Co-operative Energy-Constrained Resource Management in Mobile Multimedia Environments

Samarth H. Shah, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
Email: {shshah, klara}@cs.uiuc.edu

Wanghong Yuan
DoCoMo USA Labs
181 Metro Drive, Suite 300
San Jose, CA 95110
Email: yuan@docomolabs-usa.com

May 3, 2005

Abstract

Traditionally, the problem of minimizing energy consumption in mobile wireless devices has been addressed independently for various system components such as the CPU (via process management), network interface (via traffic management), I/O devices, etc. In this paper, we explore an integrated system architecture, consisting of co-operative process and network management, to perform multimedia streaming over a wireless LAN shared by several energy-constrained mobile devices. Co-operation among the system resource components is required so that they jointly yield the same operating QoS level for the application, under the device energy constraints. We maximize aggregate utility of all applications in a wireless LAN, while still meeting the system lifetime (i.e. energy usage), bandwidth usage, and computational capacity constraints of each mobile device. The key to co-operative energy-constrained process and network management is decoupling CPU-usage and network-usage time-scales. Consequently, both these system components minimize their individual energy consumption while still meeting the overall quality requirements of multimedia applications. Our experiments with a prototype h.263 application illustrate the benefit in terms of system energy-saving of using our scheme (34% energy-saving on the base system and 42% network energy-saving, for a device with a single application running at its highest QoS level), and its impact on application performance. We also demonstrate the flexibility of our architecture and its applicability to various mobile devices with different energy consumption characteristics.

Key terms: 802.11 Wireless LAN, GRACE-OS, Global QoS Manager, Power-Saving Mode, Continuous-Access Mode.

1 Introduction and Motivation

Mobile devices are characterized by their small size, untethered nature, and ability to communicate wirelessly. Consequently, they are hampered by low computing/storage capacity, limited lifetime of operation and low communication bandwidth. Real-time multimedia applications running on these mobile devices pose an especially challenging problem because, in addition to the constraints imposed by the mobile devices and its environment, the applications have stringent Quality of Service (QoS) constraints of their own. These QoS constraints of multimedia applications translate into stringent computing and communication demands on the system. Management of resources such as the CPU, network bandwidth and battery power, and improving QoS of multimedia applications must thus co-operate.

1.1 Scenario

One possible application scenario of multimedia streaming in low-CPU, low-memory, energy-constrained mobile devices over wireless networks is in space exploration. Robots (see Figure 1) equipped with video cameras and assorted sensors emerge from a space shuttle on a remote planet and spread out on the planet's surface. They capture and transmit video and other real-time sensory information over a wireless network back to the shuttle. This information is then relayed from the shuttle back to earth. The robots are battery-powered. The usage of the energy supply must be controlled so that the lifetime of the battery encompasses the duration of the robot's mission. This is one of the scenarios that we primarily address in this work, although our system is generally applicable to a variety of mobile multimedia environments.

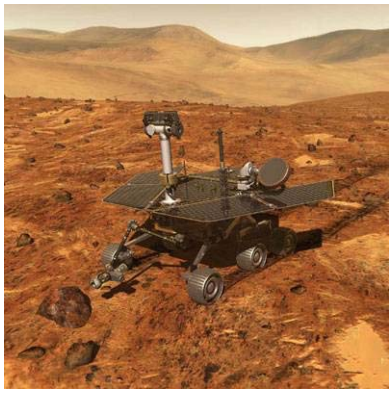

Figure 1: Robot for space exploration.

Figure 2 illustrates the operation of the robots. The figure

shows multiple robots each capturing video and monitoring environmental conditions in real-time. Each robot sends multiple streams of real-time traffic, consisting of the data it is capturing/recording in real-time, to an access-point on the space shuttle, over an IEEE 802.11 wireless LAN.

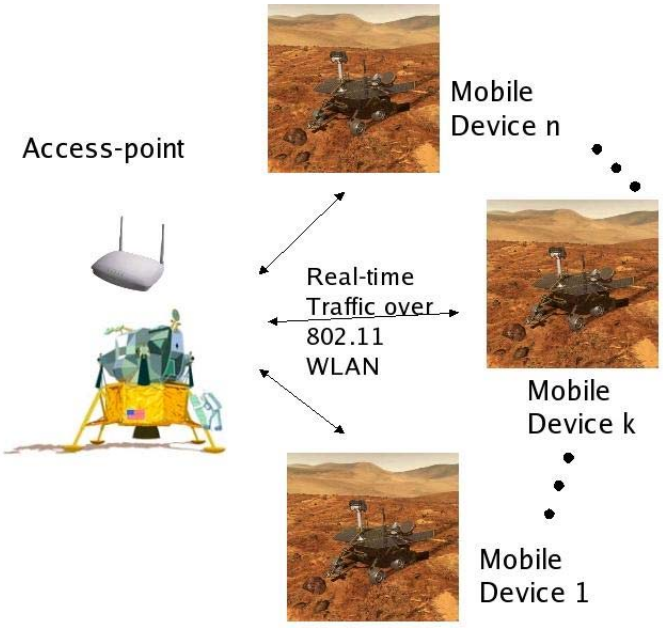

Figure 2: Problem scenario.

Given the above scenario, the problem we address can be summarized as follows: We aim to maximize the quality of all the real-time applications in the wireless LAN, while at the same time not overloading the limited computing resources of the individual devices, not exceeding total network bandwidth and ensuring that energy consumption is low enough for the battery to not expire prematurely.

1.2 Co-operation

Hardware components, e.g. CPU, wireless network interface, I/O devices, designed for mobile devices often have several operating modes that trade-off energy consumption and performance. Energy-efficiency research (see Section 2) has generally concentrated on developing strategies for adapting the operation of a *single* component over its available modes. At best, these strategies take into account the energy consumed by the other components of the system as overhead, while trading off energy consumed and performance of a particular component [4]. Such an approach ignores the *dependency* between energy saving strategies for different hardware components used by a particular application. For example, for a multimedia streaming application using CPU and network interface, the network interface must not adopt a strategy such that only 2fps data rate is possible and energy is saved, while the CPU picks an operating mode such that 3fps data rate is possible and energy is not conserved. Furthermore, even in the rare case where there is interaction between energy saving strategies at different components on the same device [16], *global* interaction across different mobile devices in the network is not considered. The latter is required because the devices may share resources, e.g. network bandwidth. Therefore, the energy saving strategies at both the network interface and the CPU must take into account available bandwidth on the network.

Our architecture *co-ordinates* energy-saving efforts at the CPU and network interface so that both the process-management and the network-management ensure the same end-to-end application quality. In addition, the Global QoS Manager (GQM) in our system arbitrates the resource shared between all applications in the wireless LAN, namely the wireless channel bandwidth. Co-operation between CPU/process-management and network-management is required because there is no point in the CPU operating at a high-performance, high-power mode for maximizing application QoS, when network bandwidth is limited or energy is low. Similarly, there is no point in allocating large bandwidth to an application running at a low QoS level, on an energy- or CPU-constrained mobile device. Our centralized GQM takes all the parameters of all the applications and mobile devices into account when allocating resources to each application, thus ensuring global interaction. The presence of a central command module in our target scenario facilitates such a centralized system management structure. The CPU-usage and network-usage time-scales in our system are *decoupled* because the CPU is shared by the applications local to the mobile device while the network is a global resource. This separation of time-scales is the key feature of our design. Our results show significant energy savings at the processor level as well as network level.

1.3 Outline

The rest of this paper is structured as follows. Section 2 contains a summary of the related research in system design for energy-aware mobile devices. In Section 3 we present models defining the entities in our system architecture. Section 4 describes our overall system and network protocol design, and system adaptation. Section 5 contains our system profiling methodology. Section 6 explains our experiments and presents our results. Finally, Section 7 concludes this paper.

2 Related Work

Energy-efficiency in mobile devices is a very broad area of research. Energy-efficient mechanisms have been designed for each hardware component and each layer of the OSI network protocol stack. In this section, we concentrate only on research related to the specific mechanisms we employ, namely CPU speed adaptation, i.e. Dynamic Voltage Scaling (DVS), and intelligent application-aware network power mode change.

2.1 CPU Energy-efficiency

In general, there are two approaches to reduce CPU energy consumption. The first one is dynamic power management

(DPM) [26], which puts the idle processor into the lower-power sleep state. The second approach is dynamic frequency/voltage scaling (DVS) [27], which lowers the operating speed and voltage of the active processor. DPM, however, is not suitable for our targeted multimedia applications, which access the CPU periodically. As a result, the idle interval in each period is often shorter than the wake-up time (e.g., 160 milliseconds for StrongARM SA-1100 [26]); so the processor cannot be put into the sleep state in the short idle intervals.

Recently, DVS has been investigated in two main areas, general-purpose systems (GP-DVS) and real-time systems (RT-DVS). GP-DVS [29, 28, 30, 22] algorithms heuristically predict the workload based on average CPU utilization. Although they save energy without degrading performance of best-effort applications, they are unsuitable for multimedia applications due to the timing constraint and demand variations of multimedia applications. For example, Grunwald et al. [29] concluded that no heuristic algorithm they examined saves energy without affecting multimedia application performance.

RT-DVS algorithms [27, 31, 5, 6], often integrated with CPU scheduling, derive workload from worst-case CPU demands of real-time applications. That is, they set CPU speed based on the assumption that applications require worst-case CPU resources. Since an application's demand is not always the worst-case, some reclamation techniques have been proposed to reclaim the unused cycles to save more energy [32, 31]. These reclamation techniques first run CPU fast (assuming the worst-case demand) and then decelerate when a job finishes early.

Our work differs from the above DVS work for two reasons. First, we target soft real-time multimedia applications. Our CPU adaptation is based on the statistical CPU demand of multimedia applications. Second, the CPU adaptation is coordinated with network adaptation to save the overall energy of the mobile device, e.g. handheld computer, laptop, or wireless network-enabled media player.

2.2 Energy-efficiency via Network Interface Mode Change

Energy-efficient network mechanisms have been proposed for the MAC protocol (e.g. [9, 10]), and at the network level for multi-hop networks (e.g. [12, 11, 13, 14]).

An early work in intelligent application-aware switching between the high-performance and low-power modes of a wireless network interface was proposed by Kravets et al [17]. They studied the trade-off between packet delay and energy-saving via energy saving at the network interface. The Bounded Slow Down (BSD) system proposed in [8] for network energy saving for wireless web access application over TCP. In [4], Anand et al proposed Self-Tuning Power Management (STPM), wherein applications provided hints to the STPM module which then took intelligent mode change decisions depending on the traffic estimate. While the authors mention that the base power of the computer, and the impact of performance deterioration on base power, must be taken into account in making network interface power saving decisions, they do not utilize CPU speed adaptation for base power saving. Furthermore, they do not co-ordinate the network interface mode changes between multiple mobile devices. In [7], the authors add a low-power radio to a mobile device for out-of-band signaling that configures the operation of the main 802.11 wireless interface over its various power modes. In [15], the authors propose remote power control of the network card from a central server that makes its decisions based on feedback from the clients. While the authors mention that their scheme can also be used to control CPU energy consumption, exact details of CPU power saving, and co-operation between CPU and network energy management are not specified. Flinn et al [18] proposed a profiling-based scheme for collaboration between OS and application to meet user specified battery-lifetime goals. In all the above works, network energy saving strategies work independent of CPU speed adaptation. At best, the base energy is regarded as an overhead in the network energy saving strategy.

The work closest to our research is [16], in which the authors propose co-operative DVS and network mode change for energy saving using a video streaming application. However, their architecture is more *vertical* than *horizontal*. While they have the application, middleware, OS, and network entities at a mobile device co-operate for energy-saving, and adapt to dynamic network conditions, they do not address conflicts between mobile devices for globally shared resources, e.g., network bandwidth. Furthermore, they use a power simulator rather than a real implementation of their system. Our system maximizes aggregate utility of all applications in the network, while taking into account both energy constraints as well as constraints in the usage of resources shared by various applications in the wireless LAN.

3 Models and Assumptions

In this section, we first present the operating parameters of the various entities in our overall system, and explain our assumptions. We present the following system models: (a) video streaming application model, (b) computing model in mobile devices, and (c) network model over 802.11. We then describe the co-operative adaptation among the modelled entities.

3.1 Application Model

Our system primarily targets periodic, soft real-time networked applications such as multimedia streams and real-time sensory information distribution. If there are any non-real-time or standalone tasks at the mobile device, they are aggregated into a single real-time task. We assume in our system that these non-real-time (best-effort) or standalone tasks are of lesser priority than the real-time, networked applications. We then allocate CPU and network resources to each individual real-time task in the network, and adapt its operation to conform to the allocated resources.

Each real-time task is characterized by the following parameters: (a) its QoS levels q_{ij}, (b) its utility at each QoS level $u_{ij}(q_{ij})$, (c) period $P_{ij}(q_{ij})$, (d) CPU cycles $C_{ij}(q_{ij})$ per period $P_{ij}(q_{ij})$ at each QoS level, (e) network bandwidth consumed $B_{ij}(q_{ij})$ per period $P_{ij}(q_{ij})$ at each QoS level, and (f) applica-

tion buffer size. These values are obtained via profiling of the application, described in Section 5.1.

Note that we will concentrate on real-time sensing or surveillance-like applications, where the sender (encoder) is a mobile device capturing/encoding multimedia (sensory) content and streaming it to a resource-rich infrastructure. This means that the sender portion of the application must secure enough resources to perform its capturing/encoding function, while the receiver is assumed to have sufficient resources, i.e. it is not part of the resource-constrained environment.

3.2 Mobile Device Model

We split the mobile device computational model into the hardware model and the process management model.

3.2.1 Hardware Model

We assume in each mobile device a CPU that supports multiple discrete speeds (frequencies) of operation, and dynamic voltage scaling. Furthermore, we assume a network interface with multiple power modes, and the ability to switch dynamically between them. Each mobile device i on the wireless network is characterized by: (a) required lifetime of the device defined by start time T_i^s and end time Ti^e, (b) device CPU energy consumption characteristics $p_i^{cpu}(f_i)$ at each possible operating frequency f_i of the CPU, (c) device network interface energy consumption characteristics $p_{i,m}^{net}$ at each mode m, (d) device battery energy constraint (calculated from the lifetime) E_i, and (e) the available bandwidth on the wireless channel BW. We assume in this work that the available bandwidth on the wireless channel BW is shared among all the mobile devices. The details of the CPU and network profiling tasks that yield these device parameters are described in Section 5.2. If at any time, should the device or application characteristics change, the resource allocation algorithm must be re-run with the new parameters.

3.2.2 Process Management Model

The key element of our approach is the decoupling of time-scales of the encoding and transmitting tasks of the video streaming application. In order to manage and schedule these tasks in accordance with their CPU and network resource allocations, we need support from the OS of the mobile device. We use *GRACE-OS* [25], an energy-efficient soft real-time process manager ideally suited to mobile devices running multimedia applications. GRACE-OS integrates dynamic voltage scaling with soft real-time Earliest Deadline First (EDF) scheduling and decides how *fast* (i.e., at what CPU speed) to execute tasks (processes) in addition to when and how long to execute them. It makes task scheduling decisions, and derives not only a task schedule, but also for each task it derives a CPU speed schedule. The CPU speed schedule is based on the probability distribution of task cycle demands. The demand distribution is obtained via profiling of the task. GRACE-OS provides APIs for the application CPU and network tasks to register with the scheduler at the start , and run according to the schedule provided by the scheduler. We have implemented GRACE-OS as a module in the Linux 2.6.5 kernel, and evaluated it on an HP Pavilion N5470 laptop with variable speed AMD Athlon CPU. For details on the online profiling, derivation of the speed schedule, etc., see [25].

3.3 Network Model

As our target scenario indicates, we assume that a number of mobile devices communicate with an access-point (AP) over an IEEE 802.11 wireless LAN. The mobile devices each have a number of real-time applications capturing and transmitting data over the wireless network to receivers in the backbone distribution system. Co-located with the AP is a *Global QoS Manager* (GQM). The GQM is a centralized co-ordinator that takes the device and application models as inputs to its global resource allocation algorithm (Section 3.4) and returns to each sending application its operating QoS level and network schedule. (The application translates this operating QoS level into its corresponding parameters $C_{ij}(q_{ij})$ and $P_{ij}(q_{ij})$ when registering with the GRACE-OS.) A centralized co-ordinator is required because arbitration between the devices/applications for the global shared resource, namely network bandwidth, must occur. This is similar to wireless bandwidth management schemes, for e.g. [24]. However, in our cooperative resource management, the bandwidth management is co-ordinated with the management of the local computing resources such as CPU and energy on each mobile device. The overall co-ordination happens centrally at the GQM to capture the strong inter-dependence between CPU-usage, energy-usage, and bandwidth-usage in media streaming applications.

Our centralized resource arbiter addresses the scenario where limited available energy on a mobile device results in lower CPU-usage for encoding frames and lower network-usage for transmitting them, which further results in more available bandwidth to other applications in the network. An alternate system model could involve the centralized arbiter merely allocating the global shared resource between mobile devices, while local co-ordinators optimize QoS of applications within each mobile device, keeping the allocated bandwidth share, CPU capacity and energy constraint in mind. Our target scenario gives us centralized control of the system and a single centralized resource co-ordinator makes altering the resource allocation policy simpler. We assume that every mobile device registers with the GQM and every application participates in the global coordination. As is evident from our target scenario, there are no random mobile devices that interfere with the functioning of the system.

3.4 Resource Allocation Algorithm

Our scheme takes into account both global resources, shared among multiple mobile devices, and local resources at each mobile device, shared by its applications, in determining the optimal operating level of each application in the network. In this subsection, we describe the global optimization at the GQM,

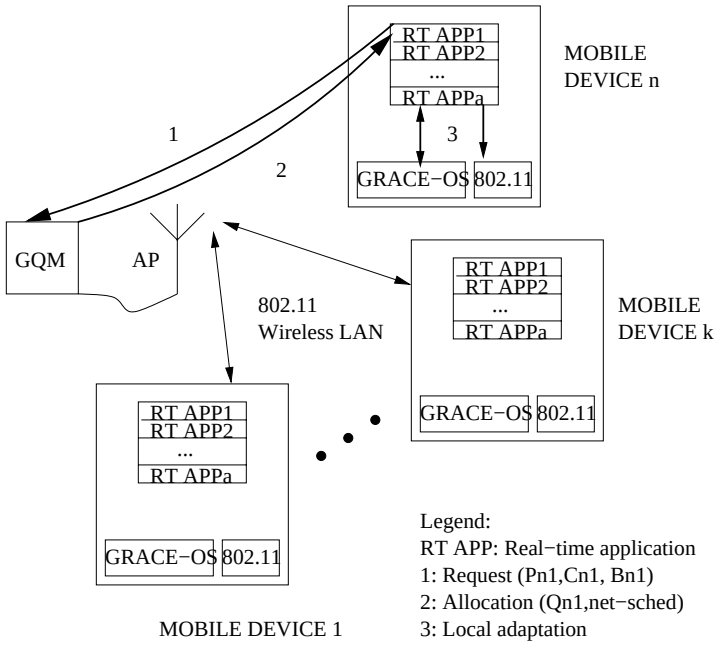

Figure 3: Network model.

which determines the QoS level of each application as well as the average energy consumption of each mobile device, given the system parameters. Devices and applications adapt their operation to *enforce* the global decisions.

The goal of the global optimization is to maximize the aggregate utility of all applications in the 802.11 wireless network subject to the constraints of energy, CPU, and network. Specifically, assume that there are n mobile devices in the network. Each device $i, 1 \leq i \leq n$, has a desired operation lifetime T_i and runs a_i applications. Each application can operate at multiple QoS levels, q_{ij}. Each QoS level q_{ij} provides utility $u_{ij}(q_{ij})$ and demands $C_{ij}(q_{ij})$ cycles and $B_{ij}(q_{ij})$ network bytes per period $P_{ij}(q_{ij})$. The global co-ordination needs to determine the QoS level q_{ij} for each task, the CPU speed f_i and power $p_i^{cpu}(f_i)$ for each mobile device, and the network power p_i^{net}. Obviously, if even the minimum QoS level of an application cannot be supported, it cannot be admitted. The global optimization problem can be formulated as follows:

$$\text{maximize} \qquad \sum_{i=1}^{n}\sum_{j=1}^{a_i} u_{ij}(q_{ij}) \tag{1}$$

$$\text{subject to} \qquad \forall i \sum_{j=1}^{a_i} \frac{\frac{C_{ij}(q_{ij})}{f_i}}{P_{ij}(q_{ij})} \leq 1 \tag{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{a_i} \frac{B_{ij}(q_{ij})}{P_{ij}(q_{ij})} \leq BW \tag{3}$$

$$\forall i \ (p_i^{cpu}(f_i) + p_i^{net}) \times T_i \leq E_i \tag{4}$$

where BW is the available bandwidth in the network, T_i is the operation lifetime of mobile device i and E_i is its residual battery energy.

The CPU energy consumption is directly determined by the CPU speed f_i. Similarly, the network energy consumption is

$$p_i^{net} = \sum\nolimits_m t_{i,m} \cdot p_{i,m}^{net} \tag{5}$$

where $t_{i,m}$ is the time fraction spent by node i's network interface in mode m, and the modes are as listed in Section 5.2. As mentioned earlier, any non-real-time or standalone applications are grouped together and allotted CPU resources and network bandwidth jointly. This is incorporated into our optimization algorithm by having the CPU constraint be slightly less than 1 and BW be slightly less than the actual channel capacity. The remainders get allotted to the non-real-time applications.

The above contrained optimization problem is NP-hard, since we can easily show that the Knapsack problem, which is a well-known NP hard problem, is an instance of the above optimization problem. We therefore use a dynamic programming algorithm that provides a heuristic solution [23]. The resulting QoS level from global co-ordination is enforced locally at each mobile device, via local adaptations inside of applications and the GRACE-OS. Applications perform local adaptation using frame buffering. The GRACE-OS performs local adaptation by via adjusting process and CPU speed schedules and scheduling network transmission bursts.

4 Co-operative Energy-Constrained System Design

This section contains detailed descriptions of the mobile device architecture, network protocols, and the interactions required for enforcing the GQM's resource allocation.

4.1 Mobile Device Architecture

Figure 4 shows a block-diagram of our system architecture. Note that this architecture pertains to energy-saving for a periodic, real-time media streaming application via optimal use of its resources (CPU and network). Further energy saving efforts at the memory and I/O devices can work in conjunction with our architecture.

Each mobile multimedia sender application consists of an encoding and transmitting task joined together by a shared frame buffer. Since our application adaptation involves buffering frames and transmitting them in bursts, we assume frame buffers at both the mobile sender and at the receiver. The former is used to aggregate frames into bursts and the latter is used to smoothen the burst out before playback. This is in contrast to [19, 20], in which buffering happens only at the receiver.

The result of the resource allocation algorithm (see Section 3.4) at the GQM is an operating QoS level for each application in the wireless LAN. This operating QoS level is associated with a network bandwidth consumption. The network bandwidth translates to a fraction of the available bandwidth between the mobile device hosting the application and the AP. From this fraction, the GQM derives a network *schedule* consisting of periods of transmission bursts and inactivity, for each application. During its transmission burst, a sending application has exclusive access to the channel, while during its inactive period, some other application can use the channel. As shown in Figure 4, the GQM returns the QoS level and the network schedule of each application, via a broadcast over the wireless LAN.

The application receives the allotted QoS level and the network transmission/inactivity schedule from the GQM. It registers its encoder and transmitter tasks with the GRACE-OS scheduler in the mobile device. The GRACE-OS scheduler schedules the application's frame encoding/buffering task periodically to conform to the allotted QoS level of the application. Even during periods of network inactivity, frames are still encoded and buffered to be transmitted as a batch during the transmission burst. The GRACE-OS adjusts the CPU frequency in accordance with the cumulative frame-encoding load at the mobile device, over all applications, real-time. It also schedules the transmitter task according to the network schedule. The transmitter task changes the wireless card's mode from power-saving (PSM) to the higher power continuous access mode (CAM), before it begins retrieving buffered frames and transmitting them as a batch. When the transmit schedule is over, the transmitter task changes back the wireless card's mode from CAM to PSM for the inactive period, unless another application on the same device is scheduled to use the card immediately after, for its transmit schedule. Note that, although the transmitter task is shown at the application level in Figure 4, it actually includes the encapsulation of the video frames into network packets in the OS protocol stack, and their transmission via the network interface. These functions are encapsulated in the `send()` system call. Similarly, the encoder task includes the video capture function accomplished via a system call to the video camera. The network power mode-changes are accomplished via `ioctl()` calls from the transmitter task to the 802.11 driver. The GRACE-OS scheduler and the frame buffer thus ensure *decoupling* of encoding/buffering and network access timescales, since the encoding task will obey the process schedule (e.g., one frame encoded every 0.5 seconds for 2fps frame-rate), and the transmitter task will obey the network schedule (e.g. one 200ms burst every 10 seconds). The separation of encoding and network tasks in a mobile video application, for decoupling of time-scales and consequent ease of adaptation was also proposed in [21]. However, the authors only adapt the application in their work, while we co-operatively adapt both the system and application.

4.2 Network Protocol

In our scheme, we attempt to minimize energy consumption at the wireless network interface by judiciously switching the card into power saving mode (PSM) when it is not active. We attempt to *batch* or *accumulate* all the network activity of an application into bursts so that the wireless card can remain in PSM over long, uninterrupted stretches. In order to achieve this, our system must have control over transmissions on the channel. For this purpose, the transmission task, which includes drawing frames from the buffer, controlling the network interface power mode-change, and transmission over the network, is registered with the GRACE-OS.

We have developed a set of network mechanisms to accomplish our goal of energy saving at the wireless-interface. Note that our network protocol described in this section works entirely at the *application* level. We assume a standard UDP/IP protocol stack over IEEE 802.11b Distributed Co-ordination Function (DCF) MAC protocol. We use off-the-shelf wireless cards and do not modify the 802.11 protocol in any way. We use IEEE 802.11 as the MAC layer because of its high data-rate and its popularity.

Essentially, an application transmitting a burst of data, and then going quiet for a length of time during which another application transmits its burst, amounts to time-slotting of the channel. We thus impose time-slotting via application-level mechanisms over a shared channel. The IEEE 802.11e standard [3], and indeed the IEEE 802.11 PCF also proposes time-slotting of the channel at the MAC level for QoS and contention avoidance. However, in these cases, the AP polls each mobile device for data at the start of its transmission burst. We use time-slotting for energy-saving, and obtain reduced contention as a by-product. We also avoid polling overhead.

Figure 5 illustrates the working of our application-level network transmission mechanisms. It also shows the decoupling of the time-scales of CPU and network activities. Shown in the figure is one GQM cycle of length T, and the CPU and network activity of a mobile device with a single application, during the cycle. The cycle starts with a *start pulse* from the GQM [1]. The start pulse signifies the end of data transmissions belonging to the previous cycle, and also acts as a beacon solicitation for network access requests from new applications in the wireless

[1] Since we work on the application level, when we refer to a "pulse," we mean a short data packet broadcast transmission.

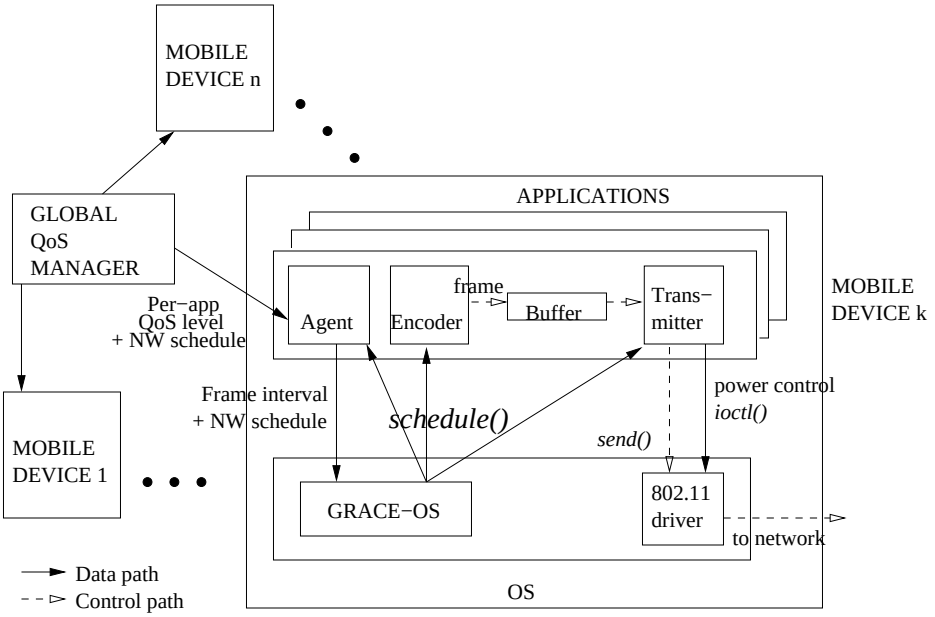

Figure 4: System design.

network. Requests from applications contain the application parameters described in Section 3.1. (The GQM obtains mobile device parameters described in Section 3.2 when a mobile device enters the network.) Existing applications can also submit re-negotiation requests, based on changed application/device parameters, after the start pulse. Requests can continue to be submitted for a duration T_p after the start pulse. New or re-negotiating applications can only submit requests immediately after the start pulse, not at other times in the cycle. Each request contains an application ID, which is a tuple containing the sender address, receiver address and port number used by the application for its data transmissions. Contention for the wireless channel between requests during the time interval T_p is dealt with by the underlying 802.11 protocol's contention resolution mechanism. After time T_p, the GQM runs its optimization algorithm and returns the results to all applications, and the AP, via a broadcast *second pulse*.

The second pulse contains the allotted QoS level of each application and its network transmission schedule. The applications are identified by their application ID. The second pulse may indicate a change of QoS level and transmission schedule for existing applications. The network schedule consists of a *transmit start time* t_{st} and a *transmit end time* t_{et}, which are both represented as offsets from the second pulse reception time at the application. A broadcast transmission over a single wireless hop is received at virtually the same time by all the recipients. The second pulse thus acts as a *synchronization* pulse. All network schedules are offsets from this synchronization pulse, which is renewed every cycle time T. In case there is some clock skew across mobile devices, adjacent network schedules may overlap and contention may occur at burst-boundaries. However, the IEEE 802.11 contention resolution mechanism handles this adequately [2].

The GQM ensures that $t_{et} - t_{st} = \frac{B^g_{ij} \cdot T}{BW}$ for an application j on mobile device i with unlimited buffer size. B^g_{ij} is the bandwidth allotted to application j on mobile device i, at its granted QoS level q^g_{ij}. If buffer size is at either the sender or receiver is limited, then the application gets allotted multiple bursts ($\lceil \frac{B^g_{ij} \cdot T}{bufsize_{ij}} \rceil$ bursts) each of maximum duration $\frac{bufsize_{ij}}{BW}$, spaced evenly apart in T. The GQM also adds some "grace" time to each transmit burst to account for the possible preemption of the network task by the frame encoding task. In our testbed, the grace time was also required to deal with stray contention on the channel from other wireless LAN installations in the building.

After transmission bursts from all the applications, at a time offset T from the previous start pulse, a new start plse is sent marking the start of a new cycle. The request-processing time T_p reduces the efficiency of the network, but the loss can be minimized by having $T_p \ll T$.

4.3 CPU and Network Adaptation

We now discuss how to enforce the global decisions locally at each mobile device. The CPU adaptation at each mobile device is straight forward: we set the CPU speed based on the total CPU demand of all applications in each mobile device. Specifically, if a mobile device i runs a_i applications and each application is globally configured with QoS level q_{ij}, which demands $C_{ij}(q_{ij})$ cycles per period $P_{ij}(q_{ij})$, then we set the CPU speed of device i as $\sum_{j=1}^{a_i} \frac{C_{ij}(q_{ij})}{P_{ij}(q_{ij})}$. This speed is the lowest speed meeting the CPU constraint in Equation 2. For details

[2] Alternatively, adjacent bursts can be spaced apart by a short time gap.

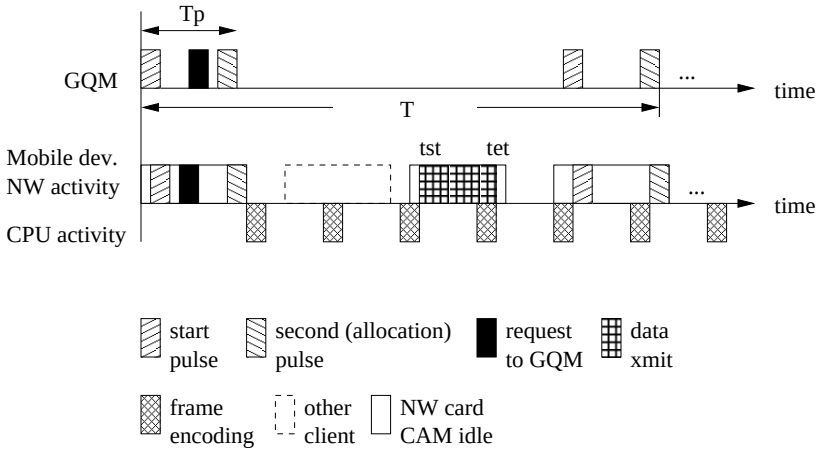

Figure 5: System timeline: one cycle.

of the CPU adaptation, please refer to our previous work on GRACE-OS [25]. The GRACE-OS schedules the non-real-time or standalone applications, within their pseudo-real-time class, on a first-come, first-serve basis.

For the network adaptation, the GRACE-OS schedules the transmission task well in advance of the application's t_{st}. This is because the transmission task must initiate a network interface mode change from PSM to CAM, and the transition times for this can vary. We measured `ioctl()` call times to be 50-60ms on average, but our individual readings varied between 15 and 90ms. At the end of the transmission schedule, the network card is switched back from CAM to PSM. If applications from the same mobile device have adjacent network schedules, then the mode switches are not required. This is indicated via a flag in the GQM's schedule allotment. The GQM *tries* to schedule the bursts of multiple applications on the same mobile device in an adjacent manner so as to minimize transitions from PSM to CAM and back.

GRACE-OS also switches the wireless interface of its mobile device from PSM to CAM to receive the two pulses from GQM, and to transmit any re-negotiation requests. In general, the card is in CAM for any network activity while it does not perform any network activity when in PSM. All mode changes are accomplished via an `ioctl()` call from the transmission task. The period is set very large for PSM.

5 System Profiling

As mentioned above, the optimization algorithm at the GQM takes certain mobile device- and application-specific parameters in order to achieve its goal of aggregate utility maximization under the constraints. These parameters are obtained via profiling of the periodic, real-time media-streaming application, of the CPU in the mobile device, and of the wireless network interface card (NIC). In this subsection, we present the profiling methodology for each of these categories, and the results we obtained for our hardware and prototype application.

5.1 Application Profiling

We consider *periodic* multimedia tasks (processes or threads) that release a job (e.g., encodes and transmits a video frame) every period. Each job consumes CPU, network, and energy resources during its execution. To provide QoS guarantees, we need predictable resource management, which in turn needs to predict the resource demand for each QoS level of each task. This subsection describes how we predict the CPU, network, and energy demand.

We take the kernel-based profiler in GRACE-OS [25] to predict the CPU demand. Specifically, we add a cycle counter to the process control block of each task [25]. This cycle counter monitors the cycle usage for each job of the task. Based on this monitored cycle usage, we then build a histogram to estimate the probability distribution of cycle demand. From the histogram, we predict the CPU cycle demand of the task as a percentile of cycle usage. For example, if a task requires meeting 96% of deadlines and 96% of its jobs use no more than 10^6 cycles, we predict the cycle demand as 10^6 cycles per period.

We take a similar approach to predict the network demand for each task. That is, we monitor the number of bytes demanded for each period and then predict the network byte demand as a percentile of the previous network usage. For our h.263 application, the 95th percentile CPU cycle demand for a frame encoding task was 114 million cycles, and its 95th percentile network demand was 20585 bytes.

Energy prediction is a little different since the OS does not manage energy as a first class resource. That is, energy is consumed by hardware components such as CPU and NIC, rather than by individual applications directly. We therefore predict the energy consumption for different states of the hardware components as follows.

5.2 Energy Consumption Profiling

We need to profile both the CPU and network interface energy consumption at various speeds/modes so that we can plug these profiled parameters into our resource allocation algorithm

of Section 3.4. We now describe these energy consumption profiling experiments and their results.

5.2.1 CPU Energy Consumption Profiling

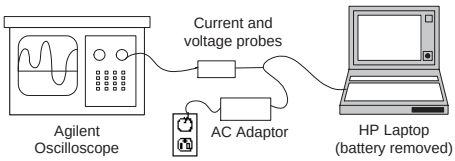

Figure 6: Power measurement with oscilloscope.

We consider mobile devices with a single adapative CPU that can operate at different speeds. Different speeds provide different performance and consume different amount of energy. Since it is difficult to measure the energy consumption for the CPU only, we measure the total energy consumed by the whole device (but without NIC) at different speeds. The AMD Athlon processor can adapt its frequency between 300MHz and 1000MHz. We use an Agilent 54621A oscilloscope to measure the base energy consumption of the entire HP Pavilion N5470 laptop, with batteries removed, at each of these CPU speeds, as shown in Figure 6. The oscilloscope measures current intake, which multiplied by the constant voltage rating of 19 Volts for the laptop gives the instantaneous power consumed. The current intake is measured around 400 times per second, and the average is obtained. The base energy consumption of the laptop alone, at different speeds, is listed in Table 1.

5.2.2 Network Energy Consumption Profiling

The IEEE 802.11b standard specifies that every compliant network card must have two power modes: a power-saving mode (PSM) and a continuous access mode (CAM). The details of the implementation of these modes can be found in the standard [2]. Our system puts the card in PSM when there is no network activity for any application on the mobile device, and having it in CAM for transmission and reception. The card is then always idle in PSM and always in CAM for any activity. The card may also spend some idle time in CAM. We thus measure PSM idle power ($p^{net}_{i,PSMidle}$), CAM idle power ($p^{net}_{i,CAMidle}$), CAM transmit power ($p^{net}_{i,CAMxmit}$), and CAM receive power ($p^{net}_{i,CAMrecv}$) consumption.

We next disable CPU speed adaptation, and repeat the energy-consumption experiments with a Cisco Aironet 350 wireless card inserted and idle in CAM, and obtain the energy consumption of the entire system. The energy consumed by the laptop without the card is subtracted from this to give the CAM idle power of the wireless card alone. The PSM idle energy consumption can also be similarly obtained.

For measuring transmit power, we used the network setup of Figure 7. The mobile laptop connected to the oscilloscope transmitted 18KB packets using UDP, as fast as possible, to the stationary laptop. (The mean frame size of our h.263 application was 18KB, and we also used UDP for streaming.) The power consumption experiment was repeated, the base power of the laptop subtracted, and the CAM transmit power obtained. The CAM receive power was similarly obtained except that, for this experiment, the stationary laptop was transmitting 18KB UDP packets in a tight loop to a dummy receive application at the mobile laptop, connected to the oscilloscope. The power consumption results are tabulated in Table 2.

In addition to the energy consumption of the wireless card, we also measured the mode-change overhead. We found the `ioctl()` call return times to vary between 15 and 90ms, yielding an average of 50-60ms. This constitutes 0.5-1% of the cycle time T, which is set at 10 seconds in our experiments. (See Section 6.) Since this is a negligible amount, we do not use it in either our resource allocation algorithm, or in our analytical results. We note however that, given this overhead, network card mode change is not feasible more than a few times in one cycle. It is certainly not feasible to do it every time a frame is encoded, for frame-rates of 2-3fps. Our batched transmission is hence the solution for network energy reduction at low overhead.

6 Experimental Results

In this section, we describe our experiments with the coordinated, energy-constrained resource management framework, and present our results. Our experiments are intended to illustrate the benefit of using our scheme in terms of energy-saving at the mobile device, and to study its impact on application performance.

6.1 Implementation

Our experimental setup is shown in Figure 7. We used a h.263 application with 3 QoS levels corresponding to frame-rates of 1, 2, and 3 frames per second (fps) respectively. As shown in the figure, our testbed consists of an HP Pavilion N5470 mobile laptop with CPU speed-adaptive AMD Athlon processor, that represents the robot in the scenario described in Section 1, connected over a 802.11 wireless network through an access-point and a crossover cable to a stationary laptop of the same model that represents the space shuttle. The mobile laptop contains the h.263 encoder/transmitter application, that uses our architecture, running on top of RedHat Linux 8.0, kernel version 2.6.5, enhanced with the GRACE-OS CPU speed adaptation and task-scheduling functionality. The stationary laptop acts as the Global QoS Manager (GQM) and also contains receiver/playback code for the h.263 application. The GQM contains the optimization code as described in Section 3.4, the parameters of which are configured a priori via the results of the CPU and network profiling experiments of Section 5.

The cycle time T of the GQM is set to 10 seconds for all our experiments, and T_p is set to 950ms. The utility of the application at a given QoS level is set equal to the frame-rate at that QoS level [3]. The CPU cycles and network bandwidth

[3] Our aim is not to find suitable frame-rate to utility mappings. Our system maximizes utility, irrespective of how it is derived from the QoS

CPU frequency	300MHz	500MHz	600MHz	700MHz	800MHz	1000MHz
Power (Watt)	22.25	25.84	28.24	31.05	35.44	39.06

Table 1: Speed-adaptive AMD Athlon CPU energy consumption profiling results.

Mode	$p^{net}_{i,PSMidle}$	$p^{net}_{i,CAMidle}$	$p^{net}_{i,CAMxmit}$	$p^{net}_{i,CAMrecv}$
Power (Watt)	0.36	1.27	2.56	2.38

Table 2: Cisco Aironet 350 network interface energy consumption profiling results.

consumed at each QoS level are determined through application profiling. The available bandwidth is set constant slightly below the theoretical channel capacity of 11Mbps. The energy constraint is set arbitrarily so that we can have applications adapt to various QoS levels [4]. The GQM directs the application to change its QoS level when available energy varies, and as the number of applications sharing the network bandwidth varies. We show the energy saving at two different QoS levels that the application is forced to adapt between. Although we cannot experiment with several mobile devices in the same wireless LAN due to equipment constraints, our energy saving and application performance results from this section are still valid because our time-slotting scheme virtually eliminates contention for the globally shared resource.

6.2 Results

We show the energy saving benefit of using our scheme via analytical results, confirmed by oscilloscope measurements. We measured average energy consumption characteristics for three different system scenarios: (a) no CPU speed adaptation or wireless card mode-changing ("No adaptation"), (b) CPU speed adaptation followed by instant frame transmission, i.e. no batching or buffering and hence no wireless card mode-changing ("CPU-only"), and (c) both CPU speed adaptation as well as wireless card mode-changing ("CPU+NW"). This gives us an idea of the energy-saving resulting from each of the individual adaptation mechanisms, namely CPU speed adaptation and wireless card mode changing. Further, we measure average base energy consumption of the laptop, average energy consumption of the wireless interface and the sum of the two, so that we can measure the energy saving independently for each component.

The results presented in this section are obtained by summing, in the OS, the product of the fraction of time spent at each CPU speed and the profiled energy consumption at that speed. This approach for energy measurement makes sense for the following reason. For a short time interval, we measured the energy with the Aglient oscilloscope directly and found that the measured energy is roughly the same as the energy result calculated in the above manner. For a long time interval, however, we are unable to measure the energy with the oscilloscope directly since this requires connecting the oscillope to another computer for sampling the data from the oscillope. Similarly, the network interface energy consumption results are derived by summing the product of the fraction of time spent at each network card power mode from Table 2, with the energy consumption at that mode. As mentioned earlier, we ignored the transition overhead for the PSM-to-CAM and CAM-to-PSM mode changes because these took a negligibly short time.

6.2.1 Energy Saving in HP Pavilion Laptop

Figures 8, 9, and 10 show the base energy consumption of the HP Pavilion laptop, the energy consumption of the Cisco Aironet 350 wireless network interface, and the sum of the two, respectively, at two different application QoS levels. We assume a single application per mobile device in this experiment, which is made to adapt its QoS as the optimization parameters are varied.

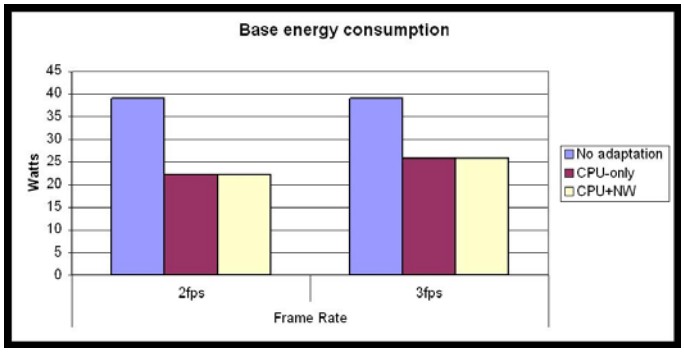

Figure 8: Base energy consumption of HP Pavilion laptop.

We note that the CPU speed adaptation reduces base energy consumption of the laptop by about 34% at 3fps, and the network card mode-changing reduces the energy consumption of the network interface by about 42% at the same frame-rate. However, because of the relative energy consumption scales of the laptop as a whole and the network interface alone, we find that the total energy saving in the CPU+NW case over the CPU-only case is only 2-3% with the HP Pavilion laptop. The effort of batching, buffering, and intelligent network-card mode-changing may thus not be justified by the results when using this particular laptop. The effect of batching and buffering on the application is shown in Figure 11, which shows the bursty

levels.

[4]We use an arbitrary function to convert lifetime to available energy. In reality, the battery power consumption vs. time curve must be used to perform this mapping.

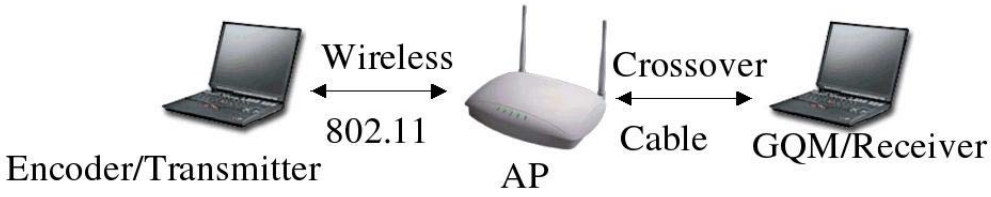

Figure 7: Experimental setup.

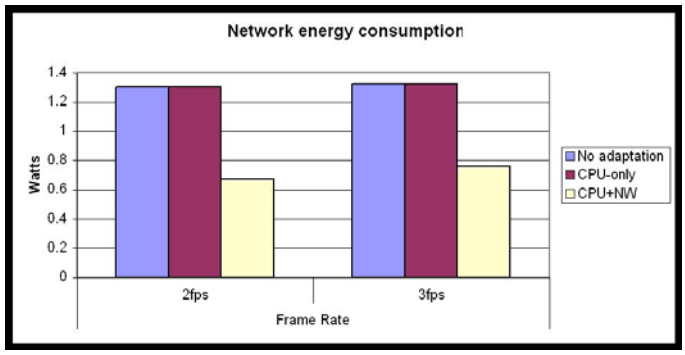

Figure 9: Network energy consumption of Cisco Aironet 350 PCMCIA card.

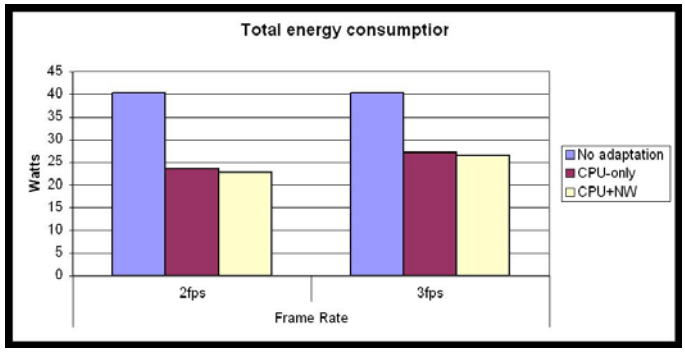

Figure 10: Total energy consumption of HP Pavilion laptop with Cisco Aironet 350 PCMCIA card.

frame arrival at the receiver for 3fps encoding rate, for a 25-second snapshot of the experiment. The figure shows frames arriving in bursts cycle time T apart for the CPU+NW adaptation scheme, whereas they arrive at the encoding rate for the other two cases.

6.2.2 Hypothetical Mobile Devices

As mentioned above, in our experiments with the HP Pavilion laptop we notice that, while CPU speed adaptation is beneficial in reducing overall energy consumption, batching/buffering network card mode-changing is not that useful. This is because of the relative energy consumption scales of the laptop as a whole and the wireless card alone. However, network card

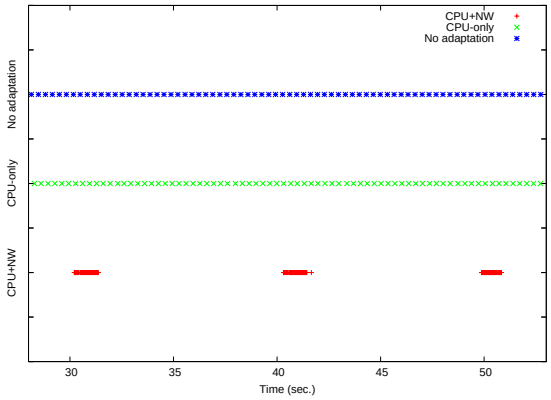

Figure 11: Bursty frame arrival at receiver for 3fps encoding rate.

Frequency (MHz)	Voltage (Volts)	Relative Power
300	1.2	.22X
500	1.2	.36X
600	1.25	.47X
700	1.3	.60X
800	1.34	.74X
1000	1.4	X

Table 3: Variation of CPU power with frequency according to the formula $f \propto f \cdot V^2$.

adaptation could still be useful in the case where the difference in the base energy consumption of the mobile device and that of the network interface is not large. This is also alluded to in [4]. Handhelds such as iPAQs may fall into this category [5]. In general, as handheld computers become more and more capable, and as laptops become smaller and more energy-aware, our co-ordinated architecture becomes more and more relevant. In this subsection, we illustrate this via experiments with hypothetical mobile devices simulated by the HP Pavilion laptop itself.

We simulate 3 hypothetical mobile devices: a 10 W base power device with a 5 W CPU, an 8 W base power device with a 4 W CPU, and a 6 W base power device with a 3 W CPU. Just for comparison, the base power of an idle iPAQ is approx-

[5]We did not use iPAQs in our experiments because they do not have the capability to support complex, multi-threaded, real-time multimedia applications.

imately 1.5 Watts [4]. The CPU power mentioned above is the CPU power at the highest speed. CPU power varies with speed according to the formula $p \propto f \cdot V^2$, where p is the CPU power, f is the CPU frequency and V is the voltage. Table 3 shows this variation of CPU power consumed with CPU speed. For this experiment, the optimization algorithm in the GQM was configured with the hypothetical CPU energy consumption values. Figure 12 shows the total energy saving obtained by running a single h.263 application that adopts our co-operative architecture at 2fps on the hypothetical mobile devices. The figure shows that, in addition to energy savings resulting from CPU speed adaptation (compare CPU-only vs. No adaptation plot), the network card mode-changing also further reduces total energy consumption by 9-13% (compare CPU+NW vs. CPU-only plot), which is a significant amount. We also obtained energy reduction via network card mode-changing of 7-10% for the 3 hypothetical mobile devices, at frame rate 3fps.

As mobile devices get smaller and smaller, energy reduction via network card mode-changing becomes more and more significant. We can thus use our architecture in a dynamic, adaptive way depending on the power characteristics of the underlying mobile device. If there is very little energy-saving to be gained for *any* mobile device in the network via our network mechanisms, we can disable them, and use CPU speed adaptation alone, with instant transmission of frames once they are encoded. On the other hand, if there is a very low power CPU in the future, or a CPU without speed adaptation capabilities, then CPU speed adaptation part of our architecture can be abandoned, while network energy reduction, if significant, can still be enabled.

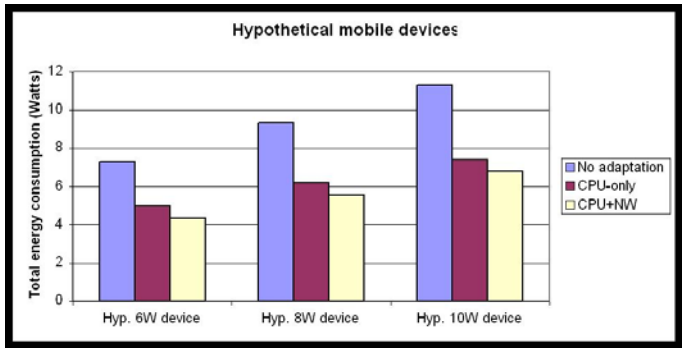

Figure 12: Total energy consumption for hypothetical mobile devices at 2fps.

switch modes *twice* in the same cycle. While transition cost itself still remains low due to the negligible transition times, energy consumption is still affected because we switch the card into the higher-power CAM mode well in advance of the scheduled transmission time. Furthermore, for each transmit schedule, the GQM adds some "grace" transmit time to account for pre-emption of the network task by the scheduler to accommodate the encoder task, and to account for stray contention. Figure 13 shows the effect of limited buffer size on energy consumption and Figure 14 shows the effect of limited buffer size on the frame arrival at the receiver. For very small buffer sizes, mode-changing has to happen very frequently. In this case, mode transition times become significant with respect to the cycle time T.

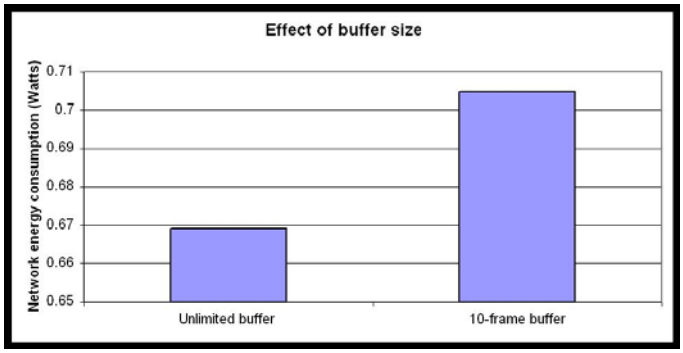

Figure 13: Effect of buffer size on energy consumption at 2fps.

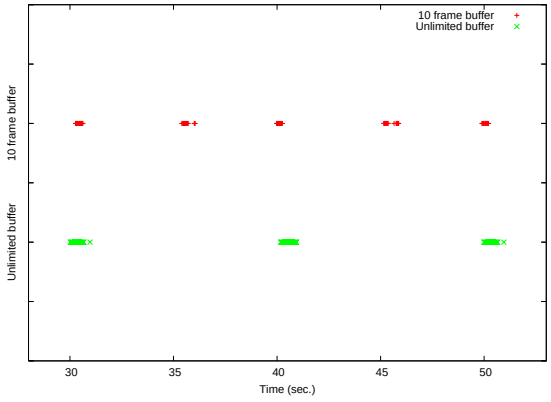

Figure 14: Effect of buffer size on frame arrival at 2fps.

6.2.3 Effect of Buffer Size

All our previous experiments described in this section assumed unlimited buffer space at both sender and receiver. In this subsection, we examine the effect of using a 10-frame buffer on energy consumption and frame arrival at receiver. Note that the smaller the buffer, the more frequently the network card has to switch modes within the same cycle time T, according to the buffer-size-aware schedule specified by the GQM. For a 2fps frame-rate and 10-frame buffer, the network card has to

6.2.4 CPU Speed Adaptation with Multiple Streams

In the previous sections, we showed the mobile device energy consumption characteristics and application performance for a single application on a mobile device. The application adapted its performance over different QoS levels as the local and global parameters were varied, and we presented results for multiple QoS levels.

We now perform experiments to illustrate the local CPU speed adaptation in response to local load. The load is var-

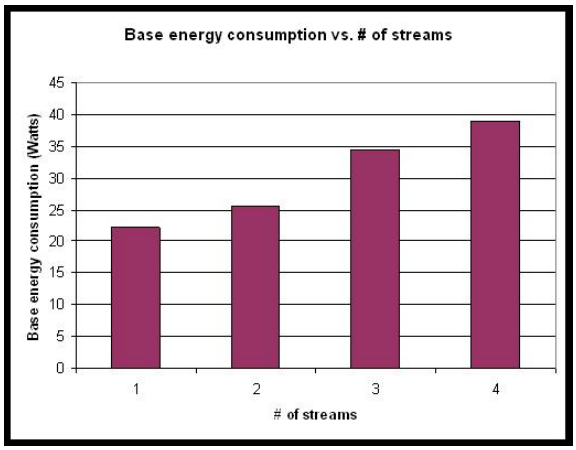

Figure 15: Base energy consumption of HP Pavilion with multiple h.263 streams.

ied by having *multiple* h.263 streams encoded and transmitted from a single mobile device (the HP Pavilion laptop). The results are shown in Figure 15. Each stream has a frame rate of 2fps. We measured only the base energy consumption of the laptop for this experiment because we are illustrating CPU speed adaptation, which is a local adaptation, and the network bandwidth is a global resource. Obviously energy consumption at the network interface also increases with increase in number of video streams being transmitted.

6.3 Oscilloscope Results

We verified the correct working of our system, and our analytical results, by connecting the oscilloscope to the mobile laptop and using it to measure energy consumption. Figure 16 shows a snapshot of the operation of an HP Pavilion with a single h.263 application at 2fps. The image was extracted from a digital photograph of the oscilloscope in operation. The various events comprising the system, namely network card in PSM mode, transition to CAM mode, and frame encoding/buffering are marked. The oscilloscope also gives the average current consumption, in Amperes, over a measurement interval. For 2fps, we obtained a current consumption of 1.2-1.25 amperes, and for 3fps, we obtained current consumption of 1.4-1.45 amperes. The current consumption multiplied by the fairly constant voltage consumption (around 19 Volts) gives the average power consumption in Watts. In each case, the oscilloscope measurement confirms our analytical result.

7 Conclusion

In this paper, we present a co-operative CPU and network management design, for energy-constrained mobile multimedia environments. Our integrated architecture combines global coordination among multiple mobile devices with local energy saving via CPU speed adaptation and intelligent wireless network interface mode change. The co-operative scheme results in both the CPU and network performance yielding the same operating QoS for the multimedia streaming application. Our scheme is also designed to manage shared and local resources while maximizing application QoS and meeting energy constraints. We implement and experiment with our scheme in a real testbed environment, using an adaptive h.263 prototype applications. Our results show energy-saving benefits (34% at the CPU and 42% at the network interface of a mobile device with a single 3fps h.263 application), at both the CPU and network interface. We find that the cooperative resource management is most effective for saving energy when the CPU and network interface consume roughly the same power, which is becoming increasingly common for mobile devices. Consequently, we discuss adaptive deployment of our scheme taking into account the energy consumption characteristics of the mobile device.

References

[1] S. Adve, A. Harris, C. Hughes, D. Jones, R. Kravets, K. Nahrstedt, D. Sachs, R. Sasanka, J. Srinivasan, and W. Yuan. The Illinois GRACE Project: Global Resource Adaptation through CoopEration. In *Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN) (held in conjunction with the 16th Annual ACM International Conference on Supercomputing)*, June 2002.

[2] IEEE 802.11, 1999 edition (iso/iec 8802-11: 1999) IEEE Standards for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan area network - specific requirements - part 11: Wireless LAN Medium Access Control (MAC) and Physical layer (PHY) specifications. *http://standards.ieee.org/getieee802/download/802.11-1999.pdf*

[3] S. Mangold, S. Choi, P. May, O. Klein, G. Hiertz, and L. Stibor. IEEE 802.11e Wireless LAN for Quality of Service. In *European Wireless*, Florence, Italy, February 2002.

[4] M. Anand, E. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *ACM MobiCom*, San Diego, CA, September 2003.

[5] S. Saewong and R. Rajkumar. Practical Voltage Scaling for Power-Aware Real-Time Systems. In *IEEE RTAS*, San Jose, CA, September 2002.

[6] S. Saewong and R. Rajumar. Optimal Static Voltage-Scaling for Real-Time Systems. In *IEEE RTSS*, Austin, TX, December 2002.

[7] E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *ACM MobiCom*, Atlanta, GA, July 2002.

[8] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *ACM MobiCom*, Atlanta, GA, July 2002.

[9] D. Qiao, S. Choi, A. Jain, and K. Shin. MiSer: An Optimal Low-energy Transmission Strategy for IEEE 802.11a/h In *ACM MobiCom*, San Diego, CA, September 2003.

[10] J. Monks, V. Bharghavan, and W. Hwu. A Power Controlled Multiple Access Protocol for Wireless Packet Networks In *IEEE InfoCom*, Anchorage, AK, April 2001.

[11] V. Kawadia and P. Kumar. Power Control and Clustering in Ad Hoc Networks. In *IEEE InfoCom*, San Francisco, CA, April 2003.

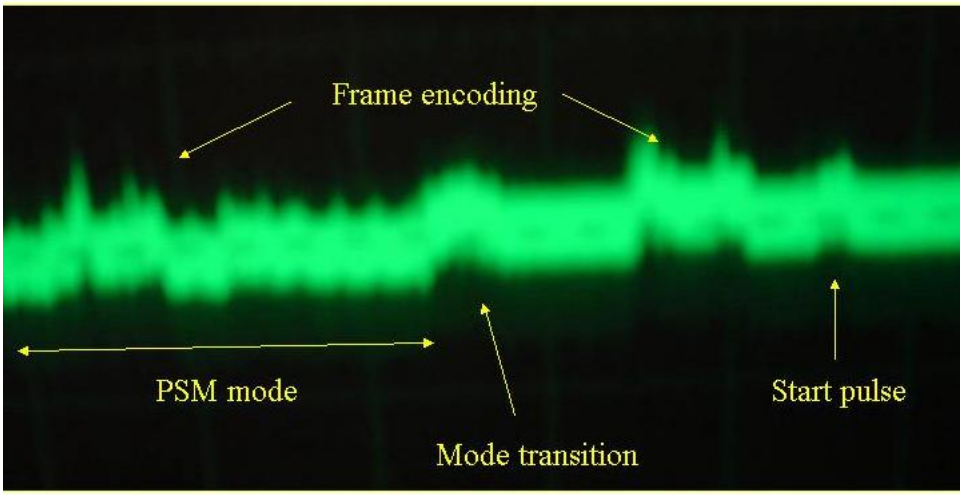

Figure 16: Oscilloscope snapshot.

[12] R. Wattenhofer, E. Li, P. Bahl, and Y. Wang. Distributed Topology Control for Wireless Multihop Ad-hoc Networks. In *IEEE InfoCom*, Anchorage, AK, April 2001.

[13] M. Woo, S. Singh, and C. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. In *ACM MobiCom*, Dallas, TX, October 1998.

[14] R. Zheng and R. Kravets On-demand Power Management for Ad Hoc Networks. In *IEEE InfoCom*, San Francisco, CA, April 2003.

[15] A. Acquaviva, T. Simunic, V. Deolalikar, and S. Roy. Remote Power Control of Wireless Network Interfaces. In *PATMOS*, Torino, Italy, September 2003.

[16] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated Power Management for Video Streaming to Mobile Devices. In *ACM Multimedia*, Berkeley, CA, November 2003.

[17] R. Kravets and P. Krishnan. Application-Driven Power Management for Mobile Communication. *ACM/URSI/Baltzer Wireless Networks (WINET)*, 6(4), July 2000.

[18] J. Flinn and M. Satyanarayanan. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Transactions on Computer Systems*, 22(2), May 2004.

[19] S. Chandra and A. Vahdat. Application-specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats. In *USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[20] C. Im, H. Kim, and S. Ha. Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications using Buffers. In *Intl. Symposium on Low-Power Electronics and Design*, Huntington Beach, CA, August 2001.

[21] M. Corner, B. Noble, and K. Wasserman. Fugue: Time Scales of Adaptation in Mobile Video. In *SPIE MMCN*, San Jose, CA, January 2001.

[22] F. Xie, M. Martonosi, and S. Malik. Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *ACM PLDI*, San Diego, CA, June 2003.

[23] M. Moser, D. Jokanovic, and N. Shiratori. An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem. *IEEE Transactions on Fundamentals*, 80(3), March 1997.

[24] S. Shah, K. Chen, and K. Nahrstedt. Dynamic Bandwidth Management for Single-hop Ad Hoc Wireless Networks. *ACM/Kluwer Mobile Networks and Applications (MONET) Journal, Special Issue on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks*, 10(1), February 2005.

[25] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *ACM SOSP*, Bolton Landing, NY, October 2003.

[26] T. Simunic, L Benini, P. Glynn, G. De Micheli. Dynamic Power Management for Portable Systems. In *ACM MobiCom*, Boston, MA, August 2000.

[27] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the IpARM Microprocessor System. In *ACM Intl. Symposium on Low Power Electronics and Design*, Rapallo/Portofino Coast, July 2000.

[28] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *ACM Intl. Symposium on Low Power Electronics and Design*, Monterey, CA, August 1998.

[29] D. Grunwald, P. Levis, K. Farkas, C Morrey III, and M. Neufeld. Policies for Dynamic Clock Scheduling. In *ACM OSDI*, San Diego, CA, October 2000.

[30] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. *ACM OSDI*, Monterey, CA, November 1994.

[31] P. Pillai and K. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *ACM SOSP*, Banff, Canada, October 2001.

[32] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *IEEE RTSS*, London, United Kingdom, December 2001.