Network-Aware Dynamic Voltage and Frequency Scaling *

Bren Mochocki, Dinesh Rajan, Xiaobo Sharon Hu, Christian Poellabauer,
Kathleen Otten and Thidapat Chantem

Department of CSE
University of Notre Dame
Notre Dame, IN 46556
{bmochock, dpandiar, shu, cpoellab, kotten, tchantem}@nd.edu

Abstract

Reducing energy consumption is an important consideration in embedded real-time system development. This work examines systems that contain a DVFS managed CPU executing packet producing tasks and a DPM-controlled network interface. We introduce a novel approach to minimize energy consumed by the network resource on such a system, through careful selection of voltage and frequency levels on the CPU. Contrary to existing claims which state that DVFS should not be employed when the CPU is not a significant consumer of energy, we show that our DVFS technique can reduce system energy by as much as 35%, even when the CPU energy consumption is negligible. Furthermore, we motivate the need to balance the CPU and network energy and present two techniques to do so. One is based on off-line analysis and the other is a conservative on-line approach. We then validate the proposed methods using both simulation and an implementation in the Linux kernel.

1 Introduction

The design of energy-aware systems has attracted considerable attention from the real-time system community in recent years. Much progress has been made with regard to task scheduling to reduce energy consumption in various hardware components or resources in a real-time system. For example, energy management for CPUs utilizing the widely popular Dynamic Voltage and Frequency Scaling (DVFS) technique is well studied for different task models (e.g., [25, 28]), circuit-level models (e.g., [8, 16]), and scenarios (e.g., [21, 14]). Reducing the energy consumed by other types of resources, such as memory, disks and network cards has also been investigated by a number of researchers (e.g., [13, 24, 26, 19, 27]).

Almost all real-time embedded systems are implemented by more than one hardware resource. In a large portion of handheld devices, where energy consumption is a critical concern, the system often contains a network card and/or other I/O devices in addition to the CPU. It is not difficult to imagine situations where reducing the energy consumption of one resource can increase the energy consumption of others, resulting in reduced savings or even an increase in the total energy consumed.

In this paper, we examine a common system architecture that consists of a DVFS capable processor and a network interface managed by its own independent dynamic-power-management (DPM) mechanism. This architecture can greatly ease the system development effort and is widely adopted. Our objective is to reduce the energy consumption of such a system as a whole by maximally exploiting the DVFS and DPM capability in the presence of dynamic changes.

One main contribution of this paper is the introduction of a novel network-aware DVFS algorithm that, rather than minimizing CPU energy consumption, seeks to minimize the energy consumed by the *network interface*. Energy is saved by carefully controlling packet release times through DVFS, while exploiting the known behavior of the network's DPM mechanism. Contrary to existing claims which state that DVFS should not be employed when the CPU is not the dominant consumer of system energy [30], we show that our DVFS technique can reduce system energy by as much as 35% when the CPU energy consumption is negligible compared to the network interface. Our second contribution is an on-line method for reducing the system-wide energy. This method leverages a network-aware DVFS algorithm and a CPU-centric DVFS algorithm. The technique is easy to implement and has a low run-time overhead. Finally, we have implemented these algorithms on an experimental platform to validate the work.

The rest of the paper is organized as follows. Section 2 presents the problem definition and also reviews related work in detail. In Section 3, we use a motivational example to illustrate the significance of the problem under consideration and reveal some interesting properties. Sections 4 and 5 give the details of our technique, while Section 6 summarizes key experimental results. Finally, we conclude the paper in Section 7.

*This work is supported in part by NSF under grant numbers CNS-0410771 and CNS-0545899 and by the University of Notre Dame Slatt Fellowship.

2 Preliminaries

We consider a system with a DVFS capable CPU and a DPM capable network interface. (Note that our methods can be readily extended to other types of devices with DPM capabilities.) Below, we first describe the task and hardware model under consideration, the necessary notation and the formal problem definition. Next, the related work is presented in detail.

2.1 Problem Formulation

We consider real-time applications consisting of a set of n periodic tasks, $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$. Each task, T_i, is described by its worst case execution cycles, wc_i, average case execution cycles, ac_i, and best case execution cycles, bc_i, with $wc_i \geq ac_i \geq bc_i$. In addition, each task has a period, p_i, and relative deadline, d_i, with $d_i \leq p_i$. Each task is invoked periodically and we refer to the k-th invocation of task T_i as job J_i^k. Each job is described by a release time, r_i^k, deadline, d_i^k, finish time, f_i^k, and actual execution cycles, c_i^k, where $bc_i \leq c_i^k \leq wc_i$. Tasks are scheduled according to the popular preemptive earliest deadline first (EDF) algorithm.

We assume that each CPU job, say J_i^k, generates one transmission request to the network interface at the end of its execution, denoted by Q_i^k. The size of a request is denoted z_i^k and must satisfy $bz_i \leq z_i^k \leq wz_i$, where bz_i and wz_i are the best and worst case request sizes of task T_i, respectively. We discuss the relaxation of assumptions in Section 7.

The DVFS processor used in our system can operate at a finite set of voltage levels $\mathcal{V} = \{V_1, ..., V_{max}\}$, each with an associated frequency $\mathcal{F} = \{F_1, ..., F_{max}\}$ and power $\mathcal{P}_{r_1} = \{P_{r_1}^1, ..., P_{r_1}^{max}\}$. The network interface has a fixed service rate and is managed by a timeout-controlled dynamic power management (DPM) scheme. This scheme is described by a constant timeout t_{out}, time to sleep tts, and a time to wake ttw. The resource has five valid states, $\{active, listen, shutdown, startup, sleep\}$, each with an associated power, $\mathcal{P}_{r_2} = \{P_{r_2}^{act}, P_{r_2}^{lst}, P_{r_2}^{dn}, P_{r_2}^{up}, P_{r_2}^{sl}\}$.

The *utilization* of a task set is the sum of each task's worst case execution time over its period. That is, the worst-case utilization can be computed as

$$U_{wc} = \sum_{i=1}^{n} \frac{wc_i}{p_i \times F_{max}}. \quad (1)$$

The average-case utilization, U_{ac}, and the best-case utilization, U_{bc}, can be computed similarly.

The timeout policy is presented in Figure 1 as a finite state machine. The resource remains in the $active$ state as long as an unserviced request is present. As soon as all requests have been serviced, the resource enters the $listen$ state and remains in the listen state until an unserviced request becomes available, or the timeout is reached, whichever comes first. If the timeout is reached, the resource then enters the $shutdown$ state for exactly tts time

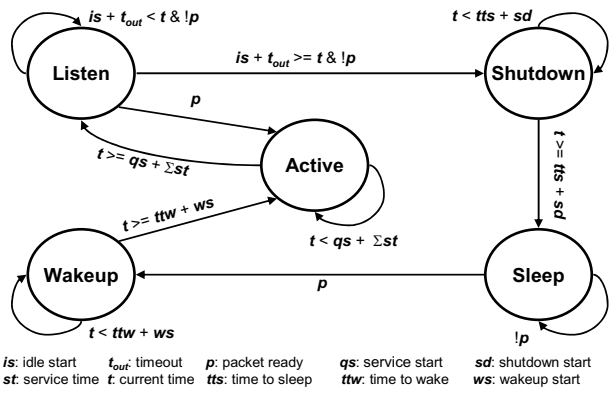


Figure 1. The network timeout policy [5].

units before entering the $sleep$ state. The resource stays in the $sleep$ state until an unserviced request becomes available, at which point it enters the $startup$ state for exactly ttw time units before once again entering the $active$ state.

The *break-even time* is the amount of time that a resource must remain in the sleep state before less energy is consumed than remaining in the idle state [15]. The break-even time can be computed as,

$$t_{be} = \max\{\frac{E_0 - P_{r_2}^{sl} \times t_0}{P_{r_2}^{lst} - P_{r_2}^{sl}}, t_0\}, \quad (2)$$

where $t_0 = tts + ttw$ and $E_0 = tts \times P_{r_2}^{dn} + ttw \times P_{r_2}^{up}$. Lu *et. al.* show in [15] that for a general workload, setting $t_{out} = t_{be}$ will consume no more than twice the energy of an optimal DPM policy.

The problem of interest, stated formally in Problem 1, is to minimize the energy of both the CPU and network simultaneously. Since many CPU-centric DVFS techniques are already available, our strategy is to first develop a network-centric DVFS technique, and then judiciously apply one of the two or a combination of the two techniques (i.e., the CPU or network centric algorithm), for a particular system.

Problem 1 *Given a system with a DVFS managed resource, r_1, an autonomous timeout-based DPM resource, r_2, and a schedulable set of tasks, $\mathcal{T}$, identify a valid voltage schedule for r_1 such that the overall system energy, $E_{r_1} + E_{r_2}$, is minimized.*

2.2 Related Work

Very little work has been done to minimize network (or other I/O device) energy through its dependence on a DVFS-ready CPU. One closely related paper by Poellabauer and Schwan does balance CPU energy with network energy [22]. This method takes into account CPU scheduling, frequency selection and packet scheduling in a cooperative way. However, it requires modification of the network card driver, CPU and packet schedulers, resulting in increased development time. Some researchers have examined the problem of energy consumption by multiple resources in various architectures, such as clustered

systems [11], sensor networks [6] and general distributed systems [7], but no real-time constraints are considered.

Several studies have shown that wireless network cards can consume a significant amount of energy and greatly reduce the battery life of mobile devices. To alleviate this problem, a number of probabilistic algorithms were proposed in [18] to allow the network card to shut down for a long period of time. Mechanisms that control packet transmissions to form bursts have also been shown to be effective. For example, [1] considers multimedia applications on portable devices where the server controls packet transmissions to clients so that the network cards at the clients' end can go into the sleep mode. It is also possible to queue messages while the network card is asleep [13], extending idle intervals. Finally, [2] proposed an adaptive framework that considers the applications' intentions on using the network and allows users to determine the trade-off level between energy consumption and system performance. Unfortunately, all of the above work fails to address any potential negative effects on the CPU.

Several papers have examined multiple CPUs as resources. For example, [9] and [29] consider independent real-time tasks executed on multiple processors. The independent task model is not able to capture the systems that we are interested in this paper, where packet release times depend on task completion times. For dependent tasks, the problem is solved off-line using mathematical programming methods (e.g., [3]), or use some sort of heuristic methods (e.g., [29]). With an off-line algorithm, it is rather difficult to respond to dynamic changes which often exist in a real-time system.

Zhuo and Chakrabarti present a technique to minimize the system-wide energy in [30]. They treat non-CPU devices as additional sources of static power that consume energy whenever a requesting task is executed on the CPU. Based on this assumption, they compute a system-efficient frequency for each task that minimizes the energy-per-cycle. Additionally, they conclude that when non-CPU devices dominate the total system energy, DVFS should not be used. Contrary to this conclusion, we show that when a device is actively managing its own power consumption (e.g., the DPM mechanism of a network interface), the proper use of DVFS can reduce the system-energy by as much as 35%, even when the CPU energy is negligible.

2.3 Look-Ahead Earliest Deadline First

In this section, we briefly review the main features of a popular on-line DVFS algorithm, called **look-ahead EDF (LaEDF)**, as it will be used in our proposed approach [21]. The major features of LaEDF are summarized in Algorithm 1. Array $cLeft$ is used to estimate the remaining cycles of the current job of each task. This value is updated at each scheduling point (Lines 4–5 and 7–8) and during execution (Lines 11–12). The function **defer()** (Line 13) attempts to push as many cycles from $cLeft$ beyond d_n as possible (Line 14). Next, the total number of cycles, s that must be executed before the next deadline, d_n is computed (Line 15). Finally, a frequency is chosen that will complete by d_n all cycles that cannot be deferred (Line 16).

Algorithm 1 Look-Ahead EDF ($\mathcal{T}$, $\mathcal{F}$)

1: **INPUT:** The task set, $\mathcal{T}$ and valid frequency set $\mathcal{F}$;
2: **select_frequency**(x):
3: Use min $F_i \in \mathcal{F}$ such that $x \leq F_i/F_{max}$;
4: **upon task_release**(T_i):
5: $cLeft_i = wc_i$;
6: defer();
7: **upon task_completion**(T_i):
8: $cLeft_i = wc_i$;
9: d_i += p_i;
10: defer();
11: **during task_execution**(T_i):
12: decrement $cLeft_i$;
13: **defer**():
14: Delay as as much of $cLeft$ past D_n as possible;
15: s = those cycles that must be executed before D_n;
16: select_frequency($s/(d_n - curTime)$);

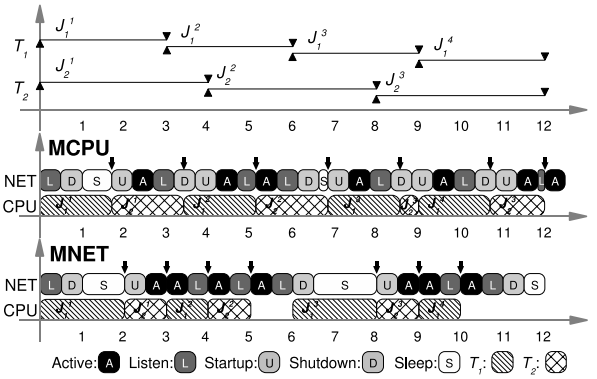


Figure 2. An example 2-task system with a voltage schedule selected to minimize CPU energy (MCPU) and network energy (MNET). The downward arrows represent packet release times.

3 Motivational Example

In this section, we use a motivational example to illustrate the potential and the challenges when multiple resources are considered to achieve lower energy consumption for the overall system.

Consider a system consisting of a CPU and a network interface device, as described in Section 2.1, with $tts = ttw = to = 0.5$, and a packet transmission time of 0.5. Figure 2 illustrates an example 2-task system, with $p_1 = 3$, $p_2 = 4$ and $wc_1 = bc_1 = wc_2 = bc_2 = 1$. Assume that at the completion of each task activation, there is one network access request. Notice that with two major power consuming devices, one could imagine a voltage scheduling policy that favors either the CPU or the network. The CPU targeted policy would strive to achieve the minimum average voltage/frequency for all tasks [28], while the network targeted policy would strive to maximize the time in sleep

mode of the network device. We refer to the CPU centric policy as MCPU and the network centric policy as MNET.

The schedule resulting from each policy is given in Figure 2. Notice that jobs in MCPU do not complete execution until time 12, while those in MNET finish at 10. Additionally, notice that the network is in sleep mode nearly 3 times more in MNET than in MCPU.

One important question that must be answered is how to choose between MCPU and MNET. The answer to this question depends on the relative contribution of each device to the total system power consumption, and on the timing parameters of the task set in question. In the example from Figure 2, when the CPU dominates the network card, MCPU reduces the energy consumed by 50% when compared to MNET. (We say that the CPU dominates the network card if the CPU's active power is much higher than that of the network card and vice versa). However, when the network dominates, MNET will reduce the system energy by as much as 24% when compared to MCPU [1].

4 MNET and MCPU algorithms

Based on the observations in the previous section, superior MCPU and MNET algorithms play a critical role in saving system-wide energy for the architecture under consideration. In this section, we introduce one algorithm for MNET and one for MCPU. To reduce run-time overhead, we strive to make the algorithms simple to implement and share as many common operations as possible.

4.1 MNET: Timeout Aware Scheduler

From the motivational example in Section 3, one can already see that energy can be saved at the network by arranging packets into bursts. This will maximize the time available for the network to remain in the sleep state and reduce the overhead due to transitioning between sleep and active modes. Since the CPU is DVFS capable, it can be exploited to adjust the release time of the packets. With this idea in mind, we introduce two heuristics that guide the MNET algorithm.

1. If the current job can produce a packet by executing at the maximum processor frequency before the network timeout occurs, then doing so will extend the length of the on-going packet burst, thus increasing the length of the next sleep interval.

2. If the current job cannot produce a packet before the network timeout occurs, then executing at the minimum feasible processor frequency will (ideally) merge the next packet with the subsequent burst, thus extending the length of current sleep interval.

[1]For this example, the CPU model is an ARM11 [4], while the network model is a D-Link Personal Air: Wireless USB Bluetooth Adapter, DBT-120 [10]. Details of these models can be found in 6.2 under **Simulation Setup**.

The proposed MNET algorithm, called Timeout Aware Scheduler (TAS), is given in Algorithm 2. Here, the CPU is made aware of the state of the network-interface packet queue with the help of the operating system (Lines 2–6). The time at which the network goes to sleep is computed at Line 8, while the remaining cycles before the currently executing task is completed are estimated at Line 10. Finally, which heuristic to apply is determined by comparing the expected packet release time at the maximum frequency with the network sleep time at Line 11. If the expected packet release time at the maximum frequency is greater or equal to the network sleep time, i.e., situation (2) above, the minimum (Line 12) frequency is selected. Otherwise, the maximum (Line 14) frequency is selected.

We compute the minimum frequency using Algorithm 1. The rationale is to lower the CPU frequency to the point where it is most energy efficient for the CPU while extending the sleep interval of the network interface. A key feature of LaEDF is that it identifies the minimum possible frequency at which to execute the active task without violating future deadlines. Though there are other algorithms exhibiting the same feature, LaEDF is rather simple to implement and has comparable performance in terms of energy savings [12]. A significant amount of detail is hidden

Algorithm 2 Timeout Aware Scheduler($\mathcal{T}$, $\mathcal{F}$, R)

```
1:  INPUT: The task set, T, a valid frequency set F and
    the resource being managed, R;
2:  idle_start(R):
3:      if(state(R)=idle, shutdown or sleep):
4:          return last time at which R became idle;
5:      else:
6:          return next time at which R will become idle;
7:  upon job release or finish time t:
8:      t_sleep = idle_start(R) + t_out;
9:      Let T_i be the active task;
10:     q_i = avg-case cycles before next request of T_i to R;
11:     if(q_i/F_max + curTime ≥ t_sleep):
12:         select frequency based on LaEDF;
13:     else:
14:         set frequency to F_max;
```

in TAS by Line 10. Although we assume that packets are consistently generated at the end of a job's execution, the actual system will require monitoring of the tasks' past performance to identify if and when a packet is produced. In Sections 6.1 and 7 we offer more details on how this is accomplished.

4.2 MCPU: Limited Look-Ahead EDF

For the CPU-centric DVFS algorithm, there are many possible choices. For the same reason that we chose LaEDF in TAS, we choose LaEDF as the basis of our MCPU algorithm. However, the greedy nature of LaEDF that was an advantage for TAS is now a key drawback for MCPU. Specifically, LaEDF applies all available slack to the currently executing job. When the active job requires close to its worst-case execution cycles to complete, this causes sub-

sequent jobs to execute at elevated CPU frequencies, which leads to less energy savings.

We control the greediness of LaEDF by modifying LaEDF in a way similar to the method proposed in [17] for fixed-priority systems. The idea is to maintain a CPU frequency no less than the frequency required to meet the *average-case* processor utilization of the current task set. We call this an average-case limiter. With this limiter, it is less likely that the currently executing job takes so much slack that it causes later jobs to run at a much higher frequency. The average-case utilization can be computed off-line from measurements of the task set in question and/or estimated on-line by recording the completion times of jobs. We refer to this technique, given in Algorithm 3, as Limited Look-Ahead EDF (LLE).

Algorithm 3 Limited Look-Ahead EDF($\mathcal{T}$, $\mathcal{F}$)

1: **INPUT:** The task set, $\mathcal{T}$ and valid frequency set $\mathcal{F}$;
2: **limit**($freq$):
3: $U_{ac} = \sum_{i=1}^{n}(ac_i/(p_i \times F_{max}))$;
4: $freq$=max($freq$, $F_{max} \times U_{ac}$);
5: **return** $freq$;
6: **select_frequency**(x):
7: Use min $F_i \in \mathcal{F}$ such that $x \leq F_i/F_{max}$;
8: F_i = limit(F_i);
9: **include remaining functions from LaEDF**;

5 Algorithm Selection

To maximally reduce the total system energy consumption, a key challenge is how to balance the conflicting goals of MCPU and MNET (i.e., LLE and TAS). In this section, we begin by introducing a parameter for capturing the relative importance of the CPU and the network card in terms of saving energy. Then, we present two methods to balance MCPU and MNET, an off-line selection approach that uses known system specifications, and a hybrid approach that can adapt to on-line changes in utilization and task-set composition.

Consider a system with two resources, r_1 and r_2. Let $P_{r_1}^{max}$ be the peak power consumption of r_1 and let $P_{r_2}^{max}$ be the peak power consumption of r_2.

Definition 1 *The value α represents the degree of dominance of r_2 over r_1 in terms of power, and is given by,*

$$\alpha = \frac{P_{r_2}^{max}}{P_{r_2}^{max}+P_{r_1}^{max}}. \quad (3)$$

To see the significance of α, let us examine the overall system energy, E_s,

$$E_s = E_{r_1} + E_{r_2}, \quad (4)$$

where E_{r_1} and E_{r_2} is the energy of r_1 and r_2, respectively. E_s can be re-written as,

$$E_s = \int_0^T P_{r_1}(t)dt + \int_0^T P_{r_2}(t)dt, \quad (5)$$

Table 1. Alpha value of various CPU/wireless adapter pairs. r_1 is the CPU while r_2 is the wireless adapter [4, 10, 20, 23].

Wireless Adapters \ Mobile Processors	Pentium-M	ARM Cortex-A8	ARM11	ARM922T	ARM996HS
DBT 120 PS	0.01	0.32	0.54	0.75	0.98
HBTC1	0.01	0.52	0.72	0.87	0.99
Aironet 350	0.02	0.53	0.73	0.88	0.99
Orinoco Gold	0.05	0.78	0.89	0.96	1.00
Air Station G54	0.05	0.78	0.90	0.96	1.00
DWL-AG660	0.05	0.80	0.91	0.96	1.00
EW-7317Ug	0.06	0.81	0.91	0.96	1.00
G132_ds	0.08	0.86	0.93	0.97	1.00
DWL-G650M_ds	0.08	0.87	0.94	0.98	1.00

where T is the system lifetime and $P_{r_1}(t)$ and $P_{r_2}(t)$ represent the instantaneous power of r_1 and r_2, respectively. Factoring out the respective peak power gives

$$E_s = P_{r_1}^{max}\int_0^T \frac{P_{r_1}(t)}{P_{r_1}^{max}}dt + P_{r_2}^{max}\int_0^T \frac{P_{r_2}(t)}{P_{r_2}^{max}}dt. \quad (6)$$

Hence, the system peak power consumption, P_s^{max}, is equal to

$$P_s^{max} = P_{r_1}^{max} + P_{r_2}^{max}. \quad (7)$$

Substituting (7) into (6) gives

$$E_s = (P_s^{max} - P_{r_2}^{max})\int_0^T \frac{P_{r_1}(t)}{P_{r_1}^{max}}dt + P_{r_2}^{max}\int_0^T \frac{P_{r_2}(t)}{P_{r_2}^{max}}dt. \quad (8)$$

Finally, factoring out the maximum system energy yields

$$E_s = P_s^{max}\left[(1-\alpha)\int_0^T \frac{P_{r_1}(t)}{P_{r_1}^{max}}dt + \alpha\int_0^T \frac{P_{r_2}(t)}{P_{r_2}^{max}}dt\right]. \quad (9)$$

Equation (9) captures quantitatively the impact of the energy consumption of each resource on the overall system energy. It also sheds light on how to balance the energy demands of different resources. Based on Definition 1, Table 1 gives a range of possible α values from combinations of mobile processor cores and wireless network adapters. As one might expect, combinations exist that produce many varying α values.

To see what factors may impact our selection between MCPU and MNET, we examine the example task set from Figure 2 in more detail. Figure 3 shows the relative energy consumption of executing this task using MCPU and MNET as α value varies. Clearly, MCPU performs better for smaller α and MNET is better for larger α. Since α is a hardware dependent parameter, it seems that one could easily select either MNET or MCPU when the system components are determined. However, such a simplistic approach could lead to quite undesirable results.

The choice of which algorithm to use also depends largely on the timing parameters in the task set including periods, deadlines and actual execution cycles. Examining Figure 3 more carefully, one can see that the two sets of

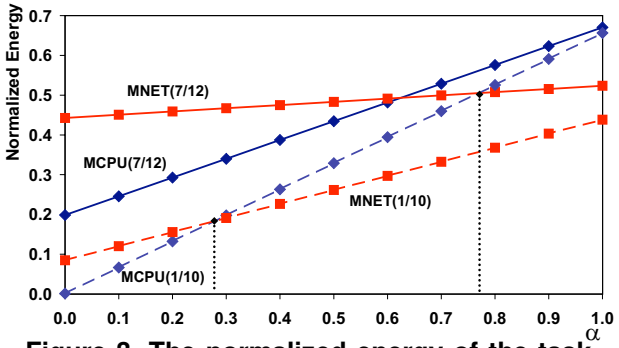


Figure 3. The normalized energy of the task set from Figure 2, with a CPU utilization of 7/12 (solid lines) and 1/10 (dashed lines). The vertical lines indicate the intersection point of each set.

lines corresponding to two different CPU utilization values (due to varying execution cycles) of the same task set intersect at two different α values. In this case, the choice between MCPU and MNET could be reversed if the system α is anywhere in the range [0.27, 0.63]. Therefore, to determine whether to use TAS or LLE, significant amount of off-line analysis is required.

We give a simple off-line approach below which considers both the α value and the average utilization of a system.

1. Estimate the energy consumption of MCPU and MNET on the target system (CPU model, network model and task set), assuming that each task/packet requires its average case execution cycles/transmission time.

2. Select the algorithm that consumes less energy.

This straightforward technique, which we refer to as off-line selection (OLS), will work well in situations where the task set and system architecture is not expected to change significantly on-line. It can be implemented in a variety of ways, including event-driven simulation or system prototyping.

5.1 Hybrid Approach

When significant dynamic changes are expected, we propose the following technique to balance the conflicting goals of MNET and MCPU. Let F_{r_1} be the processor frequency selected by MCPU to minimize the energy consumed by the processor, and F_{r_2} be the processor frequency selected by MNET to minimize the energy consumed by the network interface. To balance the energy consumption between the two resources, the actual processor frequency is selected by using α to average the two frequencies, i.e.,

$$F_{used} = (1 - \alpha)F_{r_1} + (\alpha)F_{r_2}. \quad (10)$$

We refer to the frequency selected by using Equation (10) as HYB. Because HYB averages the frequency selected by TAS with that selected by LLE, it can be considered a *conservative* selection in that we expect it to do no better than the best choice, but *no worse* than the worst choice. The beauty of this approach is its simplicity and its wide applicability. In the next section, we will evaluate each of the proposed techniques on a variety of system specifications.

6 Experimental Results

To validate our proposed techniques, we have implemented them in an OS running on a hardware platform. We have also performed a large number of simulations to investigate the various behaviors of these techniques. In this section, we first describe how our techniques were implemented in an actual system as well as sample results. Next, we discuss results generated through simulation.

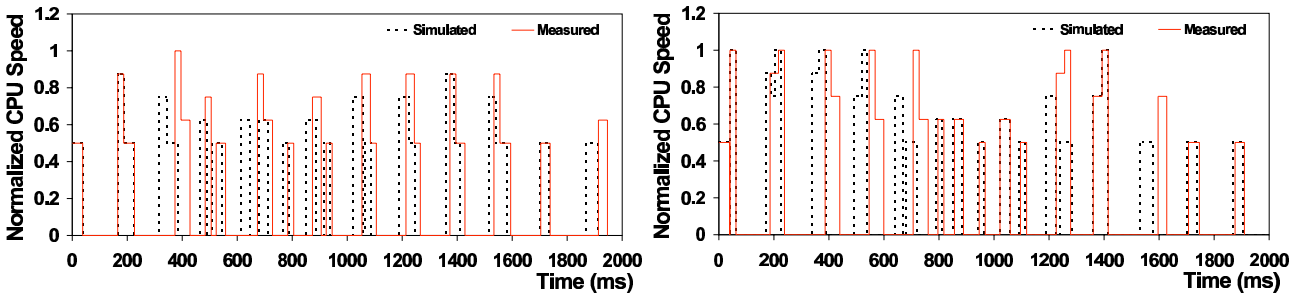
6.1 Implementation

By implementing the proposed algorithms in an actual system, we verified that the frequencies selected during simulation reflect reality, increased our confidence in the simulated results and highlighted some potential complications. We chose to implement only TAS and LLE from Section 4, because OLS selects either TAS or LLE off-line and the hybrid algorithm uses a weighted average of the frequencies selected by both. We felt that this was sufficient from a frequency-verification standpoint.

The target system was a Gateway 405R0G Laptop running Linux, kernel version 2.4.19, powered by an Intel Pentium M [20] processor and equipped with an Orinoco Gold, 11 Mbps wireless network card [23]. This configuration was chosen to validate the extensibility of our proposed algorithms to common energy constrained computing systems. OS modifications include the implementation of an EDF scheduler, the DVFS algorithms and a system monitoring tool that provides the basis for the prediction of future resource accesses.

Resource Access Monitoring: A kernel module that tracks all process resource accesses (e.g., I/O activity such as disk and network reads and writes) has been implemented to support energy management decisions. The monitored information includes task invocations, execution times, I/O operations (including data size, socket or file descriptors, etc.) and memory accesses. This information is collected for each task and stored in memory, accessible by our EDF scheduler or by any task via the /proc interface. To reduce the resource needs of the monitoring module, the amount of information stored can be limited, e.g., for a periodic real-time task, only the history of the most recent hyperperiod may be of interest. Note that the monitoring tool is an architecture independent implementation, i.e., process traces can easily be obtained for different execution platforms.

Scheduler Implementation: The standard Linux scheduler was replaced by an EDF scheduler, where a task with superuser rights can promote itself or any other task to a real-time (RT) task by specifying a period and worst-case execution time. This can be achieved via the /proc virtual file system provided in Linux. At each scheduler invocation, the current task set and the resource access activity of all RT tasks are evaluated (obtained through the resource



(a) Normalized CPU Speed vs. Time, LLE

(b) Normalized CPU Speed vs. Time, TAS

Figure 4. The normalized speed selected vs. time for a two-task system, with $T_1 = (p_1 = 150ms, wc_1 = 96MCycles)$ and $T_2 = (p_2 = 170ms, wc_2 = 120MCycles)$.

access monitoring tool) to predict future resource accesses and to appropriately select frequency/voltage levels.

Results: A comparison of the frequencies selected by the implemented algorithms in our experimental system with those of the simulation is shown in Figure 4. The implementation frequencies are found to closely match the simulation in most cases. However, there do exist a few differences in the frequency selections where the implementation chooses to execute at the next higher frequency. Upon closer analysis, this difference was found to be directly attributable to the latencies experienced in the invocation of the scheduler. The scheduler latency, which resulted from the use of a non-real-time Linux kernel, was found to vary from 1 to 30 ms. Such latencies in scheduling real-time tasks ready for their release were found to occur whenever background tasks essential for the proper functioning of the operating system (e.g., the swapper process being run to manage the page cache) were executed. As a result of such unavoidable latencies, the real time tasks at certain instances were scheduled for execution later than their actual release times, which in some cases altered the selected frequency.

6.2 Simulations

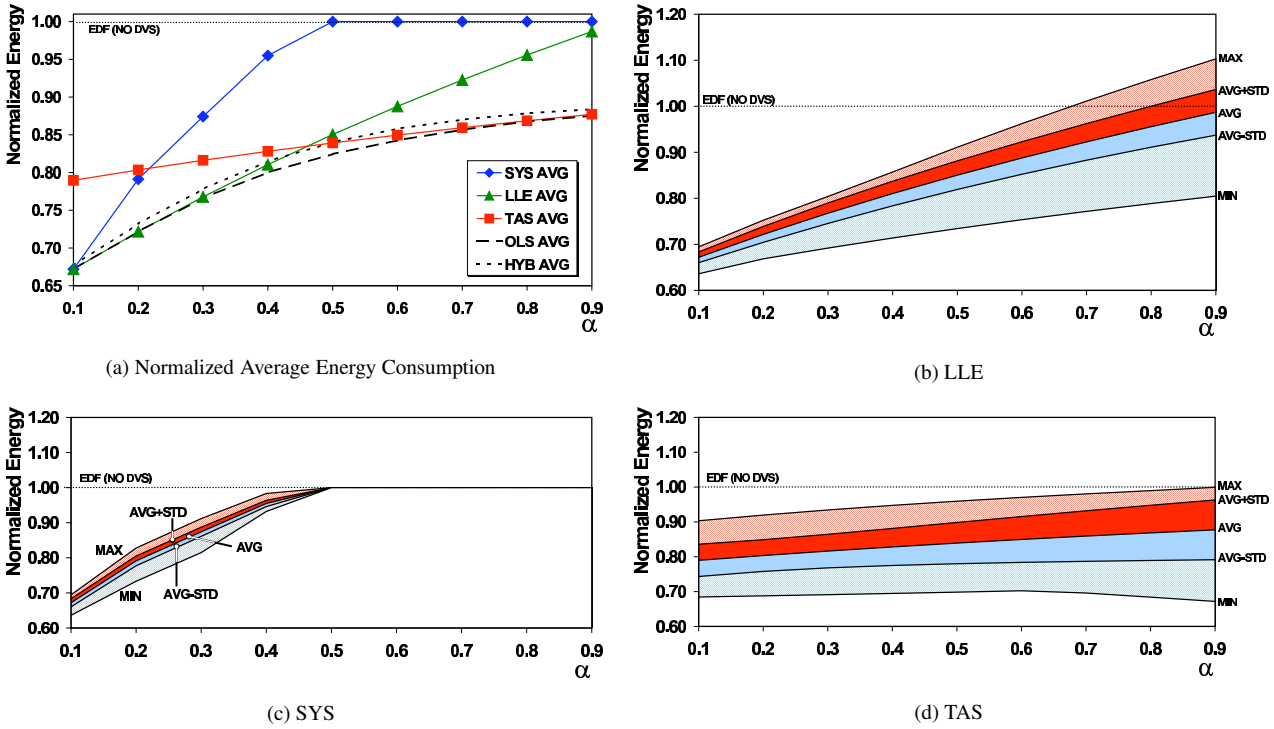
Having calibrated our simulation with an actual implementation, we felt more comfortable exploring a variety of architectures and task sets that would have otherwise not been possible, due to resource limitations and time constraints. Next, we describe the algorithms evaluated, system parameters and results.

Algorithms: In this section we examine the energy savings achieved by five algorithms. The first two are the CPU-centric and network-centric algorithms, LLE and TAS, as introduced in Section 4. The next two algorithms are the proposed off-line selection (OLS) and hybrid (HYB) techniques. The last one, called SYS, is the same as LLE, with the additional limitation that the CPU frequency selected cannot be less than the system-efficient frequency as defined in [30]. The system-efficient frequency concept was introduced in an attempt to minimize the system-wide energy when there are other energy-consuming devices in the system in addition to the CPU [30]. With our system model, there is only one device (i.e., the network card) plus the CPU and the device is used by every task. Thus, the system-efficient frequency is set assuming that the network card is awake during the execution of any task.

Simulation Setup: The experiments were run on 100 randomly generated task sets of 10 tasks each with periods in the range [1, 200] ms. The worst case execution cycles were assigned randomly such that a 100% utilization was reached. For a particular experiment, the worst-case execution cycles of each task were scaled uniformly to achieve the desired utilization. The packet size of each task was assigned in a similar manner.

The processor model is representative of an ARM11 processor core with 32 voltage levels, evenly distributed in the range [0.8,1.3] V, with corresponding frequencies in the range [138, 550] MHz and power levels in the range [34, 250] mW [4]. The network model is representative of the D-Link Personal Air: Wireless USB Bluetooth Adapter, DBT-120 [10], with $P_{r_2}^{act} = 190$ mW, $P_{r_2}^{sl} = 129$ μW and $P_{r_2}^{lst} = P_{r_2}^{dn} = P_{r_2}^{up} = 165$ mW. We assume a time overhead of 50μs to enter the sleep state and a wake-up overhead of 250μs. The power consumed during shutdown and wake-up was assumed to be equal to the idle power and the timeout was set to the break-even time of 300μs. To achieve a desired α, the relative power consumption of the CPU and network were scaled according to Definition 1.

Simulation Results: The energy savings of each algorithm was examined when varying the CPU and network utilization from 0 to 100% and α from 0.1 to 0.9. A snapshot of the results with a CPU utilization of 40% and a network utilization of 10% is given in Figure 5. Figure 5(a) shows the average-case energy savings exhibited by each algorithm. Below $\alpha = 0.45$, LLE is the best algorithm on average, by as much as 15% when compared to TAS. After $\alpha = 0.45$, the situation reverses, with TAS outperforming LLE by as much as 13%. SYS performs as well as LLE when the CPU dominates, but quickly saturates to the equivalent non-DVS system when $\alpha = 0.5$. The saturation occurs due to the assumption that the network card must be on whenever a packet-sending task is active. Note that this assumption is not appropriate when the network card has its own power-saving timeout mechanism. As expected, OLS tracks the minimum-energy envelop formed by



(a) Normalized Average Energy Consumption

(b) LLE

(c) SYS

(d) TAS

Figure 5. Normalized energy vs. α with a CPU utilization of 40% and a network utilization of 10%.

TAS and LLE. The energy of HYB closely tracks the better of the two algorithms but tends to achieve less energy savings, on average.

Figures 5(b)–5(d) show the best, average and worst case energy of LLE, SYS and TAS, respectively, when considering different CPU utilization. While TAS will, on average, reduce the system energy by only 13% when the network dominates, savings up to 22% are not uncommon and 35% are possible. LLE, on the other hand, is likely to both save and waste energy when α is large. As expected, SYS cannot save network energy (by extending sleep network intervals) as α increases, because the system-efficient frequency saturates to the maximum processor frequency when the network dominates.

In general, as the CPU utilization increases, the potential savings when α is small decreases. The same is true for network utilization when α is large. For a small α with a CPU utilization that is very small (less than 10%), all algorithms saturate to the same value, due to the lower CPU-frequency bound. For a very large CPU utilization, task deadline constraints prevent the use of DVS, resulting in reduced energy savings for all algorithms over all α values.

Figure 5(a) seems to suggest that selecting either LLE or TAS based on α alone is a good idea, but Figures 5(b) and 5(d) clearly show a high degree of variability in energy saving potential when the CPU utilization varies. Figure 6 illustrates this variability by showing the percent difference in energy between LLE and TAS over various mid-range α values and several CPU utilization values. Specifically, the vertical axis of Figure 6 represents $(E(algorithm) - E(LLE))/E(LLE)$, where "algorithm" can be either TAS or OLS. Although the energy difference, on average, is not more than 5%, for a particular task set, the penalty for choosing the wrong algorithm can be greater than 10% (see the top-most and bottom-most curve), even at the crossing point of the average case (those points where the average percent difference is zero).

The average percent difference between OLS and LLE is also displayed (the dashed line) in Figure 6. Though not shown explicitly, the minimum for OLS is zero (which matches those cases where LLE is the best choice) and the maximum will be equivalent to the maximum of TAS. Clearly OLS is better than an arbitrary selection of TAS or LLE when the CPU utilization and task composition is not expected to change significantly on-line. However, if the system is more dynamic, it may be better to use a conservative approach, such as HYB, rather than face an energy penalty of up to 16% as indicated by the top-most and bottom-most curve.

To examine the behavior of HYB, we introduce the notion of a **hit** and the **hit ratio**. We say that a task set has a hit if the algorithms satisfy $Energy(HYB) < \max\{Energy(LLE), Energy(TAS)\}$. The hit ratio is simply the ratio between the number of task sets which have a hit and the total number of task sets. For the randomly generated task sets, the hit ratio data is provided in Figure 7(a) (the dashed curve).

The significance of the hit ratio is that it indicates the opportunities when HYB would save more energy if a wrong decision is made for the task set. For $\alpha \in [0.0, 0.2]$ and

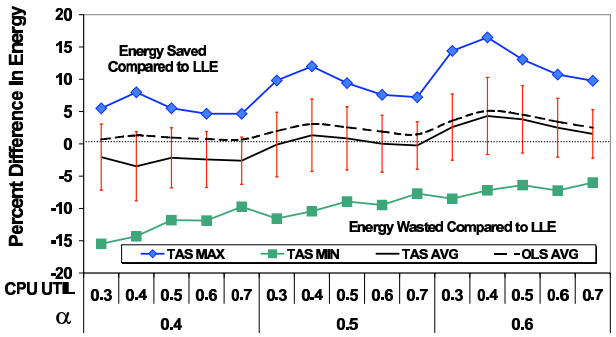


Figure 6. The percent difference between TAS and LLE for various mid-range α and CPU-utilization values. Error bars show the average energy savings, $\pm$ the standard deviation.

$\alpha > 0.9$, the hit ratio is 100%. Over mid-range α values, the hit ratio is no lower than 73%, which occurs when the CPU utilization and α are both 0.6. The triangle symbol curve indicates the fraction of tasks that have hits when LLE is the worst choice, while the diamond symbol curve represents the fraction of tasks that hit when TAS is the worst choice. Obviously, for those α values where either LLE or TAS is never wrong, the algorithm selection is clear, even if the utilization and task set vary. However, for $\alpha \in [0.4, 0.6]$, the odds are good that a wrong decision will be made, which makes HYB a better choice. Figure 7(b) shows the hit benefit and miss penalty associated with HYB for various α and CPU utilization combinations. Here, a benefit or penalty is equal to the percent difference between $Energy(HYB)$ and $\max\{Energy(LLE), Energy(TAS)\}$. From Figure 7(a) and 7(b) we can see that not only is the penalty less than the benefit in most cases (comparing the magnitudes of the curves below 0 and those above zero), but a hit is more likely in all cases.

7 Conclusions and Future Work

Contrary to existing work on system-wide energy optimization, we have demonstrated that it is indeed possible to save energy using DVFS, even when the CPU is not the dominant consumer of energy. Specifically, when a secondary device employs an independent DPM strategy to minimize energy locally, DVFS can be applied to both minimize CPU energy and shape the traffic of requests to the secondary device such that sleep intervals are maximized. This is particularly true when requests to the DPM resource are collected into a small number of bursts within each task instance (as is typically the case with network accesses) rather than being distributed evenly throughout the task. Similar results could also be applied to disk and memory accesses if caching / prefetching is used effectively.

Based on the observation above, we developed a network-centric algorithm called Timeout Aware Scheduler (TAS) that achieves energy savings of up to 35% when the network-interface energy dominates that of the CPU. When coupled with a representative CPU-centric algorithm and off-line selection, a network-aware DVFS policy that can balance the CPU and network energy is produced. When the system is highly dynamic and it is not clear off-line which algorithm to choose, our proposed hybrid approach offers a conservative alternative to off-line selection.

The presented work is based on a number of assumptions which will be relaxed in our future work. Most notably, our work has focused on transmitting packets while ignoring any incoming data. TAS will be extended to adapt its prediction of future network transmissions and timeouts to consider incoming traffic as well. TAS will consider the behavior of the Distributed Coordination Function (DCF) of the 802.11 protocol to 'align' network transmissions such that they occur when the network card is already awake to receive data from the base station.

Although the on-line hybrid algorithm is, on average, more efficient in terms of energy consumption than selecting either MNET or MCPU statically, more variables need to be considered. This includes CPU utilization, Network utilization and variable packet release times (i.e., other than at the completion of a job).

Finally, in this paper, the CPU was the 'manageable' resource and the network was a passive resource, i.e., the presented algorithms only monitor the behavior of the network card, but do not control it. Our future work will extend the TAS approach to an algorithm that actively modifies network parameters such as timeout values.

References

[1] A. Acquaviva, T. Simunic, S. Roy, and V. Deolalikar. Remote power control of wireless network interfaces. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization, and Simulation*, pages 369–378, September 2003.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189, September 2003.

[3] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2004.

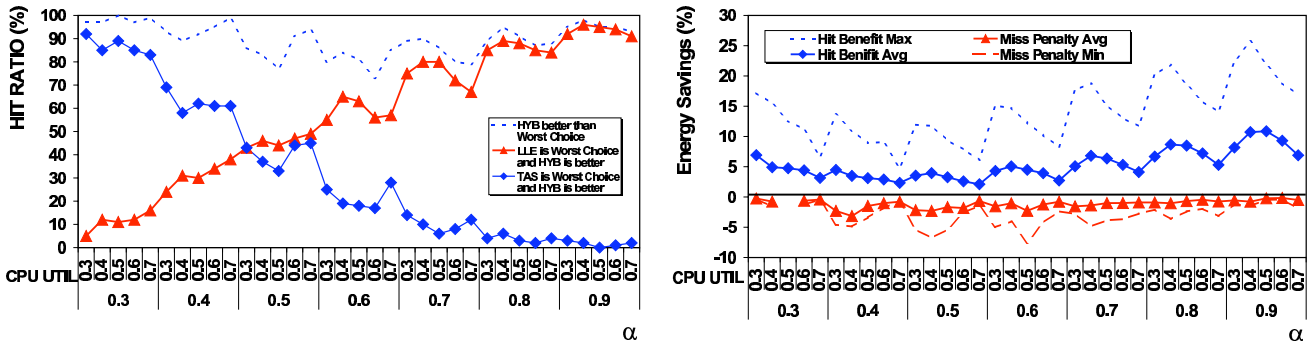
[4] ARM CPU Cores. `http://www.arm.com/`.

[5] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, jun 2000.

[6] A. Boulis and M. Srivastava. Node-level energy management for sensor networks in the presence of multiple applications. *Wireless Networks*, 10(6):737–746, nov 2004.

[7] K. W. Cameron, R. Ge, and X. Feng. High-performance, power-aware distributed computing for scientific applications. *IEEE Computer*, 38(11):40–47, nov 2005.

[8] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. In *Leakage-aware energy efficient scheduling of real-time tasks*, pages 408–417, 2006.



(a) Hit ratio of HYB

(b) Hit benefit and miss penalty of HYB.

Figure 7. Hit ratio and benefit / penalty measurements of the hybrid algorithm. In 7(a), the dashed line represents the HYB hit ratio, while the two solid curves represent the percentage of task sets where a hit occurs and either LLE is the worst choice or TAS is the worst choice. In 7(b), the hit benefit is the percentage of energy saved by a hit, while the miss penalty is the percentage of energy wasted by a miss.

[9] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. *Int'l Conf. on Parallel Processing*, 2005.

[10] D-Link Wireless Adapters. online- `http://www.dlink.com`.

[11] E. J. Kim, G. M. Link, K. H. Yum, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and C. R. Das. A holistic approach to designing energy-efficient cluster interconnects. *IEEE Transactions on Computers*, 54(6):660–671, june 2005.

[12] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 219–228, 2002.

[13] R. Kravets and P. Krishnan. Power management techniques for mobile communications. In *In Proceedings of the 4th Conference on Mobile Computing and Networking MOBICOM'98*, oct 1998.

[14] C.-Y. T. L. Leung and X. Hu. Exploiting dynamic workload variation in low energy preemptive task scheduling. In *Design Automation and Test in Europe (DATE)*, pages 634–639, 2005.

[15] Y.-H. Lu and G. D. Micheli. Comparing system-level power management policies. *IEEE Design & Test of Computers*, 18(2):10–19, mar 2001.

[16] B. Mochocki, X. S. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-Aided design (ICCAD)*, pages 726–731, nov 2002.

[17] B. C. Mochocki, X. S. Hu, and G. Quan. Practical on-line dvs scheduling for fixed-priority real-time systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, March 2005.

[18] S. PalChaudhuri and D. B. Johnson. Power mode scheduling for ad hoc networks. In *Proceedings of the International Conference on Network Protocols (ICNP 2002)*, pages 192–193, November 2002.

[19] A. E. Papathanasiou and M. L. Scott. Energy Efficiency through Burstiness. In *Proc. of the 5th IEEE Workshop on Mobile Computing Systems and Applications*, October 2003.

[20] Intel Pentium M Processor Datasheet. `http://download.intel.com/design/mobile/datashts/25261203.pdf`.

[21] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP)*, pages 89–102, 2001.

[22] C. Poellabauer and K. Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 48–55, may 2004.

[23] ORiNOCO Classic Gold PC Card. online- `http://www.handhelds.org/Compaq/iPAQH3600/iPAQ\_H3600.html`.

[24] D. Qiao, S. Choi, A. Jain, and K. G. Shin. MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 802.11a/h. In *Proc. of the ACM/IEEE Intl. Conference on Mobile Computing and Networking*, September 2003.

[25] G. Quan and X. S. Hu. Minimum energy fixed priority scheduling for variable voltage processors. *IEEE Trans. on ICCAD*, 22(8), August 2003.

[26] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. Srivastava. E2WFQ: An Energy Efficient Fair Scheduling Policy for Wireless Systems. In *Proc. of the Intl. Symposium on Low-Power Electronics and Design*, August 2002.

[27] E. Uysal-Biyikoglu, B. Prabhakar, and A. E. Gamal. Energy-efficient packet transmission over a wireless link. *IEEE Transactions on Networking*, 10(4):487–499, aug 2002.

[28] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 374–382, oct 1995.

[29] D. Zhu, N. AbouGhazaleh, D. Mossé, and R. Melhem. Power aware scheduling for and/or graphs in multi-processor real-time systems. In *Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*, page 593, 2002.

[30] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 2005 Design Automation Conference (DAC)*, pages 628–631, jun 2005.