoder.
        ELSE
            REJECT the request OR (Negotiate video quality)
    }
}
```

Fig. 4. Admission Control for single user.

2) Network Traffic Regulation: In this section, we develop a proxy-based traffic regulation mechanism to reduce energy consumption by the device network interface. Our mechanism (a) dynamically adapts to changing network(e.g noise) and device conditions, (b) accounts for attributes of the wireless access points (e.g. buffering capabilities) and the underlying network protocol (e.g. packet size) and (c) uses the proxy to buffer and transmit optimized bursts of video along with control information to the device. However, even though packets are transmitted in bursts by the proxy, the device receives packets that are skewed over time Fig. 5; this cuts power savings, as the net *sleep* time of the interface is reduced. The skew is caused due to the ethernet access protocol(e.g CSMA/CD) and/or the fair queueing algorithms implemented at the AP. Our mechanism optimizes the stream, such that the optimal video bursts sizes are sent for a given noise level, thus maximizing energy savings without performance costs.

Wireless network interface(WNIC) cards typically operate in four modes: *transmit, receive, sleep and idle*. We estimated the power consumption of the Cisco Aironet 350 series WLAN card to have the following power consumption characteristics: *transmit*(1.68W), *receive*(1.435W), *idle* (1.34W) and *sleep*(0.184W) which agree with the measurements made by Havinga et al. in [19], [31]. This observation [7] suggests that significant energy savings can be achieved by transitioning the network interface from *idle* to *sleep* mode during periods of inactivity. The use of bursty traffic was first suggested by Chandra [7], [8] and control information was used for adaptation in [29]. We improve on the above approaches by modeling the wireless access point buffer limitations and accounting for the delays and losses incurred due to fair queuing at the access points. [32] presents a power saving mechanism wherein transcoded video data is buffered on a mobile system and decoded locally and wireless adapter is transitioned to sleep until the buffered frames are decoded.

We analyze the above power saving approach using a realistic network framework (Fig. 5), in the presence of noise and AP limitations. The proxy middleware buffers the transcoded video and transmits I_t seconds of video in a single burst along with the time τ=I_t for the next transmission as control information. The device then uses this control information to switch the interface to the active/idle mode at time τ + $\gamma \times D_{EtoE}$, where γ is an empirical estimate between zero and one and D_{EtoE} is the end-to-end network delay with no noise [29].

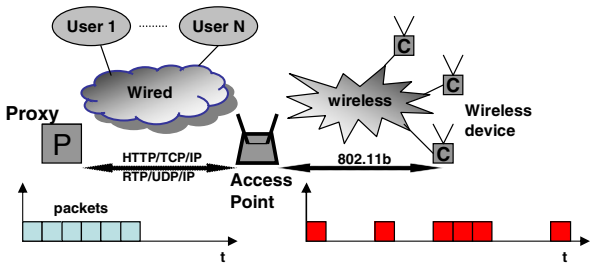

Fig. 5. Skewed packet delivery in wireless network.

Let B be the average video transmission bit-rate, F the video frame rate, S_N the packet size used by the underlying network protocol. Let there be N users in the system. We model each "*other*" user as a Poisson process that injects packets into the network with the packet inter-arrival times following an exponential distribution with rate λ, with a density function of $f(t) = \lambda.e^{-\lambda t}$. Therefore, the number of packets introduced into the network by each user in an interval 't' has an expected value $E(t) = \lambda.t$. Assume that the AP employs a simple round robin service for transmitting packets and let L_{AP} be the buffer length(in number of packets) available at the AP for downstream traffic. Let BW_s, BW_d be the bandwidth available to the wireless device for transmitting and receiving data; T_{AP}, P_d be the queueing transmission delay per packet of the AP and propagation delay of the wireless network respectively. Using the above characterization, the number of packets per frame $\alpha = \frac{B}{8 \times S_N \times F}$ and the total number of packets transmitted in one burst $P_b = \alpha \times F \times I_t$.

To simplify the analysis, we further assume that there are no loss of packets at the AP due to weak signal strengths or collisions. However, packets do get dropped if the AP buffer capacity (L_{AP}) is less than the number of arriving packets. A queueing theory analysis is used to predict packet loss rates at the AP. We omit the details due to space constraints. If T_b is the time taken to transmit the burst by the proxy, then the total expected number of packets received at the AP in that interval is $\sigma = P_b + \sum_{k=1}^{k=N-1} E_k(T_b)$, where E_k is the expected value of noise from by user k. The worst case expected transmission delay experienced by the last packet in the burst is $D = \sigma \times T_{AP}$. Using the above approach of bursty transmissions, the total "*sleep*" time(δ) of the network interface can be calculated as $\delta = \tau - (D + \gamma \times D_{EtoE})$, neglecting the propagation delay of the final packet. As significant power gains are only achieved when the network interface is in the sleep mode, the total power savings are $P_{saved} = \delta * (P_{IDLE} - P_{SLEEP})$. Observe that large values of I_t can result in packet losses at the access point and/or buffer overflows at the device. We acknowledge that a QoS aware preferential service algorithm at the access point can impact power management significantly. The above analysis can be used by an adaptive middleware at a proxy to calculate an optimal I_t(burst length) for any given video stream and noise level. Note that energy overhead for buffering the video packets is not affected by using our strategy because the number of read and write memory operations remain unchanged irrespective of the memory buffer size. We also note that this approach is practical in IEEE 802.11 networks employing pure PCF (Point Coordination Function).

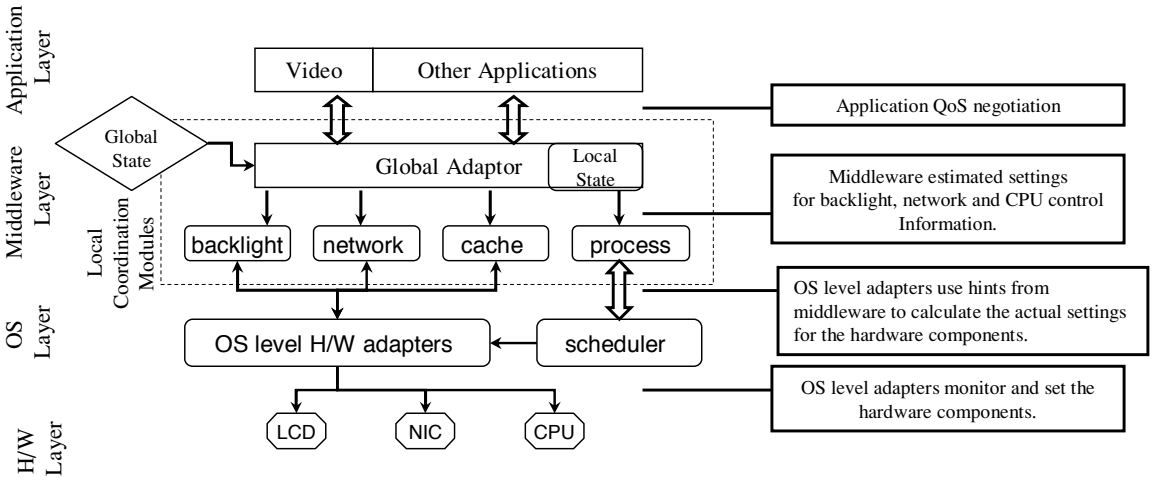

Fig. 6. Local Coordination using cross-layer adaptation on a mobile system.

B. "On-Device" Cross-Layer Adaptation

The end-to-end adaptations change both the application and the network transmission characteristics of the video stream dynamically. Consequently, the local adaptations have to be triggered to handle changes in (i) total CPU demand, (ii) backlight settings for the new video quality and (iii) the wireless network adaptor power management scheme.

The "on-device" adaptations are designed to react to changes induced by adaptations at the proxy (e.g. transcoding) and/or by local changes at the application layer (e.g. higher QoS requirement by the user, new application etc.). In our system (Fig. 6), the global adaptor module assimilates information regarding these dynamic changes by interfacing with the application layer and the proxy server. It is also responsible for updating local state information periodically to the proxy and negotiating QoS dynamically with the applications. Subsequently, it provides the middleware adaptation modules (backlight, network, process scheduler in Fig. 6) with control information to perform dynamic adaptations.

Fig. 7 shows the periodic information exchanged between the mobile device and the proxy. In our system, we send state information to the proxy every minute. However, information from the proxy is sent to the device only when adaptations lead to changes in stream or network characteristics. For example, a change in the video stream characteristics might need newer settings for the backlight, the CPU and the network adapter sleep times. Similarly starting newer applications might require the CPU settings to be modified so that all application deadline requirements are satisfied. Once the local adaptations have been performed the local state of the device is updated. Local adaptations can also lead to QoS renegotiations with the application. For example, if an application is continuously missing processing deadlines, the adaptor can negotiate a lower QoS level with the application. The adapter on the device assimilates this information and communicates it to the middleware adapatation modules. These modules in turn talk to the lower level system modules that have access to the actual hardware. By doing this, we are able to proactively adapt to changes in the video stream without having to detect these changes and then reactively adapt the system. The next section explains how our framework supports backlight adaptation for video applications.

1) Dynamic Backlight Adaptation:

The LCD backlight in mobile devices account for a significant percentage of energy consumption (e.g., around 25%-30% for a Compaq iPAQ 3650). Typical handheld devices provide several intensities at which a backlight can operate (0-255 for ipaqs running linux). A higher backlight intensity can result in a significant energy overhead. Video applications however present the opportunity to dynamically dim the backlight while simultaneously scaling the pixel luminance video frames to compensate for the reduced perception fidelity. We have studied adaptive backlight dimming with respect to video applications [9] and present our scheme for backlight compensation for the video qualities profiled above.

The perceptual luminance intensity of LCD display is determined as a product of the backlight intensity and the pixel luminance of the displayed video frame. The human eye can detect a change in the brightness of the display, if either of these two components is adjusted. However, this also means that the backlight intensity and the pixel luminance can compensate for each other. We can therefore use profiling to identify how individual pixel levels can be enhanced for our videos (done offline) and the corresponding backlight levels to use for these video frames to compensate for this enhancement so that overall user perception of the video remains unaffected beyond a certain acceptable threshold. While decoding these frames have negligible computing overhead, we can conserve battery energy by dynamically reducing the backlight intensity of our mobile system.

Let the backlight brightness level and the pixel luminance value be L and Y, respectively, and the perceived display luminance intensity I (not to be confused with $\boldsymbol{I_t}$ used earlier). We may denote I using Equation (1).

$$\boldsymbol{I = \rho \times L \times Y} \tag{1}$$

where $\boldsymbol{\rho}$ is the constant ratio, denoting the transmittance attribute of the LCD panel, and $\boldsymbol{\rho \times Y}$ turns out to be the transmittance of the pixel luminance. We may reduce the backlight level to $\boldsymbol{L'}$ by multiplying $\boldsymbol{L}$ with a dimming factor

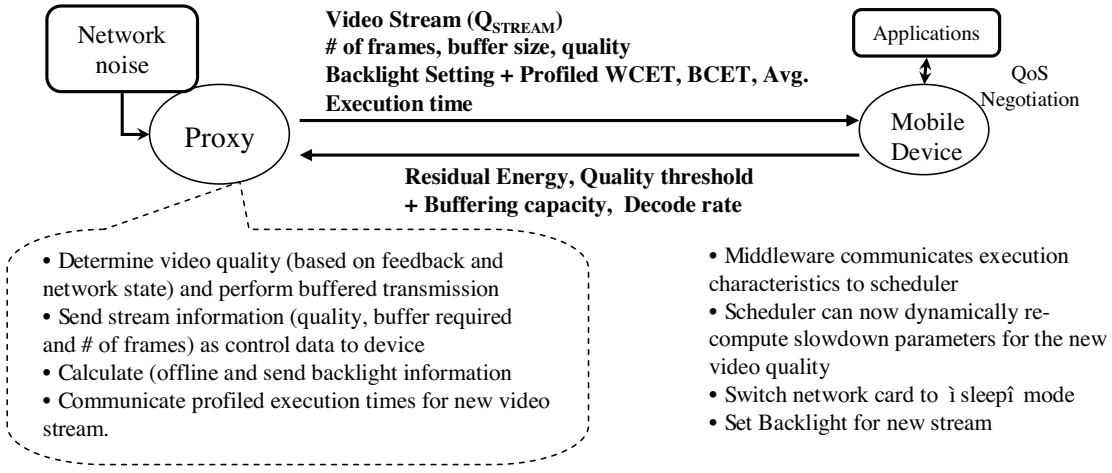

Fig. 7. Periodic FeedBack Information and Joint Adaptations.

Fig. 8. Original Image.

Fig. 9. Compensated Image.

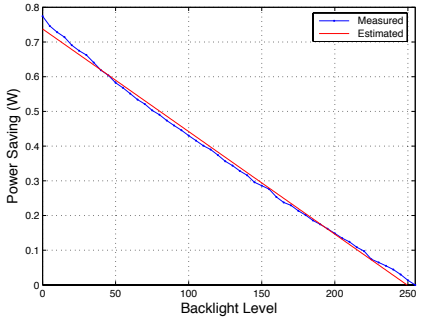

Fig. 10. Power saving.

α, i.e., $L' = L \times \alpha$, $0 < \alpha < 1$. To maintain the overall display luminance I invariable, we need to boost the luminance of the pixel to Y'. Since the pixel luminance value is normally restricted by the number of bits that represent it (denoted as n), Y' may be clipped if the original value of Y is too high or the α is too low. The compensation of the backlight is described in Equation 2.

$$Y' = \begin{cases} Y/\alpha, & if\ Y < \alpha \times 2^n \\ 2^n, & if\ Y \geq (\alpha \times 2^n) \end{cases} \quad (2)$$

Combining Equation (2) and Equation (1), we have

$$I' = \begin{cases} I, & if\ Y < \alpha \times 2^n \\ \rho \times L \times \alpha \times 2^n & if\ Y \geq (\alpha \times 2^n) \end{cases} \quad (3)$$

Equation (3) clearly shows that the perceived display intensity may not be fully recovered, instead, it is clipped to $\rho \times L \times \alpha \times 2^n$ if the $Y \geq (\alpha \times 2^n)$.

To illustrate the adaptation, in figure 8 we show the original first frame of a regular news video clip and in figure 8 the same frame after the backlight dimming and the pixel luminance compensation is shown. Figure 10 shows the power savings due to backlight is linear with the intensity of backlight used.

We also propose two supplementary methods to adapt to frequent backlight dimming across consecutive frames. First, we apply a low-pass digital filter to eliminate any abrupt backlight switching that is caused by the unexpected sharp luminance change. The passband frequency is determined by the subjective perception of the flicker moment and the frame display rate. Second, we propose to quantize the number of backlight levels, i.e., any backlight level between two quantization values can be quantized to the closest level, by which we prevent the needless backlight switching for small luminance fluctuations during one scene. In our experiments, we quantize all 256 levels to Nlevels (N=5 in our study). We switch the backlight level only if the calculated dimming factor changes drastically enough, so that it falls into another quantized level.

2) CPU Adaptation: Dynamic Frequency/Voltage Scaling:

Dynamic Voltage Scaling (DVS) has been widely studied as a technique for reducing CPU energy consumption by operating the CPU in low duty cycle modes to exploit application slack [22], [11]. A processor normally operates at a specific supply voltage V and clock frequency f. The dynamic power dissipated by the CPU (and any other CMOS circuits due to switching activity, in addition to the static component) varies linearly with frequency and circuit capacitance, and quadratically with voltage: $P \propto C \times f \times V^2$. The disadvantage of applying dynamic voltage scaling is its power-performance tradeoff.

In MPEG decoding, frames are processed in a fraction of the frame delay ($F_d = 1/frame\_rate$). The actual frame decoding time D depends on the type of MPEG frame being

processed (**I**, **P**, **B**) and is also influenced by the cache configuration $(size(S), Associativity(A))$ and DVS setting (f, V). In this study, we assume a buffered based decoding, where the decoded frames are placed in a temporary buffer and are only read when the frame is displayed. This allows us to decouple the decoder from the displaying; decoding time it still different for different frame, but we can assume an average D for a particular video stream/quality. The difference between the average frame delay and actual frame decoding time gives us the slack time $\theta = F_d - D$. When we perform DVS, we slow down the CPU so as to decrease the slack time to a minimum.

Let us assume that the initial configuration for the CPU before applying DVS is (f_0, V_0). Frame decoding time is dependent on the cache configuration (slightly) and the frequency at which the processor was running during profiling $D(S, A, f_0)$. The equation for the frame decoding slack time can be written as: $\theta_0 = F_d - D(S, A, f_0)$. After applying voltage scaling, the new values for frequency and voltage are (f_{new}, V_{new}). The frame decoding time changes: $D_{new} = F_d - D(S, A, f_0) * f_0/f_{new}$ and the slack time is $\theta_{new} = F_d - D(S, A, f_0) * f_0/f_{new}$. The optimal solution is attained when the slack time θ_{new} is closest but not less than zero. For a particular quality level Q_k the frame delay F_d is a constant (known). The value for f_{new} is optimal when it minimizes θ_{new} (the slack time). [2]

3) Quality-driven Data Cache Reconfiguration:

We also investigate the effect of data cache optimizations for video playback. Power consumption for the cache depends on the runtime access counts: while hits result in only a cache access, misses add the penalty of accessing the main memory (external). For video decoding, the best configuration of the cache is not easily predictable; in fact, it may even be counter-intuitive. Decreasing the cache size, or the associativity, yields a reduction in cache power consumption; on the other hand it will generate more memory traffic due to the more frequent misses and line cache replacements, increasing power uptake. However for our discrete video quality levels (i.e. finite application domain), we can generate optimized configurations through offline simulation and profiling of the streams. Our cache reconfiguration goal is optimizing energy consumption for a particular video quality level Q_k. In general, cache power consumption for a particular configuration and video quality is given by the function $E_{cache,k}(S, A)$. By profiling this function for the entire search space (S, A) of available cache configurations, we generate a cache energy variation graph shown in Fig. 11. Depending on the video quality Q_k streamed, the middleware/OS can then fine tune the organization of the data cache for the stream (S_k^{opt}, A_k^{opt})., given that such a functionality is supported. We found out that for all video qualities an optimized operating point exists and it improves cache power consumption by up to 10-20% (as opposed to a sub-optimized configuration).

[2]When implementing a DVS policy on a platform such as commercial PDA (Compaq IPAQ StrongARM), the assumption that the overhead of changing frequencies is negligible does not hold. As pointed out by Saweong *et. al.* [28] and confirmed by our experiments, when the processor frequency is switched, the overhead varies from **20** to **30** milliseconds because other components such as SDRAM and display have to be re-synchronized to function properly.

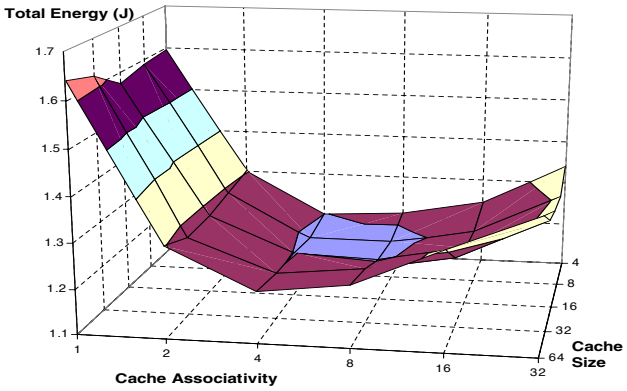

Fig. 11. Cache Energy Variation on Size and Associativity.

Cache behavior (especially misses) greatly affect memory performance (in terms of number of accesses). A memory access is performed (i) in case of a cache miss, to fetch an entire cache line and (ii) when a line needs to be replaced and is marked dirty (has been modified and needs to be written back to the memory). Hence the number of accesses to the memory can be computed as $accesses = misses + writebacks$. Because we are not modifying the line size when reconfiguring the cache, energy consumption for a single memory access is constant E_{access}. With this, the memory energy consumption is given by the formula $E_{memory,k}(S, A) = accesses_k(S, A) \times E_{access}$. By including the memory energy, the optimal operating point may change, as shown by the total energy consumption graph, as a function of cache parameters. Our results are along the line suggested in previous work [30].

We should mention that while current processors (including the ARM core on iPaq) in general do not exhibit such aggressive architectural reconfigurations, this is an active area of research and we anticipate significant advances to be made in the near future. Nevertheless, the processors on iPaq and other mobile devices have on-chip caches, which, when optimized for a particular video stream, can bring important benefits in power savings and performance due to the stream-like characteristics of the MPEG video. Our hardware does not allow us to implement this feature into our cross layer system.

IV. System Implementation & Performance Evaluation

We have designed and implemented a prototype of the DYNAMO framework on a Compaq iPAQ 3650 (running the Linux 2.4.18-rmk3 kernel), with a 206Mhz Intel StrongArm processor, with 16MB ROM, 32MB SDRAM. The iPAQ used a Cisco 350 Series Aironet 11Mbps wireless PCMCIA network interface card for communication. We used a Linux desktop with a 1GHz processor and 512MB RAM as our proxy server.

Fig. 2 shows the various implemented components of our cross-layer framework. The global adaptor is the entity that monitors global and local state changes and is responsible for dynamic adaptations in the system. The intelligence of the middleware is built into this module. It is implemented as an active runtime component (thread). The energy/system

monitors are middleware libraries that export calls to provide applications (and middleware modules) with local state information. The monitor interface has been implemented on Linux using the "*/proc*" interface. API's for manipulating and querying the network interface and the backlight are implemented as low level "ioctl" system calls compiled into libraries. The communication manager implements a simple communication protocol using UDP for the distributed middleware on the various systems (e.g. device and proxy) to communicate state and control information.

In the power-aware operating system we implement the dynamic frequency scaling of the CPU. The linux kernel scheduler is modified to implement a power aware scheduler based on the rate-monotonic criterion. We implement a set of API calls that enable the middleware/applications to set application (thread) execution parameters for dynamic adaptation and retrieve information from the scheduler. All threads created with our OS API calls are created as POSIX threads within the Linux kernel. Since we want to use time driven constraints to help scheduling the threads (used interchangeably with "tasks"), we associate the threads with the SCHED_FIFO scheduling policy, which gives them higher priority than all the other tasks in the system which are not created using this scheduling policy.

Performance Evaluation

In this section, we present the performance evaluation of our cross-layer framework. Our results establish that cross-layer adaptations can be used to accomplish good energy savings for a video streaming application on mobile handheld devices, resulting in higher quality streams and eventually higher user satisfaction. We also demonstrate through simulation results, that current hardware limitations impose restrictions on realizing a framework that can fully exploit the benefits of cross-layer adaptation. We discuss some of these existing limitations and explain how some of these issues can be addressed in future systems.

A. Experimental Methodology

We performed our experiments on a Compaq iPaq 3650 running Familiar linux 2.4.18-rmk3 kernel. The batteries were removed from the iPAQ during the experiments. We used a National Instruments PCI DAQ board to sample voltage drops across a resistor and the iPAQ, and sampled the voltages at 200K samples/sec. We calculated the instantaneous and average power consumption of the iPAQ using the formula $\boldsymbol{P_{iPAQ} = \frac{V_R}{R} \times V_{iPAQ}}$ (Fig 12). The proxy in our experiments is a Linux desktop with a 1GHz processor and 512MB of RAM. For our profiling we use several video streams with various levels of action such as *e.g. grand theft auto & michael jordan to the max* (high action), medium action and low action (e.g. news) clips.

We implemented the following components to experiment with our framework: (i) *Video Streaming Application with cross-layer awareness* (ii) *H.263 Video Streaming Server with a realtime transcoding capability*. The input image sequence used is *foreman.cif* with 300 frames replayed as a continuous stream by the server. The proxy employs a H.263 encoder developed in house is used for dynamic video streaming and realtime transcoding. The encoder supports 31 discrete quality levels as various video quantization values. All the quality levels encode at 20 frames per second. For sensing network noise, the proxy currently uses the RSSI (receive signal strength indicator) and the link quality parameters available through the cisco driver interfaces. These values are used at the proxy to control the buffering of the adaptive video transmission. We used several video sequences in our experiments but we report the results from a single video stream (foreman) due to space constraints.

[2]We report our performance results for a Compaq iPaq 3650 in this section. Note that the reported results might vary for other handheld systems, but the overall approach for the experimental methodology and evaluation for any other mobile system would roughly be the same

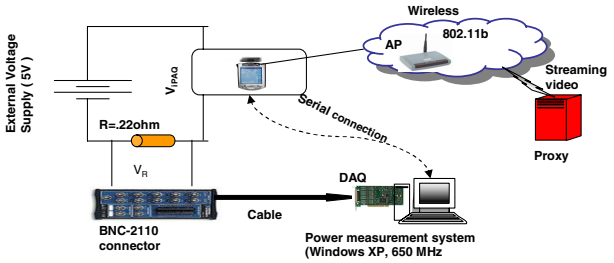

Fig. 12. Setup for Power Profiling.

B. System Overhead

In our first set of experiments, we analyze the overhead of the distributed middleware adaptations and the timing overheads of the implemented API calls.

Middleware Overheads: We measured the timing overheads of the middleware API calls that enabled the communication between the applications, middleware and the operating system. The highest penalties are incurred in using the shared memory and synchronization (semaphore) functions. Fig 13 presents the overheads of using these functions within our framework in microseconds. Fig 14 presents the timing overheads (max, min and average) involved in the various low-level querying calls. Clearly, querying the network and battery status take more time than others, with an average time overhead of about 8ms. Note that these calls are not made very frequently (typically in the order of seconds/mins) and therefore the overheads are amortized over time.

OS-API Overheads: Table 16 shows the average overheads in microseconds for our OS level API. Note that the frequency switching overhead for the IPAQ is extremely high (28 milliseconds), making it infeasible to implement DVS for every instance of the task. The first two PAAPI calls are invoked before the tasks start executing. The calls paapi_status_start and paapi_status_done are invoked by the PAAPI threads once per job instance. As shown, the overheads are pretty low, and therefore do not affect the performance of the system.

Transcoding only Energy savings: Fig 15 shows the normalized energy saved (for the FOREMAN video) on the mobile device just by performing dynamic video transcoding at the proxy using our H.2363 player and transcoder. As much as

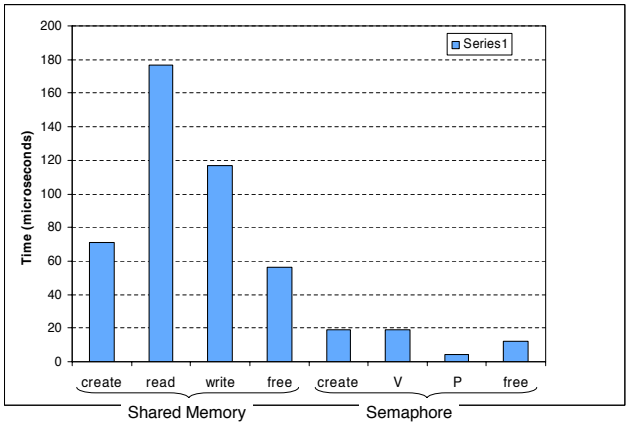

Fig. 13. Middleware overhead.

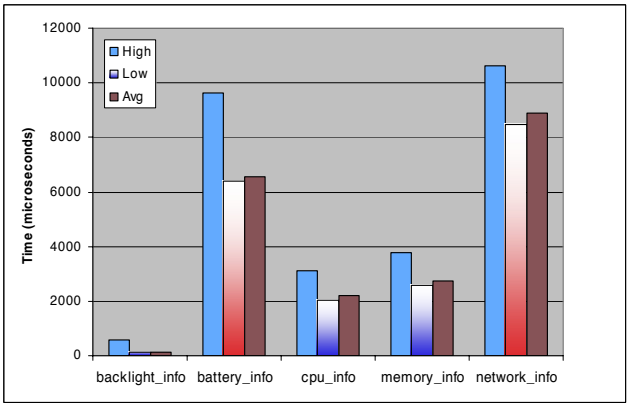

Fig. 14. System Query.

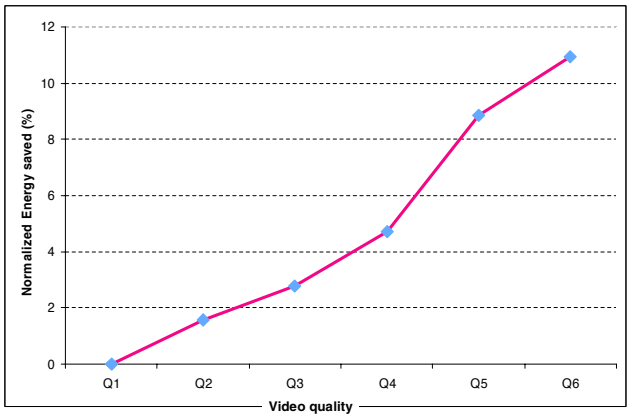

Fig. 15. Energy Savings: Transcoding only.

Function name	Overhead (μsec)
Frequency change time	28187
paapi_create_type	7
paapi_create_instance	4595
paapi_leave	12
paapi_status_start	6
paapi_status_done	10
paapi_change_type	56

Fig. 16. Table: OS API Overheads.

11% of energy can be saved over the highest video quality using dynamic transcoding. Table I shows similar results for the grand theft auto action video clip.

C. Network Card Energy Savings using End-to-End Adaptation

We evaluated the energy savings achieved using our middleware based adaptations involving intelligent traffic shaping and the energy-aware transcoding. While transcoding by itself results in energy savings, it controls the video quality which in turn affects the amount of video that can be buffered at the proxy and device. We program the middleware on the device to send feedback back to the proxy once every 5 seconds. The proxy then uses the RSSI indicator and the link quality (used as indicators for noise on a 802.11b network) of the network to dynamically control the video quality and the video buffering. With the knowledge of the decoding capacity of the device (20 fps for our device), the proxy is able to calculate the number of seconds of video that it buffers. This is sent as control information to the device before transmitting the buffered video to the device. The network adaptation module uses the cross-layer API to set the appropriate network card power management policy. The maximum buffering allowed by our device is 10KB, so the proxy uses buffer sizes from 1KB-10KB. We use fixed values for our video quality transcoding (using 5 different qualities with quantization values 1, 5, 10, 15 and 20). To synchronize the device and the proxy we wake the device 50ms before the proxy sends its next transmission and we do not turn off the device radio if the sleep duration is less that 50ms.

Figure 17 shows the adaptation of the network achieved by our framework for the FOREMAN video for a one minute period. Figure 18 shows the video frames lost due to network adaptation over the same period. As seen from the graph, the number of frames lost with the power management of the network interface is similar to the number of frames lost with the network card always on. Figure 19 shows the energy saved due to the network power optimization during the same period. The energy savings from network card optimization was found to be above 50% for the FOREMAN stream.

D. Backlight Energy Savings

In our experiments, we used a video sequence captured from a broadcasted *ABC_news* program, whose first frame is shown in Figure 8. We choose this video as representative of a typical usage of a PDA. In Figure 20, we show the basic statistics (i.e., the mean and the variance of luminance per frame) of this video.

We assume that the users are given three quality options, fair, good, and excellent, which respectively correspond to the PSNR (Peak Signal to Noise Ratio) value of 30dB, 35dB, and 40dB. After applying our algorithm as defined in [9], we obtain the adapted backlight level for these three quality preferences, as is shown in Figure 21. It can be seen that

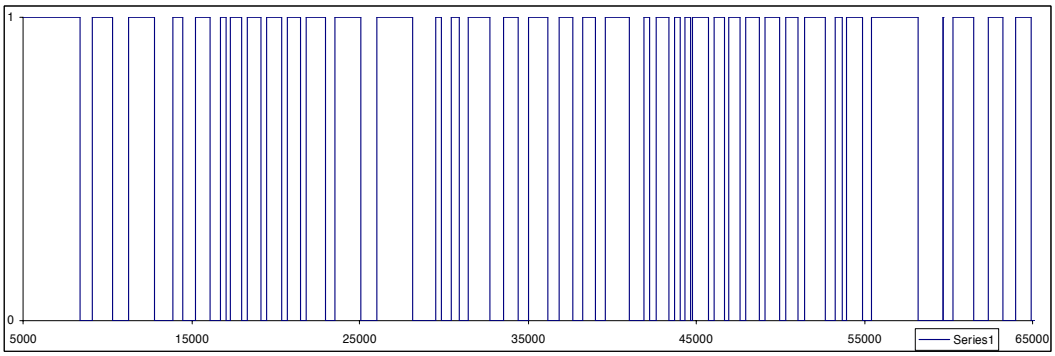

Fig. 17. On/Off phases of the network adapter over a one minute period.

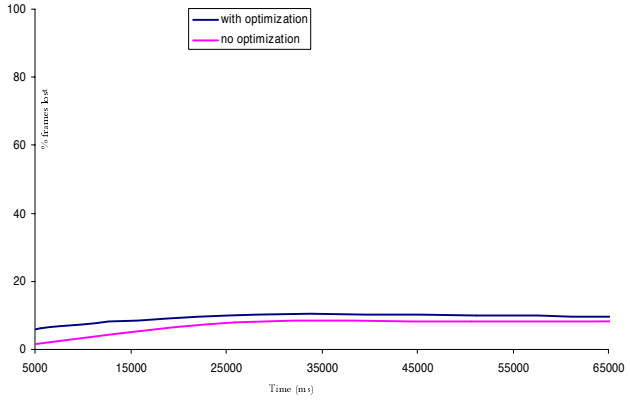

Fig. 18. Number of Frames lost with and without adaptation.

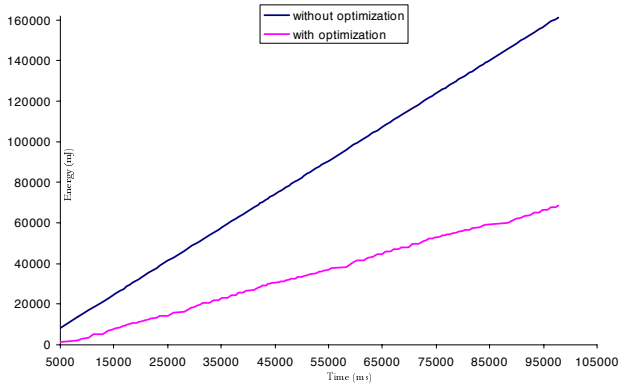

Fig. 19. Network energy consumed with and without adaptation.

TABLE II

OPTIMAL NETWORK VIDEO BURST LENGTHS(IN SECS) AND CORRESPONDING POWER GAINS FOR DIFFERENT QUALITY AND NOISE LEVELS FOR THE GRAND THEFT AUTO ACTION VIDEO, ASSUMING SUFFICIENT BUFFER AVAILABLE AT CLIENT AND NETWORK PACKET SIZE OF 500KB.

Quality Level	Burst Length (secs)(N=1)	Power Saved Watts (N=1)	Burst Length (secs) (N=3)	Power Saved Watts (N=3)	Burst Length (secs) (N=5)	Power Saved Watts (N=5)
Q1	2.3	.925	2.0	0.89	1.8	0.87
Q2	3.5	1.0	3.05	0.98	2.76	0.96
Q3	4.6	1.04	4.05	1.02	3.68	1.0
Q4	4.85	1.05	4.2	1.03	3.85	1.02
Q5	6.8	1.08	6.25	1.07	5.75	1.06
Q6	14.5	1.12	12.5	1.11	11.5	1.11
Q7	17.5	1.13	15.0	1.12	13.5	1.11
Q8	17.0	1.12	15.4	1.12	14.0	1.11

higher video quality needs higher backlight level on average. In Figure 22, we show backlight level before and after the backlight smoothing process. It is seen that the small variation and the abrupt change of the backlight switching are significantly eliminated after the filtering and quantization. In addition, as we expected, the backlight switching mostly happens at the boundary of major scenes.

In Table III, we summarize the results of our quality aware backlight adaptations (QABS). The mean backlight level of different quality preferences produces a quality on average very close to the pre-determined quality threshold. It is noted that different quality requirements result in various power saving gains. Higher quality preference must be traded using more backlight energy. Nevertheless, we can still save 29% energy that is supposed to be consumed by the backlight unit if we set the quality preference to be "Excellent".

In Figure 23, we show that the filtering and quantization process may lead to instantaneous quality fluctuation, which is contrasted to the consistent quality before backlight smoothing. Nevertheless, we observe that the quality fluctuation is around the designated quality threshold and mostly happens at scene changes.

We compare the actual energy consumption on a Compaq ipaq with and without our quality aware backlight adaptation. The energy savings from our backlight adaptation for the ABCNews clip is 35%-40% of the total energy consumed due

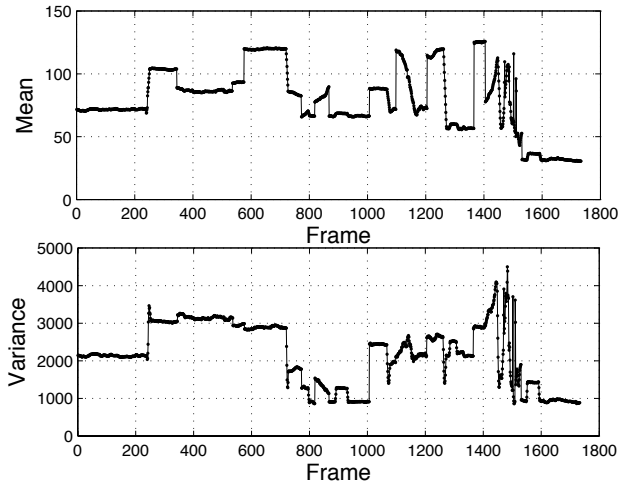

Fig. 20. Luminance per frame - *abc_news*.

to backlight. Even for videos that offer very little opportunity to aggressively perform backlight adaptation (e.g. foreman video clip, which is simply a talking head), we can achieve energy savings as high as 14-20% without sacrificing video quality.

1) CPU Energy Savings: Simulation Results: The CPU architecture simulation was implemented using the Wattch/SimpleScalar [6] power simulator. We configured our simulated CPU to resemble a typical Intel XScale processor (widely used in today's mobile devices, mostly due to their excellent MIPS/Watt performance): ARM core, 400 MHz, 1.3V, 0.18um, 32k instruction cache, 32k data cache, single issue. The Wattch simulator provides a series of statistics (e.g. cache power consumption) at the end of the simulation. The MPEG decoder from Berkeley MPEG Tools was used for decoding the video streams. Video transcoding was done using the commercially available TMPGEnc transcoder. As input video for the decoding, we used traces from various video clips from low action(e.g. "news") like content to high action (e.g. GTA) fast scene changing streams. For each of these clips, we extract a sequence to be simulated and we encode it at noticeable different levels of quality (Table I). The decoding program is then simulated through Wattch and statistics are extracted from the simulator output. The options for the decoder were chosen so as to optimize the performance (e.g. X11 screen output was disabled in the decoder). Table IV shows the optimized settings for the eight video quality levels using our simulations. Figure 24 and Figure 25 show the exhaustive search space for optimizing energy of the data caches.

E. Energy Savings with Joint Adaptation

Table V presents a comparison of the total energy consumed by an iPAQ over a one minute period with and without the optimizations for the cpu, network and the backlight. As seen from the table, simple transcoding results in very poor energy savings. However, transcoding combined with the cross-layer approach can result in significant energy savings (compare Q1 without any adaptation with Q8 with cross-layer adaptations).

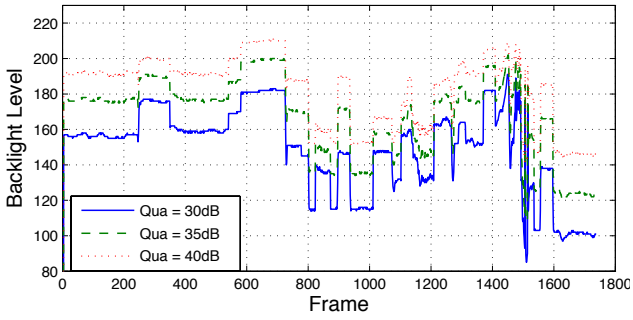

Fig. 21. Backlight level adapted to three given quality thresholds.

Figure 26 shows the energy consumption of the three most power hungry components for a Compaq iPaq 3600 without any power optimization schemes. Figure 27 shows the corresponding power savings achieved by using our cross-layer power saving approach for each of these components. As can be seen from the figures, almost 50% of the energy consumed by these components can be saved for a medium action video clip.

V. Related Work & Conclusions

One of the primary motivations for our cross layer approach was our realization that advances in battery and hardware technology cannot, by themselves, meet the energy demands of next generation mobile computers, i.e. higher levels of the system must also be involved [14], [17], [16]. There have been several research efforts to dynamically modify application behavior

dynamically to conserve energy, with some help from the operating system. Flinn and Satyanarayanan, demonstrated that a collaborative relationship between the operating system and applications can be used to meet user-specified goals for battery duration. Their application level adaptations were guided by operating system monitors that provided feedback on energy supply and demand [18]. They implemented and validated their results on the Odyssey platform for mobile computing [26].

The Milly Watt project [14] explores the design of a power-based API that allows a partnership between applications and the system in setting energy use policy. In the context of this project, a Currentcy model that unifies energy accounting over diverse hardware components and enables fair allocation of available energy among applications [39], and a prototype energy-centric operating system, ECOSystem, that implements explicit energy management techniques from the system point of view have been proposed [38]. Their goal is to extend battery lifetime by limiting the average discharge rate and to share this limited resource among competing tasks according to user preferences. PowerScope [18] is an interesting tool that maps energy consumption to program structure. It first profiles the power consumption and system activity of a computer and then generates an energy profile from this data. Puppeteer [16] presents a middleware framework that uses transcoding to achieve energy gains. Using the well defined interface of applications, the framework presents a distilled version of the application to the user, in order to draw energy gains.

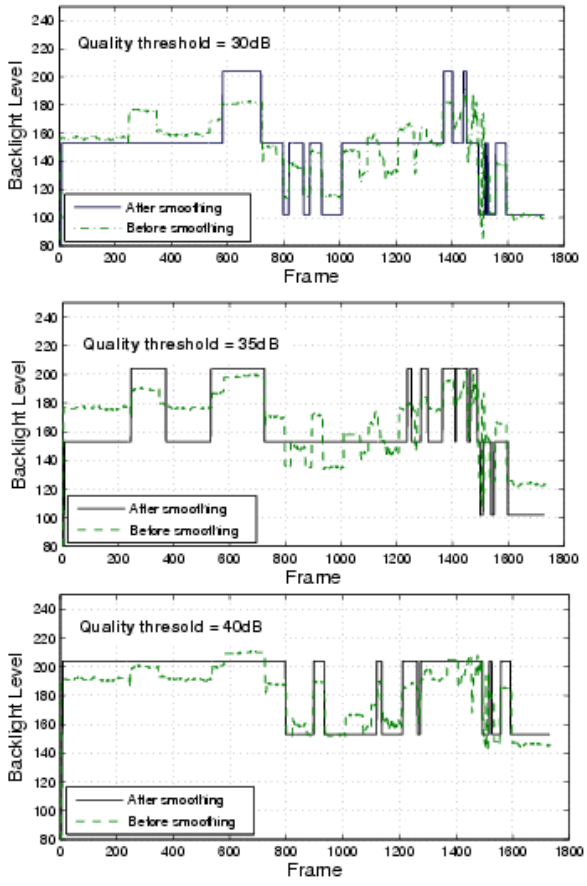

Fig. 22. Backlight level before and after filtering and quantization.

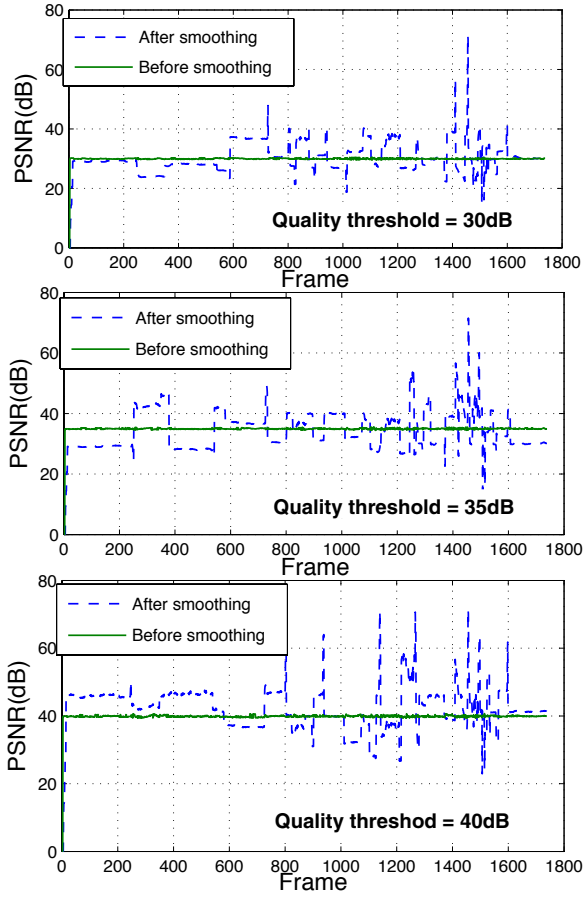

Fig. 23. Quality before and after backlight smoothing.

TABLE III

RESULTS OF BACKLIGHT SCALING.

Alfa Mean			Quality(dB)			Power Saving(%)		
Fair	Good	Excellent	Fair	Good	Excellent	Fair	Good	Excellent
149	162	186	30.17	34.28	42.31	41.8%	36.7%	27.3%

TABLE IV

ARCHITECTURAL CONFIGURATIONS FOR IDEAL ENERGY AND PERFORMANCE GAINS.

Video Quality	Cache Size	Cache Associativity	Clock Frequency	Voltage	Original Energy	Optimized Energy	Savings
Q1	8	8	100	1	1.29	0.76	47.5%
Q2	8	8	100	1	1.09	0.64	47.8%
Q3	8	8	100	1	0.95	0.56	48.0%
Q4	32	2	66	0.9	0.54	0.26	57.6%
Q5	32	2	66	0.9	0.48	0.23	57.8%
Q6	32	2	33	0.9	0.42	0.20	58.0%
Q7	8	8	33	0.9	0.29	0.14	57.3%
Q8	8	8	33	0.9	0.24	0.11	57.5%

TABLE V

ENERGY SAVINGS WITH AND WITHOUT CROSS LAYER ADAPTATION (FOREMAN VIDEO).

Quality	Adaptation	CPU (J)	Network (J)	Backlight (J)	Total (J)	% Savings
Q1	No	122.4	90.6	144	364.2	
Q1	Yes	72.216	53.8	104.688	237.9	34.78
Q4	No	90.6	90.6	144	348.6	
Q4	Yes	48.1	43.3	87.84	186.44	46.52
Q8	No	87.6	90.6	144	328.8	
Q8	Yes	37.23	29.7	79.2	153.33	53.37

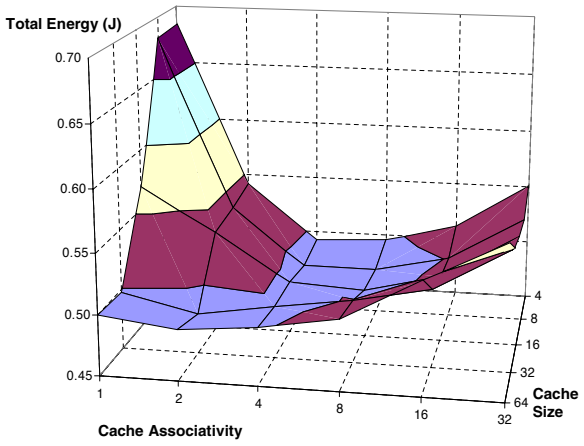

Fig. 24. Search Space for Medium Qual. *Action* Clip, no DVS.

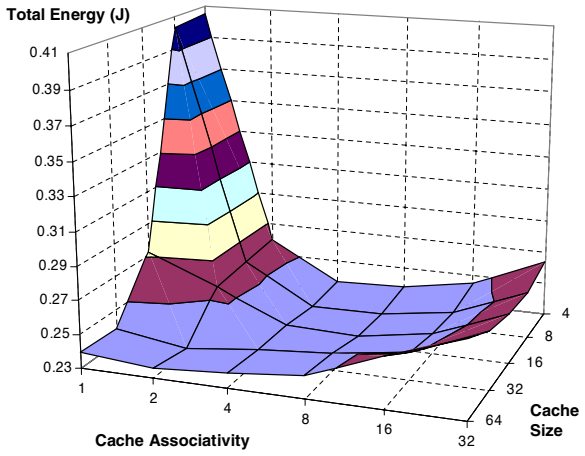

Fig. 25. Search Space for Medium Quality *Action* Clip, with DVS

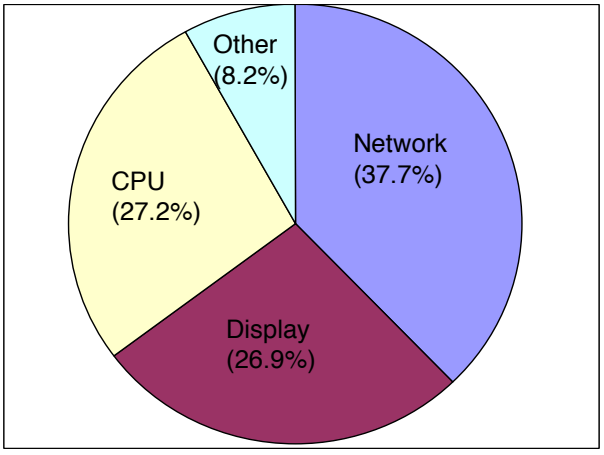

Fig. 26. Energy distribution without optimizations.

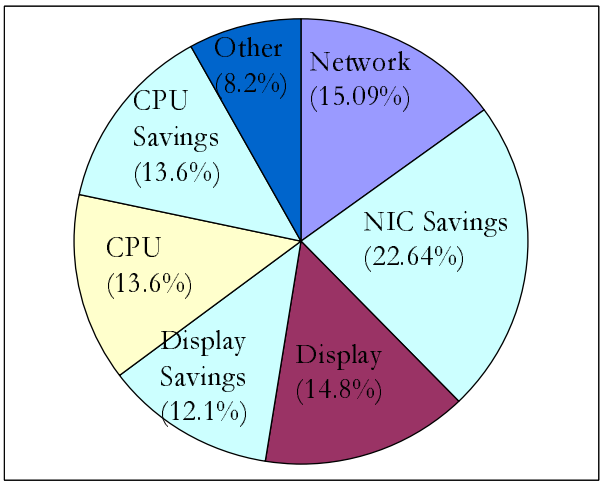

Fig. 27. Energy distribution with cross-layer optimizations.

Hughes et. al. [20] have analyzed architectural variability in the frame-level execution time for a number of multimedia applications on general-purpose architectures. [21] proposes a technique for combining two hardware adaptations (architecture adaptation and dynamic voltage scaling) for reducing energy in multimedia workloads. The study concludes that DVS alone gives most of the energy benefits, while the more aggressive architectures are more energy efficient in the presence of DVS. Most other research efforts at the hardware level, have been focussed on techniques like dynamic voltage scaling (DVS) [33], [27], [34], [22], and dynamic power management (DPM) [12], [13].

At the application and middleware levels, the primary focus has been to optimize network interface power consumption [15], [7], [8]. A thorough analysis of power consumption of wireless network interfaces has been presented in [15]. Chandra et al. [7] have explored the wireless network energy consumption of streaming video formats like Windows Media, Real media and Apple Quick Time. In [8], they have explored the effectiveness of energy aware traffic shaping closer to a mobile client. In [29], Shenoy suggests performing power friendly proxy based video transformations to reduce video quality in real-time for energy savings. They also suggest an intelligent network streaming strategy for saving power on the network interface. We have a similar approach, but we model a noisy channel. Caching streams of multiple qualities for efficient performance has been suggested in [17]. In [36], a resource aware admission control and adaptation is suggested for multimedia applications for optimal CPU gains. In [2] the authors propose a server controlled power management scheme much like ours where the server exploits the workload, traffic related information and feedback from the mobile device to minimize the energy consumption of the network interface card.

In the GRACE project [3], [37], [35] both global and local coordinators exist within the mobile device. In contrast, we employ a server to perform adaptations remotely and an end-to-end approach to enable the device to adapt to changes made at the server. Our work is similar in spirit to the work in [5], [4], wherein applications provide hints to OS level power managers for implementing power management strategies. In DYNAMO we adopt the strategy of exchanging state information dynamically at all system levels and driving simultaneous adaptations at each layer based on these exchanges.

VI. Conclusions & Future Work

In this paper, we have used a distributed middleware framework to jointly adapt power optimization strategies across various levels of system hierarchy for enhancing the user experience for streaming video onto handheld computers. We implemented a system for such cross-layer adaptation on a Compaq iPaq running Linux and concluded that current hardware limitations precluded us from achieving the full benefits of the proposed cross-layer adaptation scheme. We showed using simulation studies that noticeable improvements in the requested video stream quality, enhancing the user experience substantially.

Future Directions & Suggestions for Next Generation Systems: We are currently exploring further architectural and middleware adaptations for improving the power consumption of displays, storage devices etc. and integrating them into the framework. Another exciting future area is to study our architectural optimizations in the presence of multiple concurrent applications on the portable device. Clearly some of the techniques presented in our work have to be significantly enhanced/modified for operating in the presence of multiple concurrent applications.

We now discuss some of the insights we have gained from our experience with developing a cross-layer adaptation framework for mobile systems. Our access to the linux kernel made it particularly easy for us to probe energy information and manipulate hardware settings for the cpu, network and LCD backlight. However these values are not currently exposed in most portable systems making it a significant limitation for cross-layer power management in off-the-shelf systems running proprietary operating systems. Similarly hardware for next generation portable systems need to support some degree of hardware reconfigurability. Gains from dynamic voltage scaling can be very substantial if technology would allow fast frequency switching and provide a larger number of operating voltage/frequency states. We strongly believe that continuous feedback from applications can be very effective in performing dynamic adaptations at the operating system and lower levels. We suggest the next generation power aware operating systems allow hooks for applications to provide information (hints) regarding network accesses, computing deadlines and disk accesses.

We are noticing a growing trend in the exposing of system level knobs to applications and middleware through well defined API interfaces. Energy management solutions in existing video streaming frameworks continue to be "device centric" and therefore are very reactive in their approach. We contend that infrastructure support that provides mobile systems with additional end-to-end state information can result in proactive adaptations leading to much higher energy savings.

Acknowledgements

We would like to thank Cristiano Periera, Hyunok Oh, Liang Cheng, Michael Philpott and Radu Cornea for their help in building the various components of the DYNAMO system.

References

[1] http://dynamo.ics.uci.edu.

[2] A. Acquaviva1, T. Simunic, V. Deolalikar, and S. Roy. Server Controlled Power Management for Wireless Portable Devices. In *HP Labs, Technical Report*, 2003.

[3] S. V. Adve, K. Nahrstedt, D. Jones, and et. al. The illinois grace project: Global resource adaptation through cooperation. In *SHAMAN-02*.

[4] M. Anand, E. B. Nightingale, and J. Flinn. "Self-tuning wireless network power management". In *MOBICOM*, 2003.

[5] M. Anand, E. B. Nightingale, and J. Flinn. "Ghosts in the machine: Interfaces for better power management". In *MOBISYS*, 2004.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, June 2000.

[7] S. Chandra. Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats. In *MMCN-02*.

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

[9] L. Cheng, S. Bossi, S. Mohapatra, M. E. Zarki, N. Venkatasubramanian, and N. Dutt. Quality Adapted Backlight Scaling (QABS) for Video Streaming to Mobile Handheld Devices. In *IEEE/IEE/LNCS 4th International Conference on Networking*, Apr 2005.

[10] W. chi Feng and S. Sechrest. Improving data caching for software mpeg video decompression. In *IS&T/SPIE Digital Video Compresssion: Algorithms and Technologies*, 1996.

[11] K. Choi, K. Dantu, W.-C. Chen, and M. Pedram. Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder. In *ICCAD 2000*, 2002.

[12] F. Douglis, P. Krishnan, and B. Bershad. "Adaptive disk spin-down policies for mobile computers". In *2nd USENIX Symposium on Mobile and Location-Independent Computing*, April 1995.

[13] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the power hungry disk. In *WINTER USENIX conference*, January 1994.

[14] C. Ellis. "The case for higher level power management". In *In Proceedings of HotOS*, March 1999.

[15] L. Feeney and M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an ad hoc Networking Environment. In *IEEE Infocom*, April 2001.

[16] J. Flinn, E. de Lara, M. Satyanarayanan, D. S. Wallach, and W. Zwaenepoel. "Reducing the energy usage of office applications". In *IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[17] J. Flinn and M. Satyanarayanan. Energy-Aware Adaptations for Mobile Applications. In *SOSP*.

[18] J. Flinn and M. Satyanarayanan. PowerScope: a tool for profiling the energy usage of mobile applications. In *Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[19] P. J. M. Havinga. *Mobile Multimedia Systems*. PhD thesis, University of Twente, Feb 2000.

[20] C. J. Hughes, P. Kaul, S. Adve, R. Jain, C. Park, and J. Srinivasan. "variability in the execution of multimedia applications and implications for general-purpose architectures". In *ISCA*, 2001.

[21] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *MICRO*, 2001.

[22] P. Kumar and M. Srivastava. Predictive Strategies for Low-Power RTOS Scheduling. In *ICCD*, 2000.

[23] M. Mesarina and Y. Turner. A Reduced Energy Decoding of MPEG Streams. In *MMCN*, January 2002.

[24] S. Mohapatra, R. Cornea, and et.al. "Integrated power management for video streaming to mobile handheld devices". In *ACM Multimedia*, 2003.

[25] S. Mohapatra and N. Venkatasubramanian. "PARM: Power-Aware Reconfigurable Middleware". In *ICDCS-23*, 2003.

[26] B. D. Noble, M. Satyanarayanan, D.Narayanan, J.E.Tilton, and J. Flinn. Agile Application-Aware Adaptation for Mobility. In *SOSP*, October 1997.

[27] P. Pillai and K. G. Shin. "Real-time dynamic voltage scaling for low-power embedded operating systems". In *In Proc. of the 18th ACM Symp. on Operating Systems Principles*, 2001.

[28] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority rt-systems. In *RTAS*, 2004.

[29] P. Shenoy and P. Radkov. Proxy-Assisted Power-Friendly Streaming to Mobile Devices. In *MMCN*, 2003.

[30] P. Soderquist and M. Leeser. Optimizing the data cache performance of a software MPEG-2 video decoder. In *ACM Multimedia*, pages 291–301, 1997.
[31] M. Stemm and R. Katz. Measuring and Reducing energy consumption of network interfaces in hand-held devices. In *IEICE*, August 1997.
[32] M. Tamai, K. Yasumoto, N. Shibata, and M. Ito. Low Power Video Streaming for PDAs. In *Workshop on Mobile Multimedia Communications*, 2003.
[33] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *In Symposium on Operating Systems Design and Implementation*, 1994.
[34] Y.Shin, K. Choi, and T. Sakurai. "Power optimization of real-time embedded systems on variable speed processors". In *CAD*, 2000.
[35] W. Yuan and K. Nahrstedt. Process Group Management in Cross-Layer Adaptation. In *MMCN-04*.
[36] W. Yuan and K. Nahrstedt. A Middleware Framework Coordinating Processor/Power Resource Management for Multimedia Applications. In *IEEE Globecom*, Nov 2001.
[37] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems. In *MMCN-03*.
[38] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. "Ecosystem: Managing energy as a first class operating system resource". In *ASPLOS-02*.
[39] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: Unifying policies for resource management. In *Usenix Annual Technical Conference*, 2003.

Shivajit Mohapatra is currently a senior researcher at the Applications Research Center at Motorola Labs. He received his PhD. from the University of California, Irvine in 2005. His current research focus is the areas of Ad-Hoc and Mobile Computing. He is also interested in the areas of Game Theory, Incentive Models and Security for mobile environments. In the past, his focus areas have been Energy-Aware Computing using Cross-Layer system design, distributed middleware systems and Multimedia.

Nikil D. Dutt (SM) received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1989, and is currently a Chancellor's Professor at the University of California, Irvine, with academic appointments in the CS and EECS departments. His research interests are in embedded systems design automation, computer architecture, optimizing compilers, system specification techniques, and distributed embedded systems. He received best paper awards at CHDL89, CHDL91, VLSIDesign2003, CODES+ISSS 2003, CNCC 2006,and ASPDAC-2006. Professor Dutt currently serves as Editor-in-Chief of ACM Transactions on Design Automation of Electronic Systems (TODAES) and as Associate Editor of ACM Transactions on Embedded Computer Systems (TECS). He was an ACM SIGDA Distinguished Lecturer during 2001-2002, and an IEEE Computer Society Distinguished Visitor for 2003-2005. He has served on the steering, organizing, and program committees of several premier CAD and Embedded System Design conferences and workshops, including ASPDAC, CASES, CODES+ISSS, DATE, ICCAD, ISLPED and LCTES. He is a senior member of the IEEE, serves or has served on the advisory boards of ACM SIGBED and ACM SIGDA, and is Vice-Chair of IFIP WG 10.5.

Alex Nicolau received the Ph.D. degree in computer science from Yale University, New Haven, CT, in 1984. He is a Professor with the Center for Embedded Computer Systems, University of California, Irvine, with academic appointments in the Information and Computer Science and Electrical Engineering Departments. His current research interests include embedded systems, computer architecture, and optimizing compilers. He is the coauthor of several books and has published over 250 journal and conference papers in the above areas. He serves as Editor-in-Chief of the International Journal of Parallel Programming and has served on the program and steering committees of numerous conferences in the field, notably ICS, PACT, and Micro.

Nalini Venkatasubramanian is currently an Associate Professor in Computer Science at the University of California, Irvine.