

# Energy-Conscious Co-Scheduling of Tasks and Packets in Wireless Real-Time Environments

Jun Yi, Christian Poellabauer, Xiaobo Sharon Hu, Jeff Simmer  
Department of Computer Science and Engineering  
University of Notre Dame  
Notre Dame, IN, USA  
{jyi, cpoellab, shu, jsimmer}@nd.edu

Liqiang Zhang  
Department of Computer and Information Sciences  
Indiana University, South Bend  
South Bend, IN, USA  
liqzhang@iusb.edu

## Abstract

Exclusive access to the wireless medium, e.g., as provided by bandwidth-reservation mechanisms, limits contention and therefore is capable of providing effective real-time support to periodic communications. Furthermore, to preserve energy, wireless cards can be powered down between periodic accesses without loss of data. However, packet schedulers must be aware of the limited communication opportunities to ensure that packets are transmitted before their deadlines, CPU schedulers must execute jobs such that the packets generated by these jobs are available for transmission in time, and DVS algorithms must choose processor speeds such that job execution and therefore packet generation are not unduly delayed. This paper proposes a co-scheduling approach to integrate CPU, network, and energy management for wireless real-time systems that rely on bandwidth reservations. Both simulation and experimentation indicate significant improvements in meeting packet deadlines (up to 40%) with only small increases in overall energy consumption (less than 10%) compared to the state of the art.

## 1. Introduction

**Wireless Real-Time Applications.** An increasing number of real-time applications are deployed in wireless environments, including networked embedded control systems, robots and robot teams, mobile multimedia, wireless sensor and actuator networks, and generally many types of cyber-physical systems. Wireless networks are inherently broadcast media, leading to contention and interferences, which exacerbate the challenge of satisfying real-time constraints. Numerous previous research efforts have studied and proposed methods to resolve contention when multiple nodes require access to the medium. But existing medium access techniques have their own pros and cons when applied to wireless real-time systems. For example, CSMA is simple, robust, and scalable, but also non-deterministic and thus incapable of providing effective QoS support to periodic communications often

found in wireless real-time systems. Recently, reservation-based channel access protocols that explicitly allow wireless devices to negotiate channel access intervals have been receiving increasing attention. For example, in ad-hoc networks, coordinated sleep mechanisms have been designed [15] to allow wireless devices to coordinate medium access with their neighbors. Similarly, in infrastructure-based systems, wireless end devices can coordinate medium access with base stations using protocols such as IEEE 802.11e [5]. These methods, in some sense, build TDMA-like deterministic channel resource sharing mechanisms above the non-deterministic CSMA-based medium access platform.

**Bandwidth Reservations.** Borrowing the terminology used in 802.11e [5], every wireless device reserves a periodic time interval, called *Service Period (SP)*, where it has exclusive access to the wireless medium (Figure 1). The period between consecutive SPs is called the *Service Interval (SI)*. We call the relationship  $SP/SI$  the bandwidth ( $BW$ ) given to a wireless device.

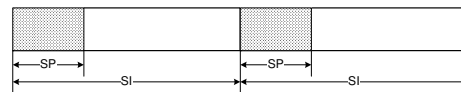


Figure 1. A wireless device's SP and SI parameters

There are two main benefits of bandwidth reservation. First, transmission errors and delays caused by contention are reduced, which supports our goal of meeting (end-to-end) real-time communication constraints. Second, during *idle intervals* (i.e., between two consecutive SPs), a wireless device will neither send nor receive data, thereby allowing it to power down its wireless card to preserve energy.

**Integrated Resource Management.** Systems relying on bandwidth reservations face a number of challenges, which can be best addressed with an *integrated* approach to managing resources such as CPU, network, and energy. In conventional operating systems, CPU schedulers (and DVS

mechanisms if available) operate independently from packet schedulers. That is, while schedulability analyses and admission tests can ensure that job deadlines are met, the packets generated by these jobs may miss their packet deadlines nevertheless. This is particularly the case in reservation-based networks, where access to the wireless channel is limited to small periodic intervals. As a consequence, this work moves the focus from *task deadlines* to *packet deadlines* to coordinate scheduling of tasks, packets, and CPU speeds. Task deadlines are essential for schedulability and admission tests and the prioritized scheduling of tasks. However, in distributed real-time systems, the primary goal is to meet end-to-end packet deadlines. Toward this end, we assume that each generated packet on a wireless device has a local packet deadline, e.g., derived from application scenarios, experience, or user-specified end-to-end deadlines [13], [16]. For example, in wireless multi-hop sensor networks, each wireless device along a route can derive a local packet deadline from an end-to-end deadline, where this local deadline may be adjusted frequently to adapt to changes in congestion or missed deadlines. Further, this work assumes that a device has already negotiated its SI and SP parameters (e.g., as proposed in [12]), i.e., the focus of this paper is on a novel *co-scheduling* approach for tasks, packets, and energy, to ensure that (a) packet deadlines are met, while (b) energy consumption is kept low.

Specifically, the proposed co-scheduling approach includes the following coordinated processes:

- **Packet Scheduling:** Given the negotiated access periods (SP, SI), a packet scheduler must be aware of such limited communication opportunities to ensure that packets are transmitted before their deadlines.
- **CPU Scheduling:** CPU schedulers must select jobs such that job execution, and therefore the generation of real-time packets by these jobs, will be completed in time to allow the packet scheduler to transmit the packets in the limited channel access periods.
- **Speed Scheduling:** A DVS algorithm must choose frequencies and voltages such that energy consumption at the CPU is reduced without delaying job execution (and therefore packet generation) beyond the packet's real-time deadline.

## 2. Related Work

Bandwidth reservations and synchronized wake-up approaches among wireless devices have found increasing attention, both in managed and ad-hoc networks. Contention-based communication in wireless networks makes it difficult to provide acceptable real-time services, therefore, the 802.11e working group has proposed the hybrid coordination function (HCF) [5] to provide QoS support for real-time transmissions over 802.11 WLANs. HCF supports both

contention-based channel access, known as enhanced distributed coordination access, and controlled channel access (HCCA). A few previous efforts have proposed real-time scheduling algorithms based on HCCA [4]. However, they only consider packet queuing and scheduling, disregarding the effects of task and speed scheduling on communication latencies in reservation-based environments. In contrast, this paper comprehensively considers the inter-dependencies of task, packet, and speed scheduling to enable energy-efficient real-time communications in networks with limited, but exclusive, transmission opportunities.

Energy management has received a tremendous amount of attention, considering energy preservation at different layers of a system and involving different resources [1], [14]. At the network level (and similarly at the disk level), Dynamic Power Management (DPM) approaches [3], [10] address the challenge of deciding when to power up or down a wireless network card, with the goals of reducing energy consumption while maintaining connectivity. Reservation-based wireless systems implicitly address this challenge by reducing a device's communication to short, periodic intervals (SPs). Between SPs, no communication to or from this device takes place, allowing the device to power down the network card.

At the CPU-level, Dynamic Voltage Scaling [2], [8], [9], [11] has also received a significant amount of attention. Here, task schedulers not only select the next job to schedule, but also decide on an appropriate speed (and voltage), with the goal of reducing the CPU's energy demands and meeting the jobs' deadlines. Among them, Look-Ahead EDF (laEDF) [8] has been shown to achieve significant energy savings. It defers as much work as possible, setting the operating frequency of the CPU to perform the minimum amount of work that must be done now to ensure all future deadlines are met. This approach is aggressive in that it utilizes as much of the available slack as possible at every scheduling point, i.e., it completes every job as late as possible as long as its deadline will still be met. However, such an approach may result in delayed packet generation, potentially leading to missed packet deadlines. Our work is based on laEDF and overcomes this drawback, ensuring that tasks' deadlines are met and meeting more packet deadlines, while still conserving significant amounts of energy.

Related to our work are also efforts that propose network-aware DVS algorithms [6], [7], [9], e.g., in [9], the authors balance CPU energy with network energy by generating and transmitting packets in bursts. Further, in [7], the authors coordinate DVS with the timeout mechanism frequently used at the network card. However, these efforts do not consider reservation-based approaches and consider only task deadlines, but not packet deadlines.

### 3. Problem Statement

**System Model.** We consider a wireless device executing a set of independent periodic real-time tasks  $\tau = \{\tau_1, \dots, \tau_n\}$ , where the  $j^{th}$  instance (job) of task  $\tau_i$  is denoted as  $J_i^j$ . Note that in the remainder of this paper, we simply use  $J_i$  to indicate any job of  $\tau_i$  when the number of instance is irrelevant to our discussion. Each task  $\tau_i$  is characterized by a *period*  $P_i$ , a *task release time*  $R_i$  (which occurs at the beginning of a period), a *task deadline*  $D_i$  relative to the release time, and the worst-case execution cycles  $WC_i$ . In this work, we assume the task set  $\tau$  is EDF-schedulable in the worst-case scenario. Finally, a DVS-capable processor can operate at a finite set of voltage levels  $V = \{V_1, \dots, V_{max}\}$ , each with an associated frequency  $F = \{F_1, \dots, F_{max}\}$  in terms of cycles per unit time, and power  $P = \{P_1, \dots, P_{max}\}$ . Note that the DVS power model used in this paper can be extended to other generic models which associate different operating efficiencies with different power levels.

A subset of  $\tau$ , denoted by  $\tilde{\tau}$ , represents traffic-generating tasks, i.e., tasks that generate one or more packets in each job. Without loss of generality, we assume that one or more packets may be generated at any possible time, i.e., from the release time to the completion time of a job. Here we logically denote multiple packets generated by the same job as a single packet  $Pkt_i^j$ . Therefore, in addition to a task deadline, each packet-generating task also has a *packet deadline*  $X_i$ , which is relative to the task release time and can differ from the corresponding task deadline (i.e.,  $X_i \geq D_i$ ). Finally, the network interface has a set of transmission rates  $R = \{R_1, \dots, R_{max}\}$ , and the interface dynamically selects a rate based on the current link quality.

**Challenges.** This paper addresses the integrated management of task, packet, and speed scheduling of DVS-capable systems with packet-generating real-time tasks. Conventional task schedulers and DVS mechanisms do not consider packet deadlines; instead they aim to meet task deadlines while reducing the energy consumption of the CPU. As a result, jobs complete as late as system utilization allows and, therefore, packets are generated at the latest possible times. In reservation-based networks, the transmission opportunities for such packets may be limited or may have been missed (e.g., when there is no SP between the release and the deadline of a packet). Figure 2 and Figure 3 illustrate this scenario with two jobs, where only job  $J_2$  generates a packet. Based on utilization, the processor will choose a frequency that prolongs job execution time, while still meeting the job's deadline, however missing the packet's deadline (Figure 2a). A DVS approach that is aware of the limited communication opportunities at the network level will choose a frequency that meets the packet deadline while still preserving energy (Figure 2b).

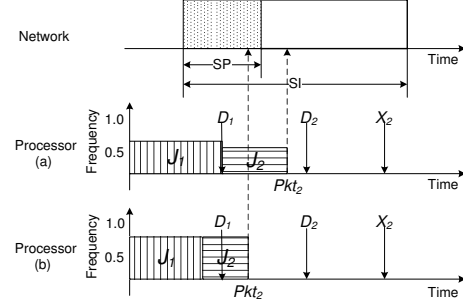


Figure 2. Packet deadline miss caused by DVS

Similarly, Figure 3a shows that, even when DVS selects the highest available execution speed, packets may not be generated in time for transmission before their packet deadlines. The problem is that the task scheduler only considers job deadlines, but not packet deadlines. In Figure 3b, jobs  $J_1$  and  $J_2$  are swapped, ensuring that the packet generated by  $J_2$  will meet its packet deadline.

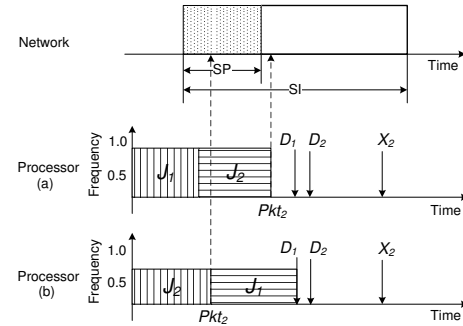


Figure 3. Packet deadline miss caused by scheduler

**Proposed Solution.** As a result, it is not sufficient to modify the packet scheduler to allocate packets to available SPs. Both the task scheduler and DVS mechanism can cause a packet deadline violation if they ignore the transmission constraints of the packet. Therefore, an integrated approach to scheduling is required, i.e., scheduling of tasks, packets, and speeds are driven by packet deadlines and bandwidth constraints. The purpose of this paper is to introduce a co-scheduling approach that ensures that packet deadlines are met while energy consumption of the system remains low.

### 4. An Integrated Scheduling Approach

The observations stated in Section 3 stress the need for an integrated approach to packet scheduling, task scheduling, and DVS management. Figure 4 displays the proposed integration framework. The *Task and Speed Scheduler (TSS)* combines both task scheduling and DVS; and the *Packet Scheduler (PS)* is responsible for queuing and delivering packets to the *Wireless Network Interface Card (WNIC)*. The WNIC transmits packets during a service period in

the order received from the PS. Figure 4 further shows the coordination process and the data shared between the components. Among the shared variables, *StartSlot* denotes the start time of the next available transmission slot at the network side and represents the consumption status of the real-time traffic at the network side. *NextGenTime* denotes the earliest possible time a potential packet will be generated and is used to represent the generation status of the real-time traffic at the processor side. Although a node has exclusive access to the reserved bandwidth, it is still subject to the link state of the channel, and therefore the transmission rate will be time varying and packets may need to be retransmitted. *LinkState* represents the link state of the WNIC and the PS uses it to estimate the current effective transmission rate. The transmission time of a packet is a function of link state, packet size, and protocol overheads. Finally, we use *NextPkt* to denote the next packet provided by the PS to the WNIC.

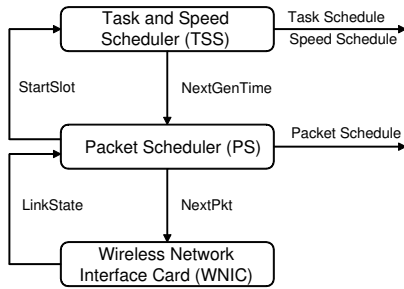


Figure 4. Integrated resource management

Based on *StartSlot*, TSS attempts to generate a series of packets through dynamic job deadline modifications such that the generated packets will be transmitted as a burst beginning at *StartSlot*. As jobs complete, and thus packets are generated, the PS moves *StartSlot* forward considering the channel reservation and the effective packet transmission time. If *NextGenTime* is beyond the current time (expressed as *curTime*) by a threshold  $\Delta$  and the packet queue is empty, the network will enter the sleep state. It will re-awake at *NextGenTime* or the beginning of the next SP if *NextGenTime* falls between two SPs. The adjustment of job execution order, computation of the operating frequency, and estimation of *NextGenTime* only occurs at job scheduling points (i.e., release and completion times). The adjustment of *StartSlot* only occurs at packet scheduling points (i.e., when packets are enqueued or dequeued) and when the link state changes.

#### 4.1. Task and Speed Scheduling

The TSS uses a look-ahead technique to dynamically adjust the task scheduling order and CPU operating frequency to meet as many packet deadlines as possible. It differs

thereby from other DVS approaches in that the focus is on meeting packet deadlines, as opposed to meeting task deadlines only. More specifically, the TSS looks ahead one packet-generating job at a time, utilizing two guidelines to ensure that a job's packet will meet its deadline: (i) steering DVS based on packet deadlines instead of job deadlines and (ii) scheduling packet-generating jobs before other real-time jobs, if needed. These guidelines are applied such that no schedulability guarantees provided by EDF are violated and the energy consumption of the CPU is still kept low. Toward this end, this paper proposes a *network-aware EDF*

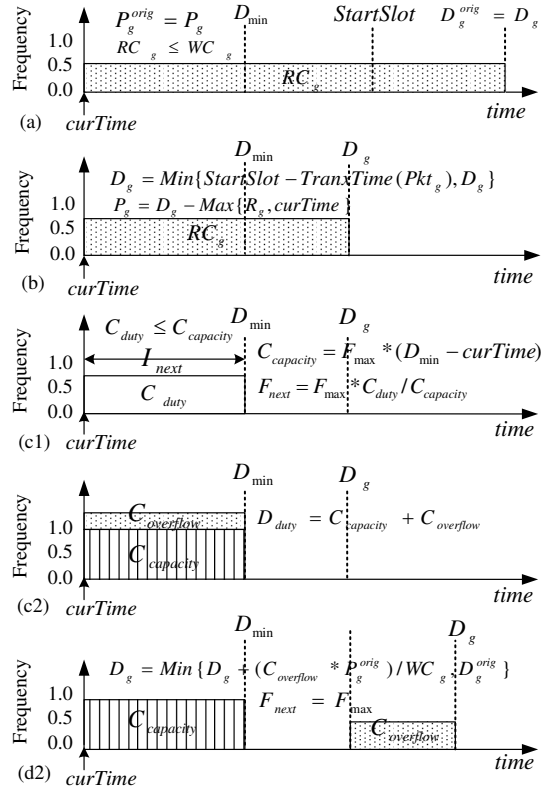


Figure 5. An example illustrating the operations of network-aware EDF

(or *naEDF*) task and speed scheduler, which is based on laEDF [8]. It is invoked at every scheduling point (i.e., job release, job deadline, and job completion) to determine the processor frequency  $F_{next}$  and the next job  $J_{next}$  for the next scheduling interval  $I_{next}$  (i.e., the interval from the current time *curTime* to the next scheduling point  $D_{min}$ ).  $F_{next}$  must be chosen to minimize the processor frequency while ensuring that the packet deadline of the next packet-generating job (which can differ from  $J_{next}$ ) will be met. The basic idea to achieve this is to take the packet generating job  $J_g$  with the earliest deadline and adjust its task parameters. More specifically, naEDF modifies the job's deadline  $D_g$  and period  $P_g$  to ensure that the next real-time packet will be available to the packet scheduler before

---

**Algorithm 1** naEDF

---

```
1: function adjust-frequency-voltage():
2: let  $J_g$  be the next packet-generating job with the earliest packet
   deadline
3: if  $R_g > curTime$  then
4:    $NextGenTime = R_g$ 
5: else
6:    $NextGenTime = curTime$ 
7: end if
8:  $D_g^{orig} = D_g; P_g^{orig} = P_g$ 
9:  $D_g = \min\{StartSlot - TranxTime(Pkt_g), D_g\}$ 
10:  $P_g = D_g - \max\{R_g, curTime\}$ 
11: Let  $J$  be the job set  $\{J_1, \dots, J_n\}$ 
12: Let  $D_{min}$  be the minimum job deadline among all uncompleted jobs
   in  $J$ 
13: Let  $P'_i = \min\{D_i - R_i, P_i\}$  for every job
14:  $U = \sum_{J_i \in J} \frac{WC_i}{P'_i * F_{max}}$ 
15:  $C_{duty} = 0$ 
16: for each  $J_i \in J$  with the decreasing order of job deadline do
17:    $U = U - \frac{WC_i}{P'_i * F_{max}}$ 
18:   if  $U \leq 1$  then
19:      $x = \max\{0, RC_i - (1 - U) * (D_i - D_{min}) * F_{max}\}$ 
20:      $U = U + \frac{RC_i - x}{(D_i - D_{min}) * F_{max}}$ 
21:   else
22:      $x = RC_i$ 
23:      $U = U + \frac{WC_i}{P'_i * F_{max}}$ 
24:   end if
25:    $C_{duty} = C_{duty} + x$ 
26: end for
27:  $I_{next} = D_{min} - curTime; C_{capacity} = F_{max} * I_{next}; C_{overflow} =$ 
    $C_{duty} - C_{capacity}$ 
28: if  $C_{overflow} > 0$  then
29:    $D_g = \min\{D_g + \frac{C_{overflow} * P_g^{orig}}{WC_g}, D_g^{orig}\}$ 
30: end if
31: select  $J_{next} \in J$  with the minimum deadline as next running job
32: select frequency as  $\min\{F_i \geq \min\{F_{max}, \frac{C_{duty}}{D_{min} - curTime}\}\}$ 
33:  $D_g = D_g^{orig}; P_g = P_g^{orig}$ 
34: upon job-released( $J_i^k$ ):
35:  $RC_i^k = WC_i / \#remaining\ cycles^*$ 
36: adjust-frequency-voltage()
37: upon job-execution( $J_i^k$ ):
38: decrement  $RC_i^k$ 
39: upon job-completion( $J_i^k$ ):
40:  $RC_i^k = 0$ 
41: adjust-frequency-voltage()
```

---

*StartSlot*. While naEDF could be based on a variety of DVS algorithms, we use laEDF as the foundation since it aggressively delays job execution as late as possible at every scheduling point to maximize energy savings.

Figure 5 illustrates the operation of naEDF, which is presented in detail in Algorithm 1. In Figure 5(a),  $J_g$  is the currently or next executing job (at *curTime*) with remaining cycles  $RC_g \leq WC_g$ . *StartSlot* is less than  $D_g$  and therefore  $D_g$  and  $P_g$  need to be adjusted to ensure that the job's packet will be generated in time if  $J_g$  is a packet-generating job. In this case, naEDF first saves the original parameters  $P_g$  and  $D_g$  (line 8 in Algorithm 1). In Figure 5(b),  $D_g$  is set to be the minimum of  $StartSlot - TranxTime(Pkt_g)$  and

the original  $D_g$ , where  $TranxTime(Pkt_g)$  is the estimated packet transmission time for the current link state (line 9). Further,  $P_g$  is shortened to  $D_g - curTime$  if  $J_g$  has already been released (i.e.,  $R_g \leq curTime$ ) or otherwise to  $D_g - R_g$  (line 10).

The remainder of Figure 5 compares two different cases. In Figure 5(c1), naEDF invokes the laEDF algorithm to compute the duty cycles  $C_{duty}$  for  $I_{next}$ .  $C_{duty}$  is the minimum number of cycles that must be executed in  $I_{next}$  to ensure that all current and future jobs meet their job deadlines. In this example,  $C_{duty}$  is within the processor's capacity  $C_{capacity}$  (i.e.,  $C_{duty} \leq C_{capacity}$ ), where  $C_{capacity}$  is the maximum cycles that the processor can execute in  $I_{next}$  if running at the highest frequency  $F_{max}$  (line 27). Note that  $J_{next}$  and  $F_{next}$  are identical to the outputs of laEDF (lines 11- 26). However, the resulting scheduling order may differ from EDF. In the second case, shown in Figure 5(c2),  $C_{duty}$  is beyond the processor's capacity, which forces  $F_{next}$  to be  $F_{max}$  and leading to a 'cycle overflow'. This overflow is expressed as  $C_{overflow} = C_{duty} - C_{capacity}$  (line 27). Figure 5(d2) (and lines 28 - 30) show how  $D_g$  must be postponed to handle this situation.  $D_g$  is pushed back by  $\frac{WC_g}{P_g^{orig}}$  cycles per unit time. Since any EDF-based algorithm implicitly reserves  $\frac{WC_i}{P_i^{orig}}$  cycles per unit time for every released job  $J_i$ , the postponed cycles at any time are less than the implicitly reserved cycles and so postponing is safe (i.e., it will not cause any other jobs to miss their deadlines). TSS sets  $NextGenTime$  to be either *StartSlot* or the release time of the next packet-generating job with the earliest deadline (lines 3- 7).

**Theorem 4.1.** *If a set of periodic tasks is EDF schedulable at the highest frequency, then the task set is also naEDF schedulable.*

**Proof.** We consider two groups of jobs: (1) the group  $G_c$  of currently released jobs  $J_1, \dots, J_n$  such that  $R_i \leq curTime < D_i$  for each  $J_i$  and (2) the group  $G_f$  of all future jobs which have not been released at *curTime*. We prove that naEDF will preserve schedulability at every scheduling point (i.e., naEDF reserves sufficient cycles for every job in  $G_f$  and runs as fast as necessary to meet every job's deadline in  $G_c$ ).

For all jobs in  $G_f$ , like laEDF, naEDF reserves a sufficient number of cycles. The number of reserved cycles between  $D_{max} = \max_{1 \leq i \leq n} D_i$  and any time  $t$  are  $U * F_{max} * (t - D_{min})$ , where  $U \leq 1$  is the system utilization. The number of reserved cycles from  $D_{min}$  to  $D_{max}$  are at least  $\sum_{1 \leq i \leq n} \frac{WC_i * (D_{max} - D_i)}{P_i^{orig}}$ , which guarantees that all future jobs will meet their deadlines using an EDF scheduler running at the highest frequency. If the reserved cycles from  $D_{min}$  to  $D_{max}$  are beyond the processor's capacity, the over-reserved cycles overflow into  $I_{next}$ , which does not affect the schedulability of the jobs in  $G_f$ , but forces jobs in  $G_c$

to run faster to prepare for the next packet-generating job (which is not yet released).

For all jobs in  $G_c$ , as in laEDF, a number of cycles from each job in  $G_c$  are allocated from  $D_{min}$  to the respective job deadlines. The cycles which cannot be allocated overflow into  $I_{next}$  and are captured with  $C_{duty}$ . If  $C_{duty}$  is less than the processor's capacity  $C_{capacity}$  within  $I_{next}$ , naEDF is able to meet all current jobs' deadlines, since their respective reserved cycles after  $D_{min}$  plus their overflow cycles within  $I_{next}$  are less than or equal to their respective remaining cycles. Otherwise, the duty cycles  $C_{duty}$  overwhelm the processor's capacity  $C_{capacity}$ . The difference  $C_{overflow}$  between  $C_{duty}$  and  $C_{capacity}$  is at most the over-claimed cycles of the next packet-generating job  $J_g$ . Retrieving all overflow cycles pushes the job deadline  $D_g$  later to at most the original job deadline.  $D_g$  is pushed back by the implicitly reserved cycles per unit time (i.e.,  $\frac{WC_g}{P_g^{orig}}$ ) for any EDF-based algorithm. Therefore, a postponed deadline will not affect any other jobs.  $\square$

## 4.2. Packet Scheduling

Packet scheduling is based on work-conserving EDF, i.e., packets are ordered and scheduled in increasing order of packet deadlines, with modifications to support the discrete channel accesses and the cooperation of the packet scheduler with the TSS and the WNIC. To this end, a reservation-aware packet scheduler is proposed, called *pktEDF*, shown in Algorithm 2. It reacts to various network events, such as transmission failure/success (lines 27, 30), reservation replenishment and consumption (line 28), link state change (line 37), and packet enqueueing (line 34). Accordingly, several timers and state variables are used to help manage these events. State variable  $Time_{complete}$  is used to record the expected transmission completion time of the current packet, based on the current link state. State variable  $Linkstate_{cur}$  is used to store the current link state. Timer  $Timer_{pkt}$  is used to monitor the transmission completion of every packet or to record the earliest expected generation time of the next packet when the packet queue is empty. The timeout of  $Timer_{pkt}$  indicates that a retransmission is required or a potential packet will be enqueued at its expiration time. Timer  $Timer_{sp}$  keeps track of bandwidth reservations. It is replenished at the beginning of SPs and expires at the end of SPs.

The scheduler transmits queued packets during SPs according to the EDF policy. Once the queue is empty (lines 9-17), the scheduler will plan a state transition of the network. If  $NextGenTime$  is at least  $\Delta$  (break-even time) away from the current time  $curTime$ , the network card will switch into the inactive state until the next available access time after  $NextGenTime$ . Otherwise, the network card will stay idle and wait for the next packet. If packets are backlogged (lines 19-25), the packet with the earliest deadline will be

---

### Algorithm 2 Enhanced EDF packet scheduler (pktEDF)

---

```

1: Let  $Timer_{sp}$  be the timer for the next SP
2: Let  $Timer_{pkt}$  be the timer for current packet in transmission
3: Let  $Linkstate_{cur}$  be the current link state
4: packetTransmission():
5: update  $StartSlot$  according to  $Linkstate_{cur}$  and the packet queue
6:  $NextPkt =$  next packet with the earliest packet deadline
7: /*if the packet queue is empty, check if it is desirable to sleep*/
8: if  $NextPkt =$  null then
9:   if  $NextGenTime$  in  $[k * SI + SP, (k + 1) * SI]$  then
10:      $StartSlot = (k + 1) * SI$  /*the beginning of next SP*/
11:     setTimerAndSleep( $Timer_{sp}$ ,  $StartSlot$ )
12:   else if  $NextGenTime - curTime > \Delta$  then
13:     setTimerAndSleep( $Timer_{pkt}$ ,  $NextGenTime$ )
14:   else
15:     Idle()
16:   end if
17:    $StartSlot = curTime$ 
18: else
19:    $Time_{complete} =$  TranxTime( $NextPkt$ ) +  $curTime$ 
20:   if  $Time_{complete}$  in  $[k * SI + SP, (k + 1) * SI]$  then
21:     setTimerAndSleep( $Timer_{sp}$ ,  $(k + 1) * SI$ )
22:   else
23:     tranxPacket( $NextPkt$ ) /* issue packet to WNIC*/
24:     setTimerAndSleep( $Timer_{pkt}$ ,  $Time_{complete}$ )
25:   end if
26: end if

27: upon packet-transmission-timeout():
28: upon sp-timer-expire():
29: packetTransmission()

30: upon packet-transmission-complete():
31: cancelTimer( $Timer_{pkt}$ )
32: removePacket( $NextPkt$ )
33: packetTransmission()

34: upon packet-generated(pkt):
35: insert  $pkt$  into the packet queue according to EDF policy
36: update  $StartSlot$  according to  $Linkstate_{cur}$  and the packet queue

37: upon linkstate-changed():
38: update  $Linkstate_{cur}$ 
39: update  $StartSlot$  according to  $Linkstate_{cur}$  and the packet queue

```

---

the next packet for transmission. If the remaining time in the currently allocated SP is sufficient (i.e., greater than the break-even time  $\Delta$ ) for the transmission of the next packet, the scheduler transmits this packet and monitors the transmission status. Otherwise, the scheduler will switch the network into the inactive state and transmit the backlogged packet in the next SP. A failed transmission will be retried until the transmission succeeds or a retry limit is reached. After a transmission attempt, the scheduler will continue to check the queue and the aforementioned procedure (line 4) is repeated. Note that the scheduler will dynamically update  $StartSlot$  to indicate the next available transmission time. If the queue is empty and the network is ready for transmission,  $StartSlot$  is set to the current time (line 17). After a transmission attempt or when a packet is enqueued, the scheduler updates  $StartSlot$  according to  $Linkstate_{cur}$  and the packet queue. Also,  $StartSlot$  is recomputed whenever the transmission rate changes (lines 37-39).

## 5. Evaluations

### 5.1. Simulation Setup

The simulator used for our evaluations is an extension of the simulator used in [7]. It is a Java-based program to simulate the operation of DVS-enabled CPUs and real-time packet schedulers. The simulator output is the processor’s energy consumption and the number of generated packets that meet their transmission deadlines. The processor model is based on the Centrino Mobile processor, operating at frequencies 1.0, 1.33, and 1.83 GHz (the ratio of energy consumptions between these speeds is approximately 1:1.32:2.1). The network model in the simulations strictly follows the HCCA model [5], which we used as our target network model in previous sections. The failure probability of a delivery (a single transmission attempt) in the simulations is 0.06 (this value is chosen to match the statistics we obtained from our experiments). Note that our results focus on CPU-level energy savings, because network-level energy savings are due to the reservation-based mechanism and not our algorithms. The performance metrics used include:

- Worst-case system utilization  $U = \sum \frac{WC_i}{\min\{D_i, P_i\} * F_{max}}$ . Whenever the utilization is fixed, the worst-case execution cycles of the tasks were scaled uniformly to achieve the desired utilization.
- Relative utilization of packet-generating tasks  $\alpha = \sum \frac{WC_i}{P_i * F_{max}}$  for all packet-generating tasks in  $\tilde{\tau}$ . This indicates how much of the system’s utilization is due to packet-generating tasks.
- Ratio  $\beta$  of reserved bandwidth (i.e.,  $\frac{SP}{ST}$  in terms of time) to average bandwidth requirement (i.e.,  $\frac{\sum TransTime(Pkt_i^j)}{SimulationTime}$ ). This represents the network utilization.
- Ratio  $\gamma$  of packet deadline to the corresponding original job deadline.

For each simulation, unless specified otherwise, we use the following default settings: the job deadline is equal to the job period, the system’s worst-case task utilization  $U$  is 0.8,  $\alpha$  is 0.5,  $\beta$  is 1.6, and  $\gamma$  is 1.2. The actual fraction of the worst-case execution cycles of a job, and the actual fraction of the worst-case packet size generated by a task are randomly varied in the ranges [0.6, 1.0] and [0.2, 1.0], respectively. Each of the following results is an average over at least 200 randomly generated task sets  $\tau$ .

### 5.2. Simulation Results

**Varying worst-case task utilizations.** The following results compare naEDF to laEDF and hiEDF, where the latter is a scheduler with DVS disabled (i.e., the jobs always execute at maximum speed). We first investigate the effects of task utilization on real-time and energy performance.

Figures 6(a) and 6(b) show the energy (normalized to hiEDF) and real-time performance with varying values of worst-case utilization  $U$ . Since hiEDF does not use DVS, it always consumes the largest amount of energy, while laEDF consumes the lowest energy in complying with its primary design objective. Only while the system utilization is slight, naEDF meets slightly less deadlines than hiEDF with much less energy consumption, since naEDF aggressively delays job executions and looks ahead only one packet-generation job at a time. In contrast, naEDF evenly spreads the required slack cycles over the whole experiment time, and on average runs at a slightly higher frequency than laEDF, leading to a larger energy consumption than laEDF. However, as shown in Figure 6(b), naEDF is always able to meet more packet deadlines than laEDF.

**Varying values of  $\alpha$ .** Next, we investigate the impact of the ratio  $\alpha$  (utilization of packet-generating tasks over total system utilization). Since both hiEDF and laEDF are unaware of packets generated by the tasks, we expect them to show stable energy consumptions and packet deadline violations. This is verified by Figures 6(c) and 6(d). The results also show that the performance of naEDF is significantly affected by  $\alpha$ . When  $\alpha$  increases, the opportunities to delay the execution of non-packet-generating jobs decrease and the replenishment of requested slack cycles must rely increasingly on speeding up the processor. Since the processor can only run at the highest speed  $F_{max}$  when the task utilization ( $U$ ) is high (default value is 0.8), the potential speedup is limited. As a result, the energy consumption of naEDF increases in proportion to the value of  $\alpha$ , but the percentage of packet deadline violations is still much lower compared to hiEDF. The only exception occurs when both the value of  $\alpha$  and the actual task utilization are close to 1, i.e., the system consists of only packet-generating tasks and the system is highly loaded. Then, naEDF misses slightly more packets than hiEDF. The reason is that naEDF only looks ahead one packet-generating job at a time, which may cause subsequent packet-generating jobs to miss their deadlines, even when the processor runs at the highest speed. In this situation, naEDF still outperforms laEDF in terms of meeting packet deadlines and hiEDF in terms of energy consumption.

**Varying values of  $\beta$ .** Next, we vary the amount of bandwidth reservations (i.e.,  $\beta$ ) to compare the three algorithms’ performance for different network load scenarios (Figures 6(e) and 6(f)). Both hiEDF and laEDF have stable energy consumptions, but their real-time performances improve with increasing  $\beta$  (simply because more bandwidth is available). When the bandwidth is highly over-provisioned, the transmission opportunities increase and all algorithms perform well. When the reserved

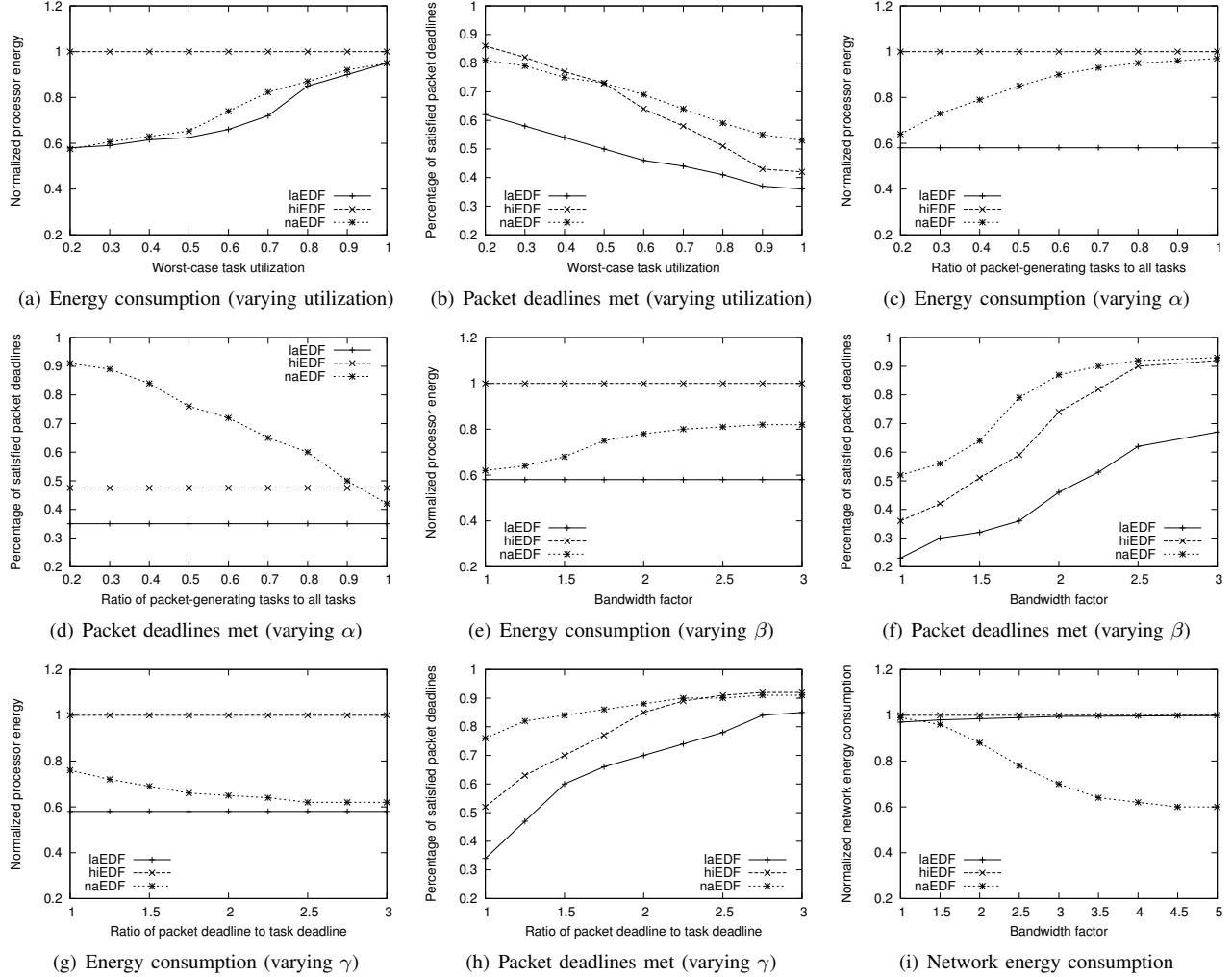


Figure 6. Real-time and energy performance

bandwidth is less than two times of the average required bandwidth (i.e.,  $\beta < 2$ ), naEDF performs significantly better than both hiEDF and laEDF. This is because the slack cycles preservation for upcoming packet-generating jobs takes effect; the cost of this is a slightly increased energy consumption. Interestingly, when  $\beta > 2$ , the energy consumption increases very slowly, because most packets can meet their deadlines, and therefore, only a few jobs need to be re-ordered or executed at a higher frequency.

**Varying values of  $\gamma$ .** Figures 6(g) and 6(h) show the effects of varying packet deadlines on real-time performance and energy. Both hiEDF and laEDF are unaware of packet deadlines and have constant energy performance. The transmission opportunities of packets depend strongly on the value of  $\gamma$  (i.e., the larger  $\gamma$ , the more SPs available), especially when  $\beta > 1$  (note that the default value of  $\beta$  in this experiment is 1.6). The number of packets that

meet their deadlines increases with increasing values of  $\gamma$  (naEDF performs particularly well for values between 1 and 2). The energy consumption of naEDF does not decrease significantly when  $\gamma$  increases, since naEDF always aggressively attempts to fill each available transmission slot (through *StartSlot*) with packets. In conclusion, naEDF performs particularly well in highly loaded systems where job deadlines and packet deadlines are close to each other.

**Energy consumption of the network.** All previous experiments measure the energy consumption only at the processor side since the default reserved bandwidth is less than two times of the average required bandwidth. In that case, the network device will stay active for most of the time, and thus the differences of energy consumptions of hiEDF, laEDF, and naEDF at the network side are very small. When the bandwidth is over-provisioned, however, the network is able to enter the sleep state whenever the

next packet-generating job cannot generate packets before the break-even time. Figure 6(i) shows that naEDF can benefit from being aware of the processor’s activity (using *NextGenTime*), i.e., it can put the network card into the low-power sleep state when no packets are expected. On the other hand, with hiEDF and laEDF this awareness is not given and for both algorithms, the network card has to stay active for the entire duration of SP.

## 6. Experimentation

Both naEDF and pktEDF have been implemented as kernel-loadable modules in Linux 2.6 (Figure 7). The naEDF scheduler utilizes the SpeedStep frequency scaling driver provided by Linux and pktEDF utilizes the transmission status notification callback functions (such as transmission timeout, failure, success, and transmission rate) provided by the wireless network card (Orinoco) to monitor actual link state. Further, naEDF relies on the kernel’s high resolution timer loadable module (hrTimer) to precisely control the accuracy of scheduling points and all other timing actions (with an accuracy of up to  $0.1\mu s$ ). The scheduling timer periodically invokes the naEDF algorithm to set the next job runnable, to put all other jobs into a waiting state, to switch between the previous and the next job (this is performed by the original Linux scheduler), to set a frequency (using SpeedStep), and to set the scheduling timer to the next scheduling point. Packets generated by real-time jobs are collected in a packet queue and the packet scheduler delivers packets from this queue individually to the wireless driver during the SPs. Further, the packet scheduler also receives feedback from the driver about events such as transmission timeouts, failures, and current transmission rates, allowing it (and the task scheduler) to adapt to changes in link state. The task and packet schedulers communicate using shared memory regions.

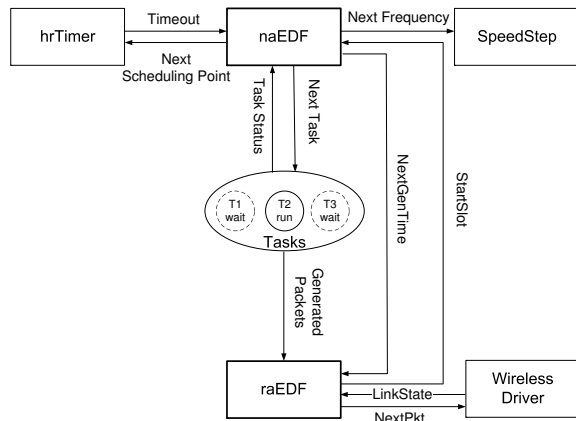


Figure 7. Linux kernel implementation

## 6.1. Experimental Setup

Experiments were performed on two Linux-based Lenovo T61 laptops, each supporting 1.0, 1.33, and 1.83GHz. Both laptops communicate via an Orinoco Gold Wireless 802.11b card with a bit rate of 2MBit/s and 95mW, 925mW, and 1425mW power consumption in the doze, receive, and transmit mode, respectively. The first laptop acts as a wireless client sending real-time packets generated by its tasks and the second laptop acts as wireless access point or gateway. We generated a set of threads to represent common periodic real-time tasks (such as video surveillance or periodic sensing of physical parameters). The channel access parameters, SP and SI, are chosen as 20ms and 100ms, respectively and the number of tasks on the client is 10. Each task has a period randomly distributed in the range [100, 500]ms and the task deadline is set to its period. The laptops were positioned at a distance of 30m, providing a typical transmission probability of 0.05-0.08.

Current wireless cards do not support the HCCA mode, therefore, we emulated a reservation-based channel by placing the laptops in a controlled environment (e.g, no other active 802.11 transmitters were in the vicinity) and by enforcing that packet transmissions only occur during SP. Further, in HCCA, retransmissions are separated by a fixed small interval (e.g.,  $30\mu s$  in 802.11b), whereas the wireless card in our experiments uses exponential backoffs. The back-off procedure in 802.11 is dynamic and unpredictable, potentially with a duration of up to 1ms between retransmissions. As a consequence, the network model of our experimental setup will lead to slightly larger (and less predictable) worst-case transmission times than in the simulation model.

## 6.2. Experimental Results

Figures 8 and 9 show the energy (normalized to hiEDF) and real-time performance with varying values of worst-case utilization  $U$ . Compared to the simulation results (Figures 6(a) and 6(b)), the experimental results deviate only slightly.

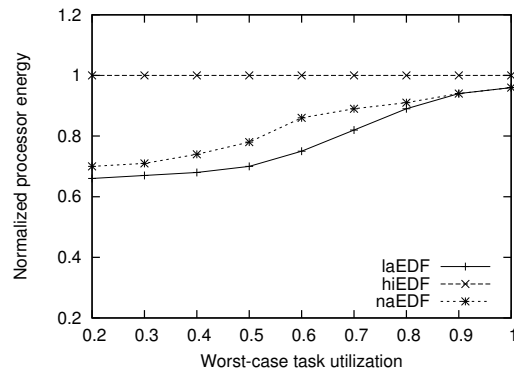


Figure 8. Energy consumption for various values of  $U$

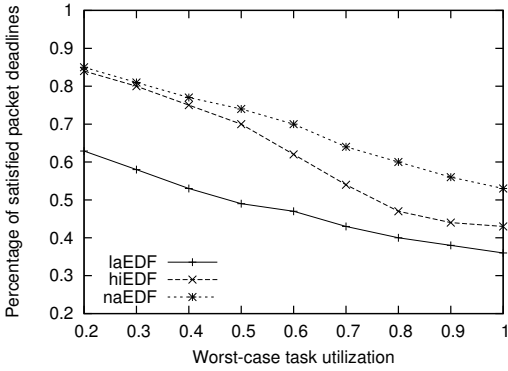


Figure 9. Packet deadlines met for various values of  $U$

Both simulation and experimentation ran for very long times, thereby averaging random perturbations (such as transmission failures and retransmissions). The small differences in the results are caused mainly by implementation overheads that were ignored by the simulation, including processor scheduling overheads and wireless driver interrupts. In addition, the exponential backoffs of our wireless card (as described above) affect the results slightly. These overheads may cause some jobs to run past their deadline and whenever this happens, the remainder of a job is executed at the highest frequency in the system. Overall, the experimental results validate our simulations and show that naEDF performs well in practical environments.

## 7. Conclusions and Future Work

In this paper, a novel approach to co-scheduling packets, tasks, and CPU speeds in reservation-based environments is proposed, with the goal to meet as many packet deadlines as possible. Bandwidth reservations are beneficial for wireless real-time systems, both in terms of reduced contention and increased network-level energy savings. However, a co-scheduling approach is required to ensure that CPU scheduling, DVS, and packet scheduling decisions take the limited transmission opportunities into account. Our future work will extend the proposed schedulers to also consider sporadic real-time tasks, task dependencies, incoming packets, and mechanisms to handle overload scenarios (e.g., using weights).

## Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0834180.

## References

[1] L. Benini, A. Bogliolo, and G. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 2000.

[2] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proc. of RTAS*, 2006.

[3] V. Devadas and H. Aydin. Real-time dynamic power management through device forbidden regions. In *Proc. of RTAS*, 2008.

[4] Y. P. Fallah and H. Alnuweiri. Hybrid polling and contention access scheduling in IEEE 802.11e w lans. *Parallel Distrib. Comput.*, 67(2), 2007.

[5] IEEE 802.11 WG. Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Medium access control (MAC) enhancements for quality of service (QoS). *IEEE 802.11e Standard*, Nov 2005.

[6] G. Sudha Anil Kumar and G. Manimaran. Energy-aware scheduling of real-time tasks in wireless networked embedded systems. In *Proc. of RTSS*, 2007.

[7] B. Mochocki, D. Rajan, X. S. Hu, C. Poellabauer, K. Otten, and T. Chantem. Network-aware dynamic voltage and frequency scaling. In *Proc. of RTAS*, April 2007.

[8] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of SOSP*, 2001.

[9] C. Poellabauer and K. Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proc. of RTAS*, 2004.

[10] 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 Mobicom*, 2003.

[11] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. of DAC*, 2001.

[12] D. Rajan, C. Poellabauer, X. S. Hu, L. Zhang, and K. Otten. Wireless channel access reservation for embedded real-time systems. In *Proc. of EMSOFT*, October 2008.

[13] M. Saksena and S. Hong. An engineering approach to decomposing end-to-end delays on a distributed real-time system. In *Proc. of WPDRTS*, Washington, DC, 1996.

[14] O. S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. In *Proc. of the IEEE*, July 2003.

[15] L. Wang and Y. Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *Mobile Networks and Applications*, 11(5):723–740, 2006.

[16] Y. Zhang and R. West. End-to-end window-constrained scheduling for real-time communication. In *Proc. of RTCSA*, 2004.