DEOS: Dynamic Energy-Oriented Scheduling for Sustainable Wireless Sensor Networks

Ting Zhu* Abedelaziz Mohaisen† Yi Ping‡ Don Towsley§
*Binghamton University †University of Minnesota ‡Shanghai Jiao Tong University §University of Massachusetts

***Abstract*—Energy is the most precious resource in wireless sensor networks. To ensure sustainable operations, wireless sensor systems need to harvest energy from environments. The time-varying environmental energy results in the dynamic change of the system's available energy. Therefore, how to dynamically schedule tasks to match the time-varying energy is a challenging problem. In contrast to traditional computing-oriented scheduling methods that focus on reducing computational energy consumption and meeting the tasks' deadlines, we present DEOS, a dynamic energy-oriented scheduling method, which treats energy as a first-class schedulable resource and dynamically schedules tasks based on the tasks' energy consumption and the system's real-time available energy. We extensively evaluate our system in indoor and outdoor settings. Results indicate that DEOS is extremely lightweight (e.g., energy consumption overhead in the worst case is only 0.039%) and effectively schedules tasks to utilize the dynamically available energy.**

I. Introduction

To ensure sustainable operations of wireless sensor networks, environmental energy harvesting has been regarded as the right solution for long-term applications. However, the harvested energy is usually not sufficient to allow the sensor nodes to keep active all the time. Therefore, the bottleneck in sustainable wireless sensor networks is energy instead of computing and communication capabilities. What is worse is that the availability of energy in a sustainable sensor network is intermittent and varies over time. Thus it becomes important to shift the focus from the constraint of computation and communication to the energy constraint and explore a method, which dynamically schedules tasks based on energy.

In this paper, we propose *dynamic energy-oriented scheduling* - a method that allocates tasks to resources based on the system's instantaneous available energy and the tasks' energy consumption. The novel features are (i) the ability to decompose and recombine tasks to eliminate redundant operations, and (ii) the concurrent execution to simultaneously utilize multiple resources (e.g., sensing and communication). In addition, admission control is applied to ensure the execution of high priority tasks when there is insufficient energy. It is a challenging task to build an energy efficient and light weight scheduling method because the scheduling operation also consumes energy. This calls for a design that reduces scheduler overhead. It should be noted that dynamic energy-oriented scheduling achieves a balance between the system's available energy and the energy consumption of tasks in real time. Such a realtime balance makes our work on energy-oriented scheduling unique and novel. More specifically, our major contributions are as follows:

- To the best of our knowledge, this is the first in-depth work to investigate optimal dynamic energy-oriented multiple tasks allocations with a time-varying and limited energy constraint.
- We have designed and implemented the first dynamic energy-oriented scheduler for efficiently allocating tasks in realtime based on the available energy and the tasks' (i) priorities, (ii) values, and (iii) energy consumption.
- We evaluate our design extensively in multiple real-world testbeds and simulations. The results indicate that our system effectively allocates tasks to efficiently utilize the realtime available energy.

The rest of the paper is organized as follows: Section II describes the motivation behind an energy-oriented design for sustainable sensor networks. Section III introduces our basic design, followed by an advanced design in Section IV. The system's performance is evaluated by conducting real-world experiments and simulation (Section V). Related work is discussed in Section VI. Section VII concludes the paper.

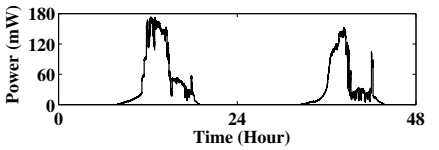
II. Motivation

The motivation of this work comes from our experience of deploying energy-harvesting sustainable sensor networks. In such networks, the harvested environmental energy is limited and cannot afford a sensor node to stay active all the time. Thus energy is more important than other resources, such as computing capacity. This section identifies the need for a dynamic energy-oriented scheduler (DEOS) design and describes the unique features of DEOS design.

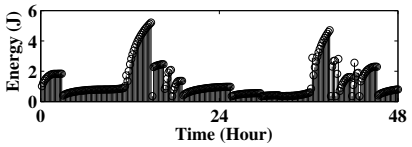
A. The Need for a DEOS Design

In traditional computing-oriented scheduling design that focuses on reducing the computational energy consumption and meeting the tasks' deadlines (the time before which the tasks have to be executed), the computing capability is usually the limiting resource. The main design goal is to maximize the utilization of computing power. However, in sustainable sensor networks, ambient energy (e.g., solar or wind) is normally not sufficient to sustain continuous active operation of sensor nodes in the long run [1], [2]. From our empirical measurement results [3], even on a sunny day, the total energy harvested at an energy-harvesting sensor node (e.g., Twin-Star node [4]) can only allow the node to work at 100% duty cycle for 6.37 hours. The limited available energy is the main factor that prohibits a node from fully utilizing its hardware resources (such as computing and sensing). Energy-oriented design aims

978-1-4673-0775-8/12/$31.00 ©2012 IEEE



(a) Harvested Power Over Time



(b) Actual Available Energy Over Time

Fig. 1. Harvested Power and Actual Available Energy Over 48 Hours

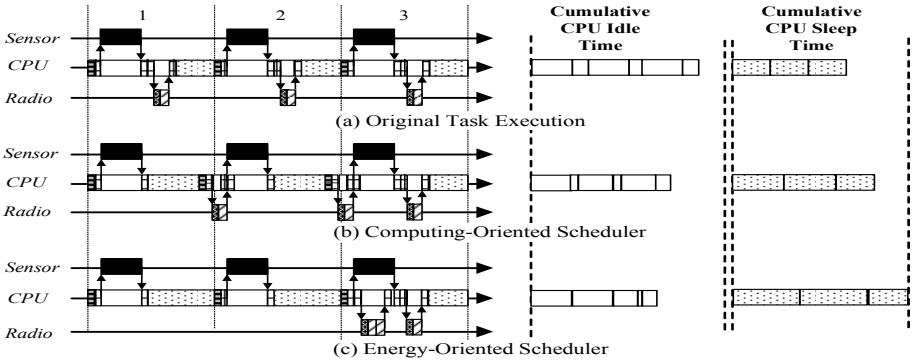


Fig. 2. Comparison among Three Different Scheduling Methods

to optimize energy utilization, which results in system-wide optimal performance under a limited energy constraint.

Moreover, due to the dynamic change of environmental energy in sustainable sensor networks, the actual available energy is intermittent and varies over time. Figure 1 shows the harvested environmental energy and the available energy suggested by our hardware platform over a period of 2 days in an outdoor environment. If we use a computing-oriented design which schedules tasks based on available CPU capacity, the system runs out of power when little energy is available.

Due to these unique features of sustainable sensor networks, it is necessary to design a resource scheduler that treats energy as a *first-class schedulable resource* and dynamically adjust the execution of tasks according to the intermittent and time-varying available energy.

B. Unique Features of the DEOS Design

Besides dynamically adjusting the execution of tasks based on the available energy, the DEOS design exhibits two unique features. *The first feature is to consider the sensor node as a single system and to aggregate the tasks on other components (e.g., communication) besides CPU*. By doing this, a system running the energy-oriented scheduler can further reduce redundant activities on other components and consume less energy than one running the computing-oriented schedulers. As an example, Figure 2 shows a typical sensing application in which the CPU triggers the sensor to sense the data, and then triggers the radio to send the data back to the base station. By separating CPU sensing activity from transmission activity and combining the transmission activity with its successor's sensing activity, a traditional computing-oriented scheduler (e.g., [5] and [6]) minimizes cumulative CPU idle time (shown in Figure 2(b)). However, the cumulative CPU sleep time of the computing-oriented scheduler is still smaller than that of an energy-oriented scheduler (shown in Figure 2(c)). This is because the energy-oriented scheduler not only aggregates CPU activities but also combines communication tasks to minimize redundant interactions between CPU and communication component.

DEOS's second unique feature is *concurrent execution*. Most hardware platforms allow concurrent activities of different components. As shown in Figure 2(c), DEOS concurrently

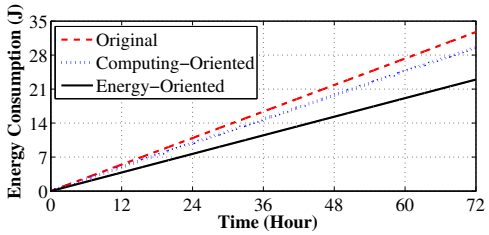


Fig. 3. Empirical Comparison

allocates sensing and communication activities to further reduce CPU idle time and increase sleep time.

To confirm this empirically, we conducted a series of experiments by running the above sensing application with three different scheduling methods on a MICAz mote every 20 seconds for 72 hours. Figure 3 shows the cumulative energy consumption of these methods. Since the sensor node sleeps most of the time, the energy consumption of the CPU is low. Therefore, compared with the original task execution, the computing-oriented scheduler only reduces total energy consumption by 9.61%. In contrast, the energy-oriented scheduler reduces total energy consumption by 29.87%.

III. Basic Design

This section introduces the basic design of DEOS. The objective of DEOS is to dynamically optimize system performance based on the intermittent and time-varying available energy. Since the execution of a scheduler also consumes energy, the scheduler should be simple and lightweight.

A. Design Overview

As illustrated in Figure 4, DEOS consists of four phases:

1) Decomposition: The energy-oriented scheduler takes all tasks as inputs and decomposes them into subtasks only if some of the subtasks can be combined together to save energy. For example, in Figure 2(a), since the transmission activities can be combined with each other, the original task can be decomposed into two subtasks: sensing and transmission. We note that unlike the traditional definition of a task, which treats activities (e.g., sensing and transmission) separately, here a single task corresponds to a logically meaningful and complete data flow. As an example, in Figure 2(a), sensing and transmission activities are traditionally defined as two separate tasks. Here, the whole sensing and transmission data flow

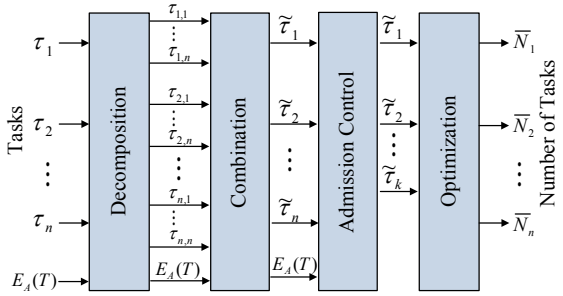


Fig. 4. Design Overview of Dynamic Energy-Oriented Scheduler

is treated as a single task, which starts with CPU activities, followed by sensing activities at the sensor component, then transmission activities at the radio component, and finally ends with CPU activities that handle the success of the radio transmission. The reason why we treat a complete data flow as a single task is to efficiently utilize energy. In the above example, after obtaining the sensing data, if there is insufficient energy to transmit the data on time, then the data is useless and the energy used to conduct the sensing operation is logically wasted.

2) Combination: To save energy, subtasks are combined using two rules: (i) eliminating redundant interactions among hardware components (e.g., CPU and radio), and (ii) allowing concurrent activities of different components. These combined subtasks and the rest of the subtasks from the original task compose a new task.

3) Admission Control: Although subtasks can be combined to save energy, the available energy may still not suffice to schedule all the tasks in the system. Before continuing, the scheduler performs a schedulability test and selects the tasks to execute based on (i) task priority, (ii) the available energy, and (iii) the energy consumption of the tasks.

4) Optimization: The scheduler maximizes the total number of instances a task is executed based on (i) the extra available energy, (ii) the required number of executions of each task, and (iii) the energy consumption of each task.

Since most tasks in sensor networks are periodic, we use T to denote the least common time period of these tasks. DEOS is invoked at the beginning of each time interval T based on the available energy. Detailed designs are described in the following subsections.

B. General Model

This section introduces the general model used in the design of the scheduler. Since the available energy is intermittent, without loss of generality, we assume that within a time interval T the available energy is $E_A(T)$. How to predict the amount of available energy is out of the scope of this paper. Our energy-oriented scheduling design is compatible with energy prediction algorithms in the literature [7], [3], [8], [1].

Assume there are n tasks, labeled $\tau_1, \cdots, \tau_n$. Task τ_i is characterized by the 4-tuple $(N_{i_{\min}}, N_{i_{\max}}, P_i, V_i)$ where:

- $N_{i_{\min}}$ represents the minimum number of instances of τ_i that need to be executed during T in order to prevent system performance from falling below some threshold (e.g., the confidence of the sensing data is below a user specified value).
- $N_{i_{\max}}$ denotes the desirable number of instances of τ_i to be executed within T, as specified by the designers.
- P_i is the priority of τ_i (the larger P_i the lower the priority).
- V_i is the value of τ_i (the larger V_i the more valuable τ_i is).

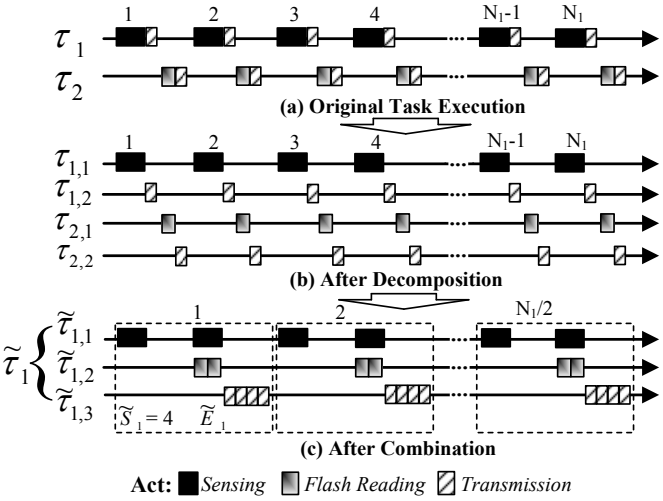


Fig. 5. Example of Task Decomposition and Combination

All task attributes are integer valued and are assigned by the user ahead of time based on application requirements. We note that a higher priority task does not indicate that the task has a higher value. For example, a flash write task consumes 0.127μJ energy per byte, which is less than the energy consumed by a radio transmission task (1.8μJ energy per byte) [9]. For some applications (e.g., habitat monitoring), each piece of data is valuable and the data can be either stored in the node's local flash memory or sent to base station via radio. The flash write task can store more data than the radio transmission task can send using the same amount of energy. Therefore, the value of the flash write task is larger than that of the radio transmission task. However, in some applications (e.g., fire detection), the timely delivery of fresh sensor data is more important, thus the radio transmission task has higher priority than the flash write task.

C. Decomposition and Combination

Based on the general model built in Section III-B, we now introduce the decomposition and combination processes. Before describing them, we introduce the following definition:

Definition 3.1 Decomposable and Combinable Task: *A task is a decomposable and combinable task if it can be divided into subtasks and some of its subtasks can be aggregated or concurrently executed to save energy.*

As an example, Figure 5(a) shows two decomposable tasks τ_1 (SensingToRadio) and τ_2 (FlashToRadio). These tasks can be decomposed into subtasks (i.e., $\tau_{1,1}, \tau_{1,2}$, $\tau_{2,1}$, and $\tau_{2,2}$ as shown in Figure 5(b)). Radio transmission subtasks $\tau_{1,2}$ and $\tau_{2,2}$ can be combined into new subtasks $\widetilde{\tau}_{1,3}$. Similarly, multiple instances of flash reading subtask can be combined. Moreover, the combination function schedules the concurrent execution of sensing and flash reading activities. Since most

applications require periodic sensing, instances of the sensing subtask cannot be combined. After decomposition and combination, combined subtasks (i.e., $\widetilde{\tau}_{1,2}$ and $\widetilde{\tau}_{1,3}$) and an uncombined subtask (i.e., $\widetilde{\tau}_{1,1}$) together form a new task $\widetilde{\tau}_1$.

In general, the size of a new task $\widetilde{\tau}_i$ is denoted as $\widetilde{S}_i$ and equals the maximum number of instances of the original subtasks that can be combined into a new single instance of task. If the new task contains m original tasks $(\tau_1, \cdots, \tau_m)$ and the number of instance of these original tasks is $N_1, \cdots, N_m$, then the total number of instances of the new task $\widetilde{\tau}_i$ is

$$\widetilde{N}_i = \frac{\sum_{l=1}^{m} N_l}{\widetilde{S}_i} \tag{1}$$

The minimum number of instances of $\widetilde{\tau}_i$ that need to be executed during T is

$$\widetilde{N}_{i_{\min}} = \lceil \frac{\sum_{l=1}^{m} N_{l_{\min}}}{\widetilde{S}_i} \rceil \tag{2}$$

The desirable number of instances of $\widetilde{\tau}_i$ to be executed within T is

$$\widetilde{N}_{i_{\max}} = \lfloor \frac{\sum_{l=1}^{m} N_{l_{\max}}}{\widetilde{S}_i} \rfloor \tag{3}$$

The priority of $\widetilde{\tau}_i$ is

$$\widetilde{P}_i = \max\{P_1, \cdots, P_m\} \tag{4}$$

The value of $\widetilde{\tau}_i$ is

$$\widetilde{V}_i = \frac{\sum_{l=1}^{m} V_l \cdot N_l}{\widetilde{N}_i} \tag{5}$$

As an example, in Figure 5, the total numbers of instances of the original tasks τ_1 and τ_2 are both N_1, and the size of the newly formed task $\widetilde{\tau}_1$ is $\widetilde{S}_1 = 4$. Based on Equation (1), the total number of instance of $\widetilde{\tau}_1$ is $\widetilde{N}_1 = (N_1 + N_1)/4 = N_1/2$. The minimum number and desirable number of instances of $\widetilde{\tau}_1$ to be executed within T are $\widetilde{N}_{1_{\min}} = \lceil (N_{1_{\min}} + N_{2_{\min}})/\widetilde{S}_1 \rceil$ and $\widetilde{N}_{1_{\max}} = \lfloor (N_{1_{\max}} + N_{2_{\max}})/S_1 \rfloor$, respectively. The priority and value of $\widetilde{\tau}_1$ are $\widetilde{P}_1 = \max\{P_1, P_2\}$ and $\widetilde{V}_1 = (V_1 + V_2) \cdot N_1/\widetilde{N}_1$, respectively.

Decomposition and combination can be done with the assistance of an energy-aware programming language (such as Eon [10]). For the sake of simplicity and consistency, we use $\widetilde{\tau}_i$ (where $i \in \{1, 2, \cdots, n\}$) to denote the task after the decomposition and combination process. If a task τ_j is not a decomposable and combinable task, then $\widetilde{\tau}_j$ and τ_j represent the same task. The energy consumed by $\widetilde{\tau}_i$ is denoted as $\widetilde{E}_i$, which is the energy consumption of a single instance of the task $\widetilde{\tau}_i$. $\widetilde{E}_i$ can be calculated either online or off-line by using the method introduced in Quanto [11].

D. Admission Control

Due to limited available energy, it may not always be possible to execute all tasks. Moreover it may not be possible to determine whether or not there is sufficient energy to execute a collection of tasks until after the decomposition and combination phases. Therefore, admission control is invoked after the task combination phase. From the original task set, the admission controller selects the tasks to execute based on their priorities and energy utilization, which is defined as follows:

Algorithm 1: Highest Priority First

Input : $\forall i \in \{1, \cdots, n\}$,
Combined Task Set Γ_1,
Minimum Energy Utilization $U_{i_{\min}}$
Output: Schedulable Task Set Γ_2

```
Initialization: Γ2 ← φ, Total Utilization U ← 0 ;
while Γ1 ≠ φ do
    Select the highest priority task τ̃j from Γ1 ;
    U ← U + U_{j_min} ;
    if U ≤ 1 then
        Γ2 ← Γ2 + {τ̃j} ;
        Γ1 ← Γ1 − {τ̃j} ;
    else
        break ;
```

Definition 3.2 Energy Utilization: *The energy utilization of task $\widetilde{\tau}_i$ is the ratio of energy spent in the execution of $\widetilde{\tau}_i$ to the total available energy during the time interval T.*

It is

$$U_i = \frac{\widetilde{N}_i \cdot \widetilde{E}_i}{E_A(T)} \tag{6}$$

Here $E_A(T)$ is the total energy available during T. To reduce energy consumption, tasks executed by wireless sensor nodes are placed in a low duty cycle. Here, duty cycle is defined as follows:

Definition 3.3 Duty Cycle: *Duty cycle is the fraction of time that any component of a system is in an active state during the time interval T.*

Here, the system is in an "active" state whenever any resource, such as CPU, radio, or memory, is active. For example, if a system's CPU is active for 3 seconds and then its radio is active for 2 seconds during a 100 seconds interval, this system's duty cycle is 5%. In another case, if a system's CPU is active for 3 seconds and its radio is active for 2 seconds at the same time, then this system's duty cycle is 3% (due to the concurrent execution). Since sensor nodes can only execute low duty-cycle tasks with limited environmental energy, we define the low duty-cycle task set as follows:

Definition 3.4 Low Duty-Cycle Task Set: *A low duty-cycle task set is a set of tasks for which the sum of their duty cycles is less than 1.*

Since the total duty cycle of the tasks is less than 1, these tasks can be completed during the time interval T if the system has sufficient energy. Definition 3.4 ensures tasks belonging to the low duty-cycle task set can be scheduled without missing their deadlines. However these tasks may not be schedulable due to the energy constraint. In order to efficiently use the precious available energy, the Highest Priority First algorithm (Algorithm 1) is used to select tasks in the admission control stage. The inputs of the algorithm are (i) the combined tasks,

and (ii) the corresponding tasks' minimum energy utilization $U_{i_{\min}}$ which can be calculated as follows

$$U_{i_{\min}} = \frac{\widetilde{N}_{i_{\min}} \cdot \widetilde{E}_i}{E_A(T)} \tag{7}$$

The output of the algorithm is the task set Γ_2, which is schedulable based on the available energy at the current time interval T. In the initialization stage (line 1), the task set Γ_2 is empty and the total energy utilization U is 0. Then the algorithm selects the highest priority task $\widetilde{\tau}_j$ from the combined task set Γ_1 and calculates the total energy utilization U (lines 3 and 4). If U is not greater than 1, the algorithm adds $\widetilde{\tau}_j$ to the task set Γ_2 and removes $\widetilde{\tau}_j$ from the combined task set Γ_1 (lines 5 to 7). The above process is repeated until the combined task set Γ_1 is empty or U is larger than 1. Since there is only a single loop in the algorithm and the maximum number of iterations is n, the time complexity of this algorithm is $O(n)$. Without loss of generality, we assume that k (where $k \leq n$) tasks are selected by the Highest Priority First algorithm. The schedulability of a task set is ensured using the following theorem:

Theorem 3.5: *Tasks selected by the Highest Priority First algorithm is schedulable if and only if*

$$U = \sum_{i=1}^{k} U_i \leq 1; \qquad i \in \{1, \cdots, k\} \tag{8}$$

Where U is the total energy utilization of the system during the time interval T.

Proof: (*Only if.*) We show that tasks cannot be scheduled if $U > 1$. The total demand of energy needed by the selected tasks $(\widetilde{\tau}_1, \widetilde{\tau}_2, \cdots, \widetilde{\tau}_k)$ in T can be calculated as

$$\sum_{i=1}^{k} U_i \cdot E_A(T) = U \cdot E_A(T)$$

If $U > 1$, then $U \cdot E_A(T) > E_A(T)$ which indicates that the total demand requested by these tasks exceeds the available energy $E_A(T)$. Therefore, these tasks cannot be scheduled.

(*If.*) We show sufficiency through contradiction. Assume that the condition $U \leq 1$ is satisfied, however, the selected tasks are still not schedulable. Let $\widetilde{\tau}_m$ be the first task that cannot be scheduled by using the Highest Priority First algorithm due to insufficient energy and all the other tasks $(\widetilde{\tau}_1, \widetilde{\tau}_2, \cdots, \widetilde{\tau}_{m-1})$ can be scheduled. Then the total demand of energy requested by the tasks $(\widetilde{\tau}_1, \widetilde{\tau}_2, \cdots, \widetilde{\tau}_m)$ is $\sum_{i=1}^{m} U_i \cdot E_A(T)$. Since $\widetilde{\tau}_m$ cannot be scheduled, we get the following inequality condition

$$E_A(T) < \sum_{i=1}^{m} U_i \cdot E_A(T) \tag{9}$$

On the other hand, the task set $\{\widetilde{\tau}_1, \widetilde{\tau}_2, \cdots, \widetilde{\tau}_m\}$ is a subset of the selected task set $\{\widetilde{\tau}_1, \widetilde{\tau}_2, \cdots, \widetilde{\tau}_k\}$. Therefore, we get the following inequality condition

$$\sum_{i=1}^{m} U_i \cdot E_A(T) \leq \sum_{i=1}^{k} U_i \cdot E_A(T) = U \cdot E_A(T) \tag{10}$$

From (9) and (10), we get $E_A(T) < U \cdot E_A(T)$. That is equivalent to $U > 1$, which is a contradiction. □

These selected tasks $\widetilde{\tau}_i$ (where $i \in \{1, \cdots, k\}$) are guaranteed to be executed at least $\widetilde{N}_{i_{\min}}$ times. Once all tasks have been executed $\widetilde{N}_{i_{\min}}$ times, there may be extra available energy, which is not sufficient to schedule a new task but can be used to increase the number of times the selected tasks are executed. This extra available energy can be calculated as follows

$$E_E(T) = E_A(T) - \sum_{i=1}^{k} \frac{\widetilde{N}_{i_{\min}} \cdot \widetilde{E}_i}{\widetilde{S}_i} \tag{11}$$

Therefore we need an algorithm to choose additional task copies to maximize the total value of the tasks with an energy constraint. We note that there is an optimization trade-off between priorities and values of tasks. The admission control selects the tasks with higher priorities. After admission control, the selected tasks are ensured to be executed for $\widetilde{N}_{i_{\min}}$ times, which meet the application's requirement. Therefore, maximizing the total value of selected tasks based on the extra available energy is more important.

E. Task Quantity Optimization

The rationale behind the task quantity optimization is to maximize the value of selected tasks based on (i) the quantity of each task, (ii) the values of these tasks, and (iii) the extra available energy. The problem can be formulated as

$$\text{maximize} \qquad \sum_{i=1}^{k} N_i' \cdot \widetilde{V}_i$$

$$\text{subject to} \qquad 0 \leq N_i' \leq (\widetilde{N}_{i_{\max}} - \widetilde{N}_{i_{\min}}) \tag{12}$$

$$\sum_{i=1}^{k} N_i' \cdot \widetilde{E}_i \leq E_E(T) \tag{13}$$

Here N_i' is the number of additional executions of task $\widetilde{\tau}_i$. Constraint (12) ensures that the number of copies of every selected task is less than $\widetilde{N}_{i_{\max}}$. Constraint (13) specifies that the total energy consumption of these selected tasks is less than the extra available energy $E_E(T)$. This corresponds to the Bounded Knapsack Problem, which can be solved using dynamic programming. We use the algorithm proposed in [12], which has running time of $O(kE_E(T))$. For energy constrained sensor nodes, the total number of task types is limited, thus k is typically small. Since the extra available energy ($E_E(T)$) is not sufficient to schedule a new type of task, the value of $E_E(T)$ is also small. Therefore, the time complexity of the algorithm is low, which is corroborated by our measurements of the whole DEOS method's overhead in Section V.

After task quantity optimization, the number of copies of task $\widetilde{\tau}_i$ selected by the scheduler can be calculated as follows

$$\overline{N_i} = \begin{cases} \widetilde{N}_{i_{\min}} + N_i', & i \in \{1, \cdots, k\} \\ 0 \quad , & i \in \{k+1, \cdots, n\} \end{cases} \tag{14}$$

Without loss of generality, we assume that the tasks are sorted based on their index i and the first k tasks are selected by the scheduler.

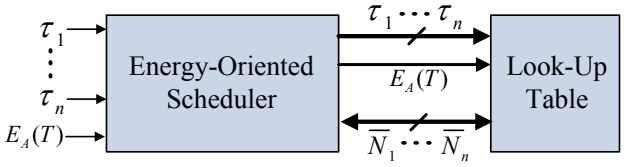


Fig. 6. Look-Up Table Design

IV. Advanced Design

In previous sections, we have introduced the basic design of the DEOS. In this section we describe three improvements.

A. Look-up Table

In wireless sensor networks, most of the tasks are predetermined and are executed for a long time (e.g., several months) after the sensor nodes are deployed. Although the environmental energy changes dynamically, the variation range of the available energy is limited. For example, in Figure 1(b), the available energy only has 57 distinct quantized values during the whole 48-hour experiment period. To reduce computational overhead and improve energy efficiency of the scheduler, we propose a look-up table based approach (shown in Figure 6).

Similar to the look-up table used in computer architecture, when the energy-oriented scheduler receives task set $\{\tau_1, \tau_2, \cdots, \tau_n\}$ and available energy $E_A(T)$ information, it uses this information as an index to check whether the look-up table contains the corresponding result which is the numbers of each task to be executed (i.e., $\{\overline{N_1}, \overline{N_2}, \cdots, \overline{N_n}\}$). If the look-up table already contains the result, then the scheduler skips the basic scheduling operations (i.e., decomposition, combination, admission control and optimization) and directly uses the result in the look-up table to schedule the tasks. Otherwise, the scheduler goes through the basic scheduling operations to obtain the schedule and records this in the look-up table.

In order to reduce the size of the look-up table, we use a bitmap to represent a task set. That is, one bit of data represents a single task. For example, $[010010]$ represents a task set containing 6 tasks among which tasks τ_2 and τ_5 need to be scheduled. To reduce the look-up time, a binary search algorithm is executed, which has time complexity of $O(logI)$. Here I is the total number of indexes.

B. Elimination of Tasks' Energy Uncertainty

In previous sections, we assume that the scheduler has the energy consumption information of all the tasks. However, some new tasks may be added to support new applications after deployment and their energy consumption may be unknown. In this case, the scheduler will try to schedule these new tasks together with other tasks based on their priorities. When the scheduler encounters the first energy uncertain task and there exists some extra available energy during the scheduling stage, it stops scheduling the rest of the tasks and starts to execute the energy uncertain task together with the other already scheduled higher priority tasks. While executing the energy uncertain task, it records the energy consumption of this task using an online energy meter (such as iCount [13]). After measuring the energy consumption of this task, the scheduler schedules this task based on its measured energy consumption and resumes scheduling the rest of the lower priority tasks based on the remaining available energy. When the scheduler encounters the second energy uncertain task, it conducts the same procedure as described above. This process continues until the energy utilization of all the scheduled tasks is larger than or equal to one.

C. Sporadic Task Handler

In previous sections, we introduced the design of energy-oriented scheduling based on the assumption that all tasks are predetermined. However, sporadic tasks may arise in some applications, such as event detection and handling. To manage sporadic tasks, we design an admission controller. When a sporadic task τ_S arrives, the handler will schedule τ_S based on τ_S's priority and deadline. There are three cases:

Case 1: If other tasks have not been scheduled or executed when τ_S arrives, then the scheduler treats τ_S the same as the other regular tasks and executes the Highest Priority First algorithm to select the tasks. τ_S will be selected, if it's priority is high enough. If τ_S's priority is very low, then τ_S is not important. Therefore, it may not be executed when there is insufficient energy.

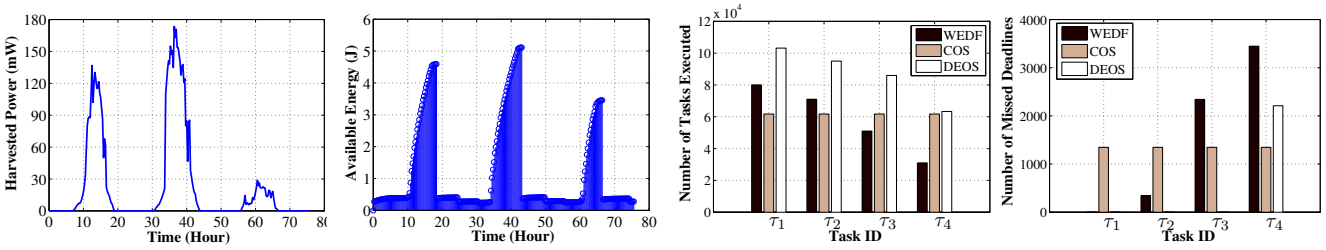
Case 2: Other tasks have already been scheduled at the time of arrival of τ_S and τ_S's deadline is later than the finish time of all the currently scheduled tasks. Therefore, the scheduler treats τ_S the same as the other regular tasks and tries to schedule it in the next scheduling period.

Case 3: Other tasks have already been scheduled when τ_S arrives and τ_S's deadline is within the current scheduling time interval. In this case, the scheduler has to reschedule the tasks based on τ_S's priority and energy consumption. Since all other tasks have already been scheduled, there is no extra expected energy available to schedule τ_S. The scheduler finishes the execution of current instance of the task to avoid thrashing. By doing this, we avoid τ_S's pre-emption in the middle of an unfinished task. Since the energy consumption value of a single instance of the task is very small, the system's performance will not be affected.

The scheduler then sequentially pauses the execution of tasks which have lower priority than τ_S, until the amount of energy obtained from these paused tasks is sufficient to execute τ_S or all the lower priority tasks are paused. If there is still not enough energy to execute τ_S after all the tasks (which have lower priorities than τ_S) are paused, the scheduler will resume these lower priority tasks and try to schedule τ_S in the next scheduling period until either τ_S is executed or τ_S misses its deadline.

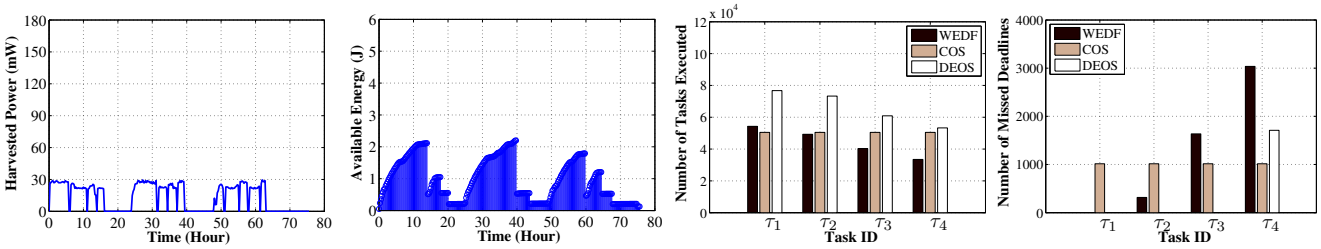
V. Implementation and Evaluation

In this section, we evaluate DEOS performance under different types of environmental energy patterns. We have implemented DEOS using TinyOS and NesC on MICAz motes. The key advantage of the energy-oriented design is its ability to efficiently schedule tasks based on the dynamically available energy. We use the following two metrics to evaluate the performance of our system.



(a) Harvested Power (b) Available Energy (c) Number of Tasks Executed (d) Number of Missed Deadlines

Fig. 7. Outdoor Experiment



(a) Harvested Power (b) Available Energy (c) Number of Tasks Executed (d) Number of Missed Deadlines

Fig. 8. Indoor Experiment

Number of Tasks Executed: the total number of instances of each task that are executed throughout the experiment.

Number of Missed Deadlines: the total number of instances of each task that are not executed before the task's deadline. For example, due to insufficient energy, a task τ_i may only be scheduled or executed for N_i times ($0 \leqslant N_i < N_{i_{min}}$) within the scheduling interval T, which is 5 minutes in our experiment. In this case, we consider that τ_i misses its deadline. The number of τ_i's missed deadlines is $N_{i_{min}} - N_i$.

To compare performance, we also implemented the following two schedulers as baselines.

- **Weighted Earliest Deadline First (WEDF):** With the information of available energy, the scheduler allocates tasks based on their deadlines. If tasks have the same deadline, the task with higher priority will be scheduled earlier.
- **Computing-Oriented Scheduling (COS):** The scheduler allocates tasks based on their computational properties so as to minimize the CPU idle time.

A. Experiment Setup

We ran our system under two different scenarios: outdoors and indoors. These scenarios are carefully selected to represent a wide range of energy harvesting patterns: (i) periodically and dynamically changing energy for the outdoor environment, and (ii) periodic and relatively stable energy for the indoor environment. For each scenario, our TwinStar hardware platform [3] woke up every 5 minutes to predict and record the available energy pattern. The recorded energy information in the outdoor scenario is used as an input for the identical MICAz mote to work under 3 different scheduling methods: WEDF, COS, and DEOS. Similar experiments are conducted for the indoor scenario. Four types of task (summarized in Table I) are executed by the MICAz mote. Each task represents an application as follows:

TABLE I
PROPERTIES OF TASKS

Task Types	$N_{i_{min}}$	$N_{i_{max}}$	Priority (P_i)	Value (V_i)
τ_1	40	150	1	3
τ_2	40	150	2	4
τ_3	40	150	3	2
τ_4	40	150	4	1

Task τ_1 for Event Detection: This task periodically samples a sound sensor and sends out the value via radio.

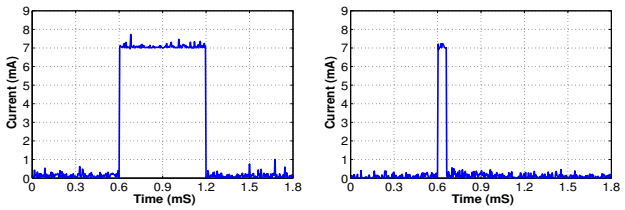
Task τ_2 for Environmental Monitoring: This task periodically samples a temperature sensor and writes the value to flash memory.

Task τ_3 for Time Synchronization: This task periodically sends out a packet containing the sensor node's local time. It is normally used in flooding protocols [14].

Task τ_4 for Data Compression: This task periodically reads an 8 byte temperature values from flash memory, calculates the average value, and then writes the average back to memory.

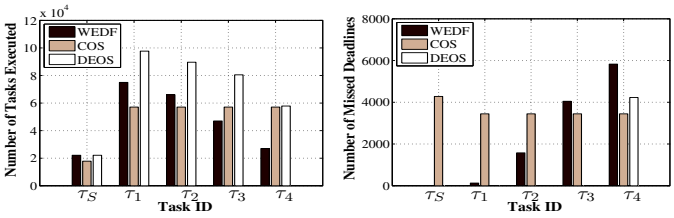
B. Outdoor Experiment

In the outdoor experiment, we deployed our hardware platform outside a fifth-floor apartment for 76 hours. Figure 7(a) shows the energy harvested by the hardware platform in this experiment. The corresponding predicted available energy is shown in Figure 7(b). Figure 7(c) compares the number of tasks executed using weighted earliest deadline first (WEDF), computing-oriented scheduling (COS), and dynamic energy-oriented scheduling (DEOS) methods. By using the Highest Priority First algorithm, WEDF and DEOS allow the MICAz mote to execute a larger number of instances of high priority tasks (i.e., τ_1, τ_2, and τ_3) than COS. For example, the system running DEOS executed τ_1 for 103168 times, which is 28.93% greater than the number of executions of τ_1 under COS. With task combination and concurrent execution, DEOS efficiently utilized energy. As a result, the system running DEOS exe-



(a) Without an Advanced Design (b) With an Advanced Design

Fig. 9. Scheduler Overhead



(a) Number of Tasks Executed (b) Number of Missed Deadlines

Fig. 10. Impact of Sporadic Tasks

cuted more instances of tasks than the system running WEDF and COS.

Figure 7(d) shows the number of instances of tasks that missed their deadlines when the MICAz mote is running WEDF, COS, and DEOS. By scheduling and executing higher priority tasks first and waiving the execution of lower priority tasks during energy deficient stages, tasks τ_1, τ_2, and τ_3 did not miss a deadline during the whole experiment period under DEOS. By contrast, tasks suffered missed deadlines under WEDF and COS. Moreover, without aggregating the computing activities, the system running WEDF consumes more energy than the system running COS. Therefore, the system running WEDF has a greater total number of missed deadlines than the system running COS when energy is a bottleneck. Although τ_4 misses more deadlines under DEOS than under COS, DEOS still executes 63324 instances, which is larger than the required minimum number. Since τ_4 is a data compression task, the delayed execution of τ_4 does not affect the system's behavior.

C. Indoor Experiment

In the indoor experiment, our hardware platform was deployed under an overhead light in our lab for 76 hours. The light was turned on in the morning, when people arrived the lab, and turned off in the middle of the day or during the night when no one was inside the lab.

Figures 8(a) and 8(b) show the energy harvested by the hardware platform and the corresponding predicted available energy, respectively. In Figure 8(a), the fluctuations in the energy level were due to the turning on and off of neighboring overhead lights. Since the light intensity in the indoor environment is much lower than that in the outdoor environment, the available energy in the indoor experiment (shown in Figure 8(b)) is much less than in the outdoor experiment (shown in Figure 7(b)). This results in fewer task executions in the indoor experiment (shown in Figure 8(c)). Interestingly, the indoor experiment has fewer missed deadlines (shown in Figure 8(d)) than the outdoor experiment. This is because the duration of the low available energy stage in indoor experiment is shorter than the one in outdoor experiment.

D. Overhead Analysis

Dynamic energy-oriented scheduling is an online process and the execution of the scheduler itself consumes energy, hence its overhead must be carefully considered. We used an oscilloscope to record the current draw of a MICAz mote during the scheduling process in the indoor experiment. Figure 9 shows the scheduler overhead with and without an advanced design. By using a look-up table, the scheduler can reduce the current draw by almost 11 times. Even without using the look-up table, scheduler overhead is very small. The scheduler draws approximately $7.2mA$ for about $0.5983ms$ (shown in Figure 9(a)), which translates into $12.923\mu J$. In the worst case of the outdoor and indoor experiments, the available energy is $32.845mJ$. Therefore, scheduler overhead is only 0.039%, which is negligible.

E. Simulation Evaluation: Impact of Sporadic Tasks

Sporadic tasks affect the performance of the scheduler. In this section, we use the available energy trace in outdoor experiment as an input and evaluate the impact of sporadic tasks through simulation. Let τ_S denote a sporadic task which sends out emergent data packets. τ_S is generated according to a Poisson process and is given the highest priority. The deadline of τ_S is $0.1T$. In other words, τ_S has to be executed within $0.1T$ after its generation. Every data point on a graph represents the average value of 10 runs, and 95% confidence intervals for the data are within $2 \sim 8\%$ of the mean shown. Figure 10 compares τ_S's impact on the system running WEDF, COS, and DEOS. Compared with the outdoor experiment result without the sporadic task (shown in Figure 7(c)), the sporadic task τ_S causes a reduction in the number of executions of tasks when the system is running WEDF, COS, and DEOS (shown in Figure 10(a)). However, by using DEOS, all instances of tasks τ_S, τ_1, τ_2, and τ_3 are executed before their deadlines (shown in Figure 10(b)). Since τ_4 can be delayed, missing its deadline does not affect the system's behavior. The system running COS does not schedule tasks based on their priority. Therefore, the sporadic task causes the number of missed deadlines to increase by 255.9% when the system is running COS.

VI. Related Work

Scheduling algorithms have been widely used by most modern systems to balance load [15], improve throughput [16], share bandwidth between user flows [17], and reduce pipeline interlock [18]. Furthermore, researchers have proposed specific scheduling methods to improve the performance of particular systems, such as servers [19], routers [20], and storage systems [21]. Despite this rich literature, the existing scheduling methods do not treat energy as a first-class schedulable resource and dynamically schedule tasks based on the changing available energy in the system.

On the other hand, energy management is an intensively studied area. Many solutions have been proposed for different systems, including energy management for data centers [22], storage systems [23], and smart homes [24], [25]. Moreover, some researchers took different approaches by designing multiple task scheduling for data dissemination [26], [27], [28], and developing ultra-low-power energy-harvesting devices [29]. However, none of these energy management solutions investigate scheduling in sustainable sensor networks where environmental energy can change dynamically. The most closely related works are Dewdrop [30], STAM [31], and virtual battery [32]. Dewdrop and STAM only schedule recurring tasks based on the environmental energy. In contrast, DEOS dynamically optimizes energy utilization among multiple tasks for most of hardware platforms in sensor networks. Virtual battery logically divides energy among applications to virtualize their private energy sources. Since virtual battery focuses on energy virtualization and isolation among applications, it is highly complementary to our energy-oriented scheduler which can dynamically schedule multiple tasks in a single application.

VII. Conclusions

This work introduces the concept of dynamic energy-oriented scheduling (DEOS). By conducting decomposition, combination, concurrent execution, admission control, and task quantity optimization, the DEOS method can optimally allocate tasks based on the dynamically changed available energy. Although we design and evaluate the DEOS in sustainable wireless sensor network, the concept of DEOS can also be applied to other energy harvesting embedded systems [33].

To our knowledge, this work is the first to dynamically schedule tasks based on the tasks' energy consumption and the unstable environmental energy. We invested a significant amount of effort to evaluate our design in two real-world settings. The results indicate the effectiveness of our design compared to the other designs.

VIII. Acknowledgements

This work was supported in part by NSF under grant #1019343 to CRA for the CI Fellows Project and Binghamton University academic program and faculty development fund.

References

[1] A. Kansal, D. Potter, and M. B. Srivastava, "Performance aware tasking for environmentally powered sensor networks," in *SIGMETRICS '04*.

[2] X. Jiang, J. Polastre, D. Culler, "Perpetual environmentally powered sensor networks," in *IPSN '05*.

[3] T. Zhu, Z. Zhong, Y. Gu, T. He and Z. Zhang, "Leakage-Aware Energy Synchronization for Wireless Sensor Networks," in *MobiSys '09*.

[4] Z. Zhong, T. Zhu, T. He, and Z. Zhang, "Demo Abstract: Leakage-Aware Energy Synchronization on Twin-Star Nodes," in *SenSys '08*.

[5] A. Rowe, K. Lakshmanan, H. Zhu, R. Rajkumar, "Rate-Harmonized Scheduling for Saving Energy," in *RTSS '08*.

[6] F. Fauberteau, S. Midonnet, and D. Istrate, "Power Saving of Real Time Embedded Sensor for Medical Remote Monitoring," in *ICONS, pp.63-67, 2009 Fourth International Conference on Systems*, 2009.

[7] C. Bergonzini, D. Brunelli, and L. Benini, "Algorithms for harvested energy prediction in batteryless wireless sensor networks," in *IWASI '09*.

[8] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy, "Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems," in *SECON '10*.

[9] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-low power data storage for sensor networks," in *IPSN' 06*.

[10] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, et al, "Eon: A language and runtime system for perpetual systems," in *SenSys '07*.

[11] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," in *OSDI' 08*.

[12] U. Pferschy, "Dynamic programming revisited: Improving knapsack algorithms," in *Computing 63, 419-430*, 1999.

[13] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring," in *IPSN '08*.

[14] T. Zhu, Z. Zhong, T. He and Z.-L. Zhang, "Exploring link correlation for efficient flooding in wireless sensor networks," in *Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2010.

[15] T. Bonald, M. Jonckheere, and A. Proutiere, "Insensitive Load Balancing," in *SIGMETRICS '04*.

[16] P. Chaporkar, K. Kar, X. Luo and S. Sarkar, "Throughput and Fairness Guarantees Through Maximal Scheduling in Wireless Networks," in *IEEE Transactions on Information Theory,Vol. 54, No. 2, Feb. 2008*.

[17] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, "Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing," in *SIGMETRICS '05*.

[18] S. S. Muchnick and P. B. Gibbons, "Retrospective: Efficient Instruction Scheduling for a Pipelined Architecture," in *20 Years of PLDI (1979-1999): A Selection, 2004*.

[19] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers," in *Multimedia '99*.

[20] E. W. Biersack, B. Schroeder, and G. Urvoy-Keller, "Scheduling in practice," in *SIGMETRICS Performance Evaluation Review 34(4): 21-28, 2007*.

[21] E. Bachmat, and J. Schindler, "Analysis of methods for scheduling low priority disk drive tasks," in *SIGMETRICS '02*.

[22] V. Valancius, N. Laoutaris, L. Massouli, C. Diot, and P. Rodriguez, "Greening the Internet with Nano Data Centers," in *CoNext '09*.

[23] G. Mathur, P. Desnoyers, D. Ganesan and P. Shenoy, "CAPSULE: An Energy-Optimized Object Storage System for Memory-Constrained Sensor Devices," in *SenSys '06*.

[24] T. Zhu, S. Xiao, P. Yi, D. Towsley, and W. Gong, "A Secure Energy Routing Protocol for Sharing Renewable Energy in Smart Microgrid," in *IEEE SmartGridComm*, 2011.

[25] T. Zhu, A. Mishra, D. Irwin, N. Sharma, P. Shenoy, and D. Towsley, "The Case for Efficient Renewable Energy Management for Smart Homes," in *ACM BuildSys '11*.

[26] Y. Gu, T. Zhu, and T. He, "ESC: Energy Synchronized Communication in Sustainable Sensor Networks," in *ICNP '09*.

[27] S. Xiong, J. Li, M. Li, J. Wang, and Y. Liu, "Multiple Task Scheduling for Low-Duty-Cycled Wireless Sensor Networks," in *INFOCOM '11*.

[28] T. Zhu and D. Towsley, "E2R: Energy efficient routing for multi-hop green wireless networks," in *IEEE INFOCOM on Computer Communications Workshops*, 2011.

[29] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman, "Challenge: Ultra-Low-Power Energy-Harvesting Active Networked Tags (EnHANTs)," in *MOBICOM '09*.

[30] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: An Energy-Aware Runtime for Computational RFID," in *NSDI '11*.

[31] D. Audet, L. C. de Oliveira, N. MacMillan, D. Marinakis, and K. Wu, "Scheduling Recurring Tasks in Energy Harvesting Sensors," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011.

[32] Q. Cao, D. Kassa, N. Pham, Y. Sarwar, and T. Abdelzaher, "Virtual Battery: An Energy Reserve Abstraction for Embedded Sensor Networks," in *RTSS '08*.

[33] T. Zhu, Y. Gu, T. He and Z.-L. Zhang, "eShare: A Capacitor-Driven Energy Storage and Sharing Network for Long-Term Operation," in *SenSys '10*.