Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 732652, 11 pages
http://dx.doi.org/10.1155/2013/732652

Research Article

A Nonclairvoyant Real-Time Scheduler for Ambient Energy Harvesting Sensors

Hussein El Ghor,[1,2] Maryline Chetto,[1] and Rafic Hage Chehade[2]

[1] *IRCCyN Lab, University of Nantes, 1 rue de la Noë, 44321 Nantes, France*
[2] *Lebanese University, IUT Saida, Saida 813, Lebanon*

Correspondence should be addressed to Hussein El Ghor; hussein.ghor@ul.edu.lb

Received 3 March 2013; Revised 22 April 2013; Accepted 25 April 2013

Academic Editor: Guangjie Han

Ambient energy harvesting also known as energy scavenging is the process where energy is obtained from the environment, converted, and stored to power small devices such as wireless sensors. We present a variant of EDF scheduling algorithm called EH-EDF (Energy Harvesting-Earliest Deadline First). Decisions are taken at run-time without having prior knowledge about the future energy production and task characteristics. We gauge the performance of EH-EDF by means of simulations in order to show its benefits. We evaluate and compare several variants of EH-EDF in terms of percentage of feasible task sets. Metrics such as average length of the idle times are also considered. Simulations tend to demonstrate that no online scheduler can reach optimality in a real-time energy harvesting environment.

1. Introduction

An algorithm is said to be nonclairvoyant if its scheduling decisions are taken at run-time with no prior knowledge about the characteristics of the future tasks [1]. Consequently, a nonclairvoyant scheduling algorithm is necessarily online. The problem of online scheduling in real-time systems has been a fertile ground for theoretical research for many years.

There are many real-time applications concerned with nonpredictability and consequently with nonclairvoyant scheduling. In that system, periodic and aperiodic tasks coexist. Periodic tasks typically arise from sensor data or control loops at regular intervals. In contrast, aperiodic tasks generally arise from arbitrary events (external interrupts).

When considering real-time systems that take time as the only limiting factor, it is important to differentiate between underloaded and overloaded real-time systems. A real-time system is said to be underloaded if there exists a feasible schedule for the workload; that is, the deadlines of all tasks are met under timing constraints. On the contrary, overloaded real-time systems do not have a feasible schedule where all tasks meet their deadlines. Thus, the objective will be to optimize some criteria such as the ratio of deadline success.

In addition, real-time systems can be classified into three categories: hard, soft, and weakly hard. In hard real-time systems, all tasks must be guaranteed to complete within their deadlines. For soft real-time systems, it is acceptable to miss some of the deadlines occasionally with additional value for the system to finish the task, even if it is late. In weakly hard real-time systems, tasks are allowed to miss some of their deadlines but there is no associated value if they finish after the deadline.

Real-time task scheduling determines the order in which tasks have to be executed. The well-known scheduling algorithm is the Earliest Deadline First (EDF) algorithm [2]. EDF schedules at each instant of time t the ready task whose deadline is closest to t. EDF algorithm is optimal in underloaded settings; that is, EDF is guaranteed to meet all the task deadlines for any feasible task set.

Nowadays, energy management is becoming the central topic of research in real-time systems. In today's applications, most real-time embedded systems are powered by batteries. Therefore, great interest has risen in powering these systems by renewable energy sources. Many energy harvesting methods can be used to harvest energy from a controlled or ambient environment either to power devices directly or to store

the energy in capacitors or batteries for later use. Radio-frequency- (RF-) powered systems, solar-powered systems, wind-powered systems, motional energy harvesting systems, thermoelectric-powered systems, and piezoelectric conversion systems are examples of such methods. These harvesting methods support a wide range of applications such as Heliomote [3] and Prometheus [4] and can also be used to increase the lifetime of preexisting devices.

Recently, we addressed the scheduling problem for a uniprocessor platform that is powered by a renewable energy storage unit and uses an harvester such as photovoltaic cells. We presented a scheduler called EDeg (Earliest Deadline with energy guarantee) [5]. The set of tasks is perfectly known offline as in applications where all the tasks run periodically. EDeg is clairvoyant since it must know in advance both the energy source profile and the characteristics of the tasks (arrival time).

To extend the applicability of EDeg scheduling framework, we need to adapt it to situations where the scheduler has to take decisions without a priori knowledge of the future. For real-time energy harvesting applications, a scheduling algorithm will be nonclairvoyant if, in addition, it ignores the incoming environmental energy in the future. We may imagine an application where either the set of tasks or the future energy profile is known but not both.

We focus here on on-line nonclairvoyant scheduling in an underloaded real-time energy harvesting system that executes aperiodic tasks on a uniprocessor platform. We propose a scheduling algorithm named Energy Harvesting-Earliest Deadline First (EH-EDF) which extends the well-known EDF algorithm. We modify EDF so as to count for the limitation of energy. We benefit from a slack-based method to let the processor idle and thus to recharge the energy storage unit as much as possible without violating deadlines.

The remainder of the paper is organized as follows. In the next section, we summarize the related work. The system model and necessary terminology are introduced in Section 3. In Section 4, we present the fundamental concepts about the slack time. Section 5 describes our scheduling scheme, EH-EDF, with some indications about practical issues. Section 6 illustrates the simulation study, whereas the preliminary results are presented in Section 7. Section 8 concludes the paper and gives some new directions for future work.

2. Literature Review

Most of the previous research work around real-time scheduling disregards energy management or assumes that the energy is not a limiting factor for task execution.

Energy consideration is now added as a crucial issue because of the great advances in both hardware and software technology. This enables system designers to develop large, complex embedded systems. Such systems consume a large amount of power and rely mainly on a limited energy storage. Many technical challenges lie ahead in order to make an energy harvesting system work effectively. Among them is to either minimize the total energy consumption without violating deadlines or maximize the performance of hard energy constrained systems with a fixed energy budget.

With the goal to minimize the total energy consumption, Pillai and Shin [6] present several novel algorithms for real-time dynamic voltage scaling called real-time DVS (RT-DVS). They modify the OS's real-time scheduler and task management service in order to achieve significant energy savings without violating deadlines. Later, Aydin et al. [7] address the problem of power-aware scheduling for periodic tasks with the aim to reduce CPU energy consumption by the help of dynamic voltage scaling. The authors propose an offline algorithm to compute the optimal speed, assuming worst-case workload for each arrival. An online speed reduction mechanism is introduced to recompute energy based on the actual workload. The third component in this solution is to perform a speculative speed adjustment mechanism based on the expected workload. Unlike the work in [6], Aydin et al. [8] take into account the frequency-dependent and -independent power components as well as the power consumption of components other than the CPU when addressing the problem of minimizing overall energy consumption.

Many other studies address the ways to maximize the system performance of underloaded real-time systems that have to operate under a fixed energy budget.

Moser et al. [9] give an optimal scheduling algorithm called LSA for tasks with deadlines, periodic or not, that run on a monoprocessor device that is powered by a rechargeable storage unit. They consider that the source power is predictable but time varying. LSA can be considered as an idling variant of EDF. The system starts executing a task only if the task has the earliest deadline among all ready tasks, and the system can keep on running at the maximum power until the deadline of the task. In that work, the consumption power of the computing system is characterized by some maximum value which implies that for every task, its total energy consumption is directly connected to its execution time through the constant power of the processing device. The main disadvantage of this work lies in that the LSA algorithm executes tasks at full power. Moreover, in practice, the total energy consumed by a task is not necessarily proportional to its execution time.

In [5], we relax the restrictive hypothesis that links energy requirement and execution time of tasks. We present a scheduling algorithm called EDeg (Earliest Deadline with energy guarantee). Simply executing tasks according to the EDF rule either as soon as possible (EDS) or as late as possible (EDL) may lead to violate some deadlines. EDeg executes tasks according to the EDF rule with idling phases and relies on two fundamental concepts, namely, slack time and slack energy. Before authorizing a task to execute, we must ensure that the energy availability will permit to execute all future occurring tasks and the current highest priority one. When this condition is not verified, the processor has to stay idle so that the storage unit recharges as much as possible and as long as all the deadlines can still be met despite execution postponement. In [10], we prove the efficiency of this scheduler through a simulation study. EDeg is clearly clairvoyant since it needs both the characteristics of the future occurring tasks and prediction about the future incoming energy.

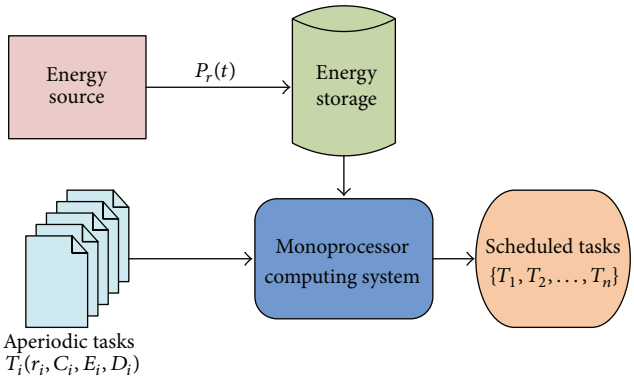

Figure 1: Real-time energy harvesting system.

3. System Model and Terminology

3.1. Task Set. We consider a set of aperiodic tasks that execute on a uniprocessor platform as depicted in Figure 1. Each task is known by the system at the time of its arrival. An aperiodic task set can be denoted as follows: $\Psi = \{\tau_i, i = 1, \ldots, n\}$. Every task τ_i is characterized by (r_i, C_i, D_i, E_i), where r_i represents the arrival time of task τ_i. In the worst case, the execution of τ_i requires a Worst Case Execution Time (WCET) of C_i time units. And it consumes a Worst Case Energy Consumption (WCEC) given by E_i. We assume that the WCEC of a task has no relation with its WCET. A deadline for τ_i occurs at time D_i by which the task should complete its execution. We assume that $0 \leq C_i \leq D_i - r_i$ for each $1 \leq i \leq n$.

Definition 1. The processor load L_p of a task set Ψ gives the processor utilization of Ψ:

$$L_p = \sum_{i=1}^{n} \frac{C_i}{D_{\max}}, \tag{1}$$

where $D_{\max}$ represents the longest deadline in Ψ.

Definition 2. The energy load L_e, measured in joules/s or energy unit/time unit, gives the average power consumed by Ψ:

$$L_e = \sum_{i=1}^{n} \frac{E_i}{D_{\max}}. \tag{2}$$

3.2. Energy Source. We assume that the ambient energy is harvested and converted into electrical power. We cannot control the energy source but we can predict the expected availability with a lower bound on the harvested source power output, namely, $P_r(t)$. Generally, the harvested power is time varying including solar energy which can be assumed constant on average in a long-term perspective. However, on a short-term perspective, the harvested power is highly unstable. This power is then the instantaneous charging rate that incorporates all losses caused by power conversion and charging process. Clearly, we make no assumption about the nature and dynamics of the energy source, making our approach more easily implemented in real systems where data about the energy source may not be available beforehand.

3.3. Energy Storage. We consider an ideal energy storage unit (supercapacitor or battery) of nominal capacity E, corresponding to a maximum energy (expressed in Joule or energy unit). The energy level has to remain between two boundaries $E_{\min}$ and $E_{\max}$ with $E = E_{\max} - E_{\min}$. The stored energy may be used at any time later and does not leak any energy over time. If the storage is fully charged and we continue to charge it, energy is wasted. In contrast, if the storage is fully discharged, no task can be executed.

At some time t, the stored energy is denoted as $E(t)$. At any time, the stored energy is no more than the storage capacity; that is,

$$E(t) < E \quad \forall t. \tag{3}$$

Considering a task set $\Gamma = \{\tau_i = (r_i, C_i, D_i, E_i) \mid i = 1 \ldots n\}$, we want to compute the remaining energy in the energy storage unit at time t. We assume that the energy storage capacity is equal to E energy units at $t = 0$. Let τ_i be the highest priority instance ready at time $t = 0$. As tasks are ordered according to their deadline under EDF, τ_i must be run first. The remaining energy in the energy storage unit at time $t = C_i$ is

$$E(t) = E + \int_0^{C_i} P_r(t)\, dt - E_i. \tag{4}$$

4. Fundamental Concepts

4.1. Slack Time. The slack time of a hard deadline task set at current time t is the length of the longest interval starting at t during which the processor can stay idle without leading to deadline violations.

Let us consider a task set Ψ as described previously. Let Ψ' be the set of tasks ready to be processed at current time t. And let us define the slack time of task τ_j as the maximum processor time that can be used after executing τ_j and higher priority tasks. Then the slack time of $\tau_j \epsilon \Psi'$ is computed as follows:

$$slack \cdot time\left(\tau_j, t\right) = \left(D_j - t\right) - \sum_{D_i \leq D_j} C_i. \tag{5}$$

It comes that the slack time of the system at time t is computed from the slack time of all the tasks as follows:

$$slack \cdot time\,(t) = \min\left(slack \cdot time\left(\tau_j, t\right)\right). \tag{6}$$

4.2. Illustrative Example 1. Consider a task set $\Psi = \{\tau_i, i = 1, \ldots, 4\}$ with $\tau_i = (r_i, C_i, D_i)$. Let $\tau_1 = (0, 3, 18)$, $\tau_2 = (4, 2, 12)$, $\tau_3 = (5, 3, 24)$, and $\tau_4 = (0, 4, 16)$. Let us compute the slack time at time 6 after executing the tasks according to EDS from 0 to 6.

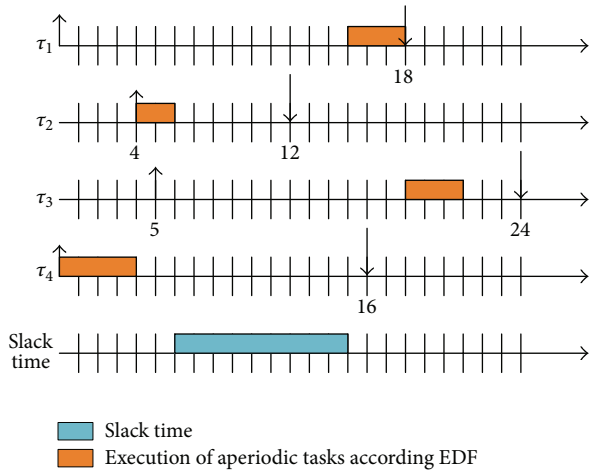

FIGURE 2: Computing the slack time at $t = 6$.

τ_4 is executed from time 0 to time 4 and τ_2 from time 4 to time 6. At time 6, tasks τ_1 and τ_3 are both ready for execution. Their slack time and the slack time of the system are computed according to (5) and (6):

$$slack \cdot time(\tau_1, 6) = (D_1 - 6) - \sum_{D_i \le D_1} C_i = (18 - 6) - 3 = 9,$$

$$slack \cdot time(\tau_3, 6) = (D_3 - 6) - \sum_{D_i \le D_3} C_i = (24 - 6) - 6 = 12,$$

$$slack \cdot time(6) = \min(slack \cdot time(\tau_1, 6), slack \cdot time(\tau_3, 6)) = 9. \quad (7)$$

Figure 2 describes the resulting schedule where the processor is let idle from time 6 during a time interval whose length equals the slack time. We note that τ_1 starts execution at the latest time while τ_3 has a slack equal to 3 time units before deadline. This corresponds to the slack time of τ_3 minus the slack time of the system.

4.3. Illustrative Example 2. Consider the same task set as in Section 4.2. Nevertheless, we add a task $\tau_5 = (8, 5, 20)$. Assume that we execute τ_4 from time 0 to time 4 and then τ_2 from time 4 to time 6. When computing the slack time at time 6, we have two ready tasks τ_1 and τ_3 with $slack \cdot time(\tau_1, 6) = 9$ and $slack \cdot time(\tau_3, 6) = 12$. Consequently $slack \cdot time(6) = 9$.

τ_5 is released at $t = 8$. This leads to the update of the slack time (Figure 3). First, we note that the slack time function linearly decreases with time when the processor is let idle. And the slack time of a task is only affected by tasks with a lower deadline. As the deadline of τ_3 is lower than the deadline of the new occurring task τ_5, we deduce that $slack \cdot time(\tau_3, 8) = slack \cdot time(\tau_3, 6) - 2 = 10$.

As the deadline of τ_1 is greater than the deadline of the new occurring task τ_5, the computation of the slack time of τ_1 must be achieved thanks to (5):

$$slack \cdot time(\tau_3, 8) = (24 - 8) - (3 + 3 + 5) = 5. \quad (8)$$

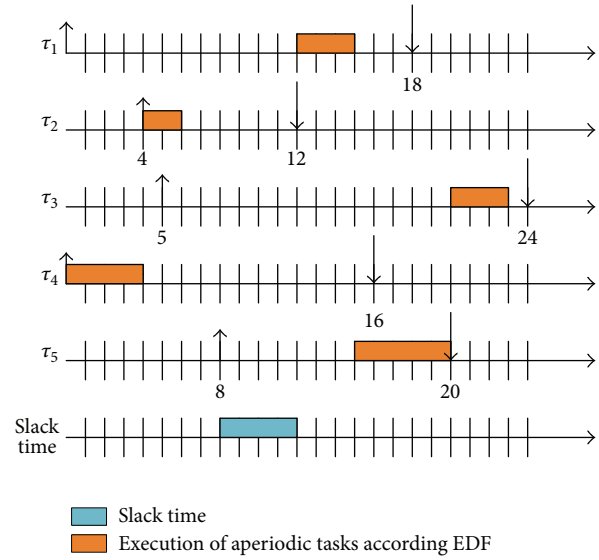

FIGURE 3: Updating the slack time at $t = 8$.

By the help of (5), we have $slack \cdot time(\tau_5, 8) = 4$. Thus the slack time of the system will be changed to 4 in order to meet the deadline of the new occurring task (τ_5).

Now, we are prepared to introduce a new online scheduler specifically adapted to aperiodic tasks in an energy harvesting context.

5. The EH-EDF Scheduling Algorithm

In this section, the scheduler ignores the future energy production and the future arrival times of tasks.

5.1. Presentation of the Scheduler. The intuition behind EH-EDF algorithm is to schedule aperiodic tasks as soon as possible according to EDF. When a new task arrives, it is inserted in the ready task list. When the energy in the storage unit reveals to be insufficient for executing tasks, the only solution consists in postponing them as much as possible. We have to perform the computation of the slack time of the system from the ready task list. The scheduler lets the processor idle until the energy storage unit replenishes or the slack time becomes zero.

The slack time is updated whenever a new task arrives even in the recharging phase. The processor continues idling as long as the system has slack.

We propose the so-called *Energy Harvesting-Earliest Deadline First (EH-EDF) algorithm* following the idea described previously.

The major components of the EH-EDF algorithm are $E(t)$ and $slack \cdot time(t)$. $E(t)$ is the residual capacity of the storage unit at time t which is the energy that is currently stored and $slack \cdot time(t)$ is the slack time of the system at current time t. PENDING is a Boolean which equals true whenever there is at least one instance in the ready list queue. We use the function wait() to put the processor in sleep mode and

Input: A Set of aperiodic Tasks $\Psi = \{\tau_i \mid \tau_i = (r_i, C_i, E_i, D_i)\ i = 1, \ldots, n\}$
Scheduled according to *EDF*, current time t, battery with capacity ranging from $E_{\max}$ to $E_{\min}$, energy level of the battery $E(t)$, source power $P_r(t)$.
Output: EH-EDF Schedule.
(1) **while** "(1)" **do**
(2) **while** "PENDING=true" **do**
(3) **while** "$(E(t) > E_{\min})$" **do**
(4) execute()
(5) **end** "**while**
(6) **while**" $(E(t) < E_{\max}$ and $Slack \cdot time(t) > 0)$ "**do**
(7) wait()
(8) **end**" **while**
(9) **end** "**while**
(10) **while**" PENDING=false "**do**
(11) wait()
(12) **end**" **while**
(13) **end** "**while**

Algorithm 1: Energy Harvesting-Earliest Deadline First (EH-EDF).

function execute() to put the processor in active mode and schedule the tasks according to EDF.

The framework of the EH-EDF scheduling algorithm is as Algorithm 1.

From the EH-EDF framework, we notice that tasks do not run after $E_{\min}$. EH-EDF charges the energy storage to the maximum level, provided there is sufficient slack time and the storage unit is not fully replenished. Such condition can be easily detected through an interrupt mechanism and adequate circuitry between the storage unit and the processing device. The slack time is computed when entering the *wait* state and decremented at each time instant.

Therefore, we waste recharging power only when there are no pending tasks in the ready list and the storage unit is full.

5.2. Illustrative Example. Consider a task set Ψ with five aperiodic tasks as in the previous example such that $\Psi = \{\tau_i \mid 1 \le i \le 5\}$, where $\tau_i = (r_i, C_i, D_i, E_i)$. Let $\tau_1 = (0, 3, 18, 9)$, $\tau_2 = (4, 2, 12, 12)$, $\tau_3 = (5, 3, 24, 7)$, $\tau_4 = (0, 4, 16, 10)$, and $\tau_5 = (8, 3, 20, 10)$. The energy storage capacity is assumed to be equal to 10 energy units. For sake of simplicity, the rechargeable power, P_r, is constant along time and equals 2.

Ψ is temporally feasible; that is, all deadlines can be met when abstracting for energy. But Ψ reveals to be not feasible with energy limitations since the storage unit empties at time 6.

When applying EH-EDF (Figure 4) to Ψ, the energy storage capacity empties at $t = 6$. The energy storage recharges as much as possible. The recharging time is computed from the current *slack time* in order to still guarantee all the deadlines while avoiding energy overflow.

In details, the energy storage is full at time 0. The highest priority task T_4 executes until time 4 when the energy storage capacity is given by the following formula: $E(4) = E_{\max} - E_1 + P_r C_1 = 8$ energy units.

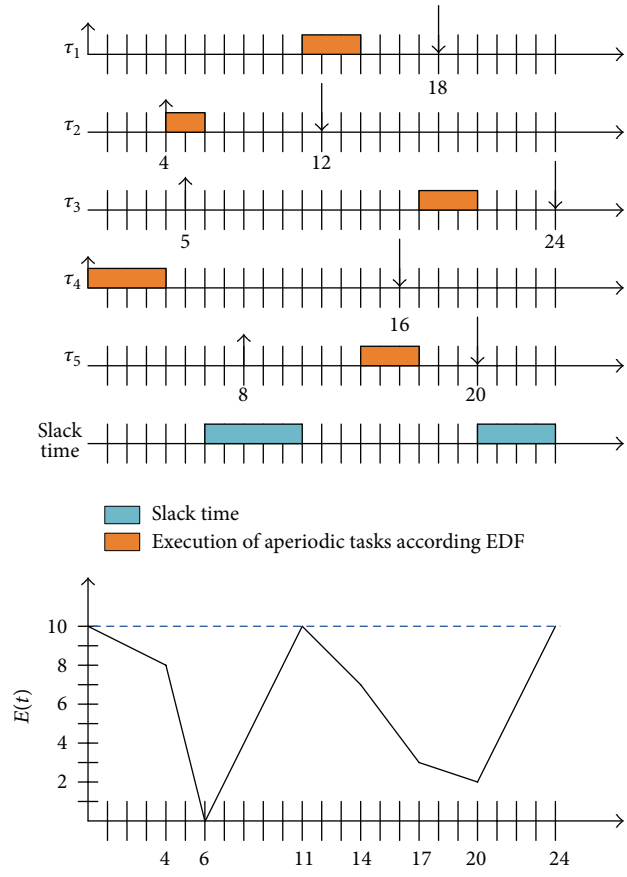

Figure 4: Example on EH-EDF scheduling.

T_2 is ready at time 4. As the highest priority task, it executes until time 6 when the energy storage empties. The processor has to remain idle as long as the storage has not fulfilled and the slack time is not zero. According to (5), the slack time of all released tasks and the slack time of the system are computed.

T_1 and T_3 are released at time 6. The slack time of T_1 and T_3 is equal to 9 and 15, respectively. As $slack \cdot time(6) = 9$, the processor has to stay idle until time 15 for recharging the energy storage unit.

T_5 is released at time 8. As the slack time for T_5 is 6, $slack \cdot time(8) = 6$. The battery is recharged until time 11 when it is full. Thus, we stop recharging at time 11 to avoid wasting energy.

At time 11, the energy storage is equal to 10 energy units, and T_1 has the highest priority. It executes until time 14 and the remaining energy $E(14) = 7$ energy units. T_5 is then the highest priority task and executes until time 17 when the energy level equals 3 energy units.

At time 17, T_3 executes until time 20 where the energy level equals 2 energy units. The processor has no task to execute and remains idle until time 24 where the energy storage is full again.

In contrast to EDF, EH-EDF feasibly schedules the task set Ψ given the characteristics of the storage unit and the power source profile.

6. Simulation Study

This section describes experiments that have been conducted to evaluate the Energy Harvesting-EDF (EH-EDF) algorithm. To measure the effectiveness of EH-EDF, we develop a discrete-event simulation in C/C++. We report a performance analysis which consists of five experiments.

The simulation environment consists of a simulation kernel (scheduler) with a number of components involved in the management and analysis of simulations. The main components are the task generator, the scheduler, and the CPU.

The generator of aperiodic tasks has been designed to accept the following input parameters: the number of desired tasks n, the processor load L_p, the energy load L_e, and the recharging power $P_r(t)$. The output is a task set $\Psi = \{\tau_i(r_i, C_i, D_i, E_i),\ i = 1\ to\ n\}$. The execution time of tasks is randomly generated such that $L_p = \sum_{i=1}^{n}(C_i/D_{\max}) \leq 1$. Moreover, the energy consumption of tasks is randomly generated from the energy load factor such that $L_e = \sum_{i=1}^{n}(E_i/D_{\max}) \leq P_r$. Deadlines are greater than or equal to the computation times.

Simulation results are then ordered to excel files to be stored and analyzed.

6.1. Formal Definition of Scheduling Strategies. For the sake of comparison, we implement five energy harvesting scheduling policies where aperiodic tasks execute as soon as possible according to *EDF*.

EH-EDF: when the battery empties, the processor is put into sleep mode until the battery replenishes or the slack time becomes zero.

EH-EDFx: when the battery empties, the processor is put into sleep mode for x units of time where x is an input of the scheduler.

EH-EDF1: when the battery empties, the processor is put into sleep mode until the energy level reaches a threshold value, namely, E_{th}, given as an input of the scheduler.

EH-EDF2: when the battery empties, the processor is put into sleep mode until the slack time becomes zero regardless of the energy level.

EH-EDF3: there are two threshold parameters, namely, $E\text{th}_{\min}$ and $E\text{th}_{\max}$. when the energy level reaches $E\text{th}_{\min}$, the system is put into sleep mode until either the slack time be null or the energy level be $E\text{th}_{\max}$.

6.2. Measurement Support

6.2.1. Aperiodic Task Sets Generation. We use a simulator that generates 50 tasks with maximum deadline equal to 3360. The worst-case computation times are set according to the processor load L_p, where L_p can be 30%, 60%, or 90%. Results presented in this section are averages over groups of fifty task sets.

6.2.2. Energy Parameters Generation. The energy consumptions of tasks (WCEC) are randomly generated but constrained by the energy load L_e. All tasks are assumed to linearly consume their energy budget over time. In addition, all tasks are dischargeable. This means that E_i/C_i is greater than $P_r(t)$ for all t. The rechargeable power is constant along time during the execution of a task and varies from one task execution to another. A random generator enables us to produce for every quantum of time a power energy profile with minimum value 10 and a maximum value here 35.

6.2.3. Simulations Description. We start the simulation with a battery fully recharged ($E(0) = E_{\max}$). When a deadline is missed, we discard the task and update the slack time. The simulation is repeated for 50 task sets for a given processor and energy utilization ratio. For a fair comparison of the previous strategies, all simulations are performed under the same conditions. We report the performance analysis that consists of the following measures:

(i) percentage of feasible task sets;

(ii) the impact of the slack time and energy storage capacity on the performance of EH-EDF;

(iii) average idle time corresponding to recharging phases of the energy storage;

(iv) energy storage low level.

The above measurements are compared under different scenarios for the five energy harvesting scheduling policies stated previously. These policies cover all the possibilities of the EH-EDF algorithm. We measure the impact of the slack time and energy storage capacity on the performance of EH-EDF, EH-EDFx, and EH-EDF1.

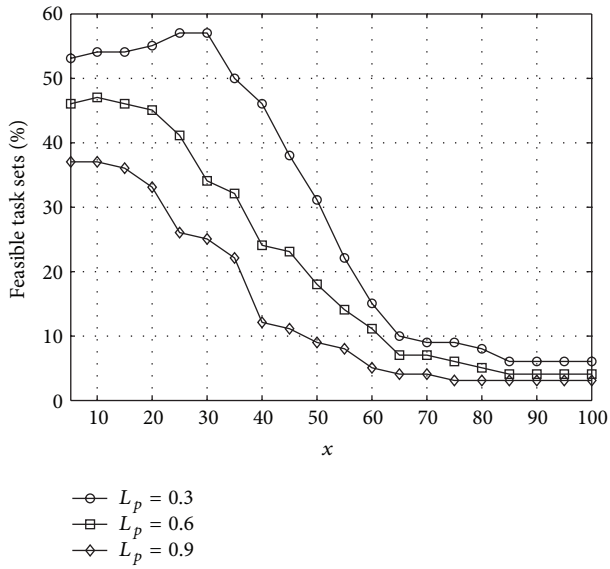

Figure 5: Effects of parameter x on EH-EDFx ($E_{\max} = 100$).

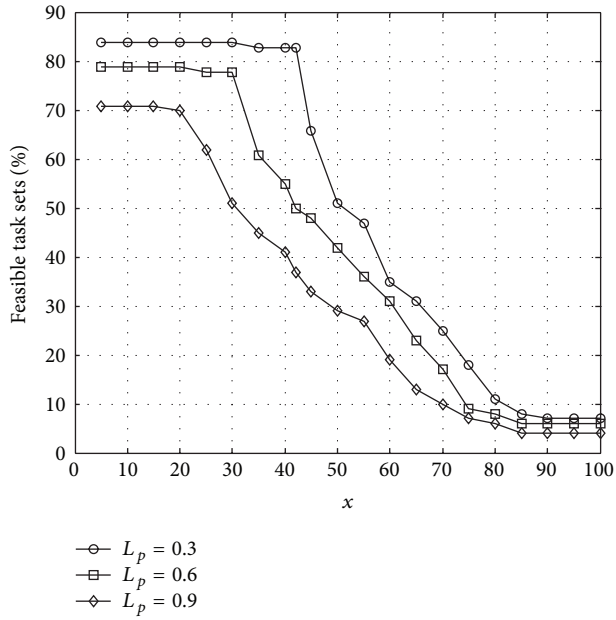

Figure 6: Effects of parameter x on EH-EDFx ($E_{\max} = 200$).

7. Preliminary Results

7.1. Impact of Parameter x on EH-EDFx. In this section, we experiment on the effect of the slack time (x) on EH-EDFx. We report the results of this simulation study where the processor load L_p is set to 0.3, 0.6, and 0.9, respectively. The rechargeable power is constant during execution of a task and varies between a task and another. We took a random function that randomly gives a number between 10 and 35. The maximum ambient power is 35. So all tasks are discharging tasks ($L_e \leq 35$). Simulations are performed first with $E_{\max} = 100$ (Figure 5) and second with $E_{\max} = 200$ (Figure 6).

When L_p is set to 0.3, EH-EDFx benefits from the high idle time to recharge the energy storage. Thus, any parameter x will be acceptable to recharge the battery without violating deadlines till $x = 30$. After this value, the percentage of feasible task sets begins to decrease, and a higher number of deadlines are missed.

When L_p is set to 0.6, the total idle time decreases. We observe that the performance of EH-EDFx is roughly constant until x reaches 20 where the number of violated deadlines begins to increase.

At higher values of processor load, the performance of EH-EDFx is approximately constant until x reaches a value of 15 where a higher number of deadlines are violated.

In the second experiment, we double the size of the energy storage unit ($E_{\max} = 200$) while keeping the other parameters unchanged.

When L_p is set to 0.3, any parameter of x will be acceptable to recharge the energy storage without violating deadlines till a high value of $x = 42$ where the percentage of feasible task sets begins to decrease. The percentage of feasible task sets is 84% when $x = 42$ which is approximately 58% more than in the case when $E_{\max} = 100$. This is because as the size of the energy storage increases, EH-EDFx will be able to execute more tasks, and consequently the percentage of feasible task sets will increase. As L_p increases, the percentage of feasible task sets is, respectively, 44% and 50% more for $L_p = 0.6$ and 0.9 than in the case when $E_{\max} = 100$.

We conclude that the slack time and the energy storage capacity have a great impact on the system performance. As we increase the energy storage size, the mean system life time increases, but without reaching optimality.

7.2. Impact of Parameter E_{th} on EH-EDF1. In this section, we experiment on the effect of parameter E_{th} on EH-EDF1 (Figures 7 and 8). We report the results of this simulation study where the processor load L_p is set to 0.3, 0.6, and 0.9, respectively. Simulations are performed first for $E_{\max} = 100$ (case a) and second for $E_{\max} = 200$ (case b).

For $E_{\max} = 100$, we observe that EH-EDF1 gives approximately an average constant performance until $E_{\text{th}} = 0.2E_{\max}$. In details, when L_p is set to 0.3, the percentage of feasible task sets for EH-EDF1 is constant until a critical value of E_{th} (15% for case (a) and 25% for case (b)). After this critical value, the performance increases without reaching optimality. When $L_p = 0.6$, the performance of EH-EDF1 is constant until a critical value of E_{th} (20% for case (a) and 30% for case (b)). As L_p increases, the percentage of feasible task sets decreases until it reaches a maximum (76% for case (a) and 89% for case (b)).

As a conclusion, we demonstrate through the previous simulations that the slack time and energy storage capacity have a great effect on the performance of EH-EDF algorithm. In addition, we note that EH-EDFx and EH-EDF1 give approximately the same performance levels in terms of deadline missings.

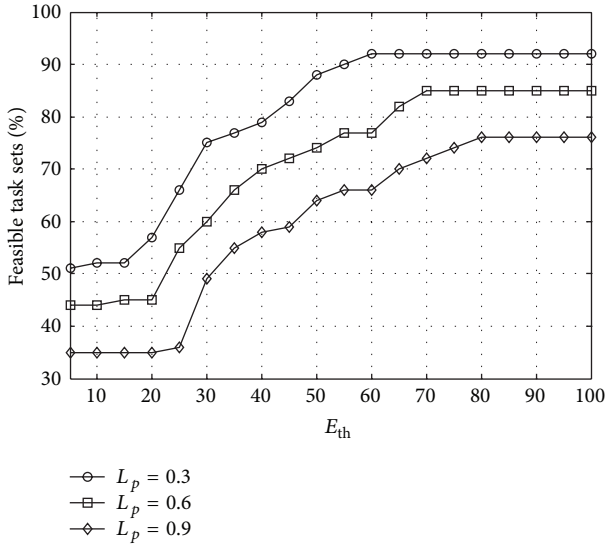

Figure 7: Effects of parameter E_{th} on EH-EDF1 (case a).

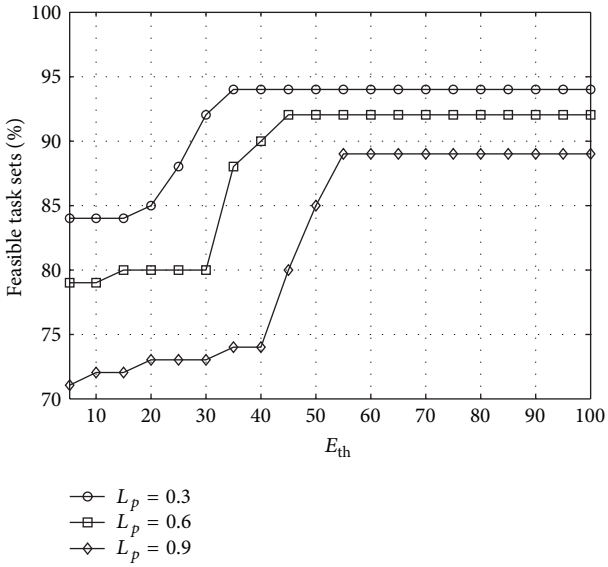

Figure 8: Effects of parameter E_{th} on EH-EDF1 (case b).

7.3. Percentage of Feasible Task Sets. In this section, we experiment on task sets which are feasible. Simulations are performed by varying $E_{\max}$. Based on the previous simulations, we choose $x = 20$ for EH-EDFx and $E_{\text{th}} = 25\%$ for EH-EDF1 and EH-EDF3. We report the results of this simulation study where the processor load (L_p) is set to 0.3, 0.6, and 0.9, respectively. EH-EDF2 is eliminated from this section due to its poor performance. This proves that the maximum upper bound for the energy storage capacity has a great impact on the performance.

First, we consider that $E_{\max} = 100$ (Figure 9). Our experiment demonstrates that EH-EDF outperforms the other policies. This is because EH-EDF will benefit from the idle time to recharge the energy storage capacity without violating deadlines.

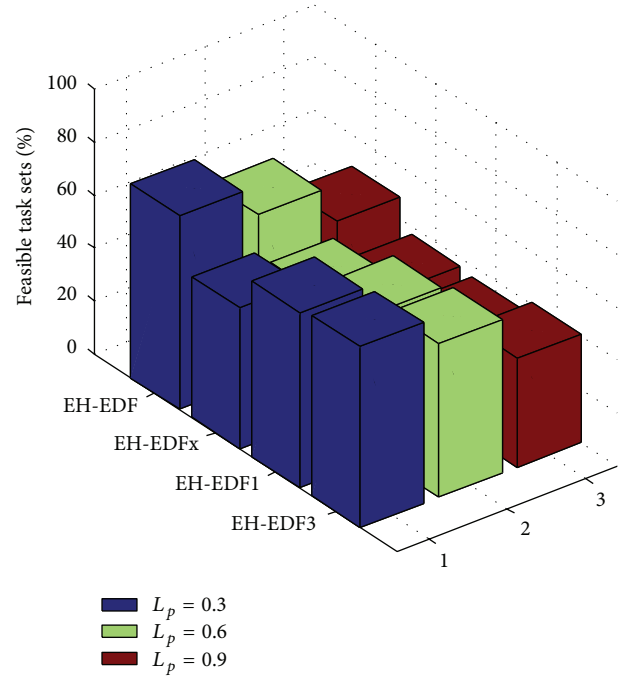

Figure 9: Percentage of feasible task sets ($E_{\max} = 100$).

In details, when L_p is set to 0.3, EH-EDF proves to have the highest percentage of feasible task sets with 6.9%, 9.5%, and 27.4% more than EH-EDF1, EH-EDF3, and EH-EDFx, respectively. As L_p increases, EH-EDF outperforms the other policies but with a performance decrease of 15% from the first case. This is because as L_p increases, the total idle time decreases and consequently the relative performance of EH-EDF decreases. At higher values of processor load, the performance loss of EH-EDF is about 34% when compared with low processor load.

Secondly, we double the size of the capacity of the energy storage unit while keeping the other parameters unchanged (Figure 10). As previously, EH-EDF gives a percentage of feasible task sets 11%, 18%, and 24%, respectively, higher than with EH-EDF1, EH-EDF3, and EH-EDFx, respectively. When the size of the energy storage unit is doubled, the performance increases of about 21%.

As a summary, this experiment shows that it is highly probable that no online algorithm can achieve optimality. In other words, only clairvoyant algorithms that have a complete knowledge of the task properties and energy production can achieve a valid schedule whenever one exists.

7.4. Energy Storage Low Level. In this section, we measure the number of times the energy storage unit empties by varying the processor load. We consider the same values as depicted in Section 7.3. Simulations are performed for $E_{\max} = 100$ (case a) and $E_{\max} = 200$ (case b). When $E_{\max} = 100$, we observe from Figure 11 that EH-EDF presents the best behavior relative to the other policies. This is because EH-EDF will benefit from the idle time to recharge the energy storage to its maximum value while respecting all deadlines. In details, the average number of times the energy storage is

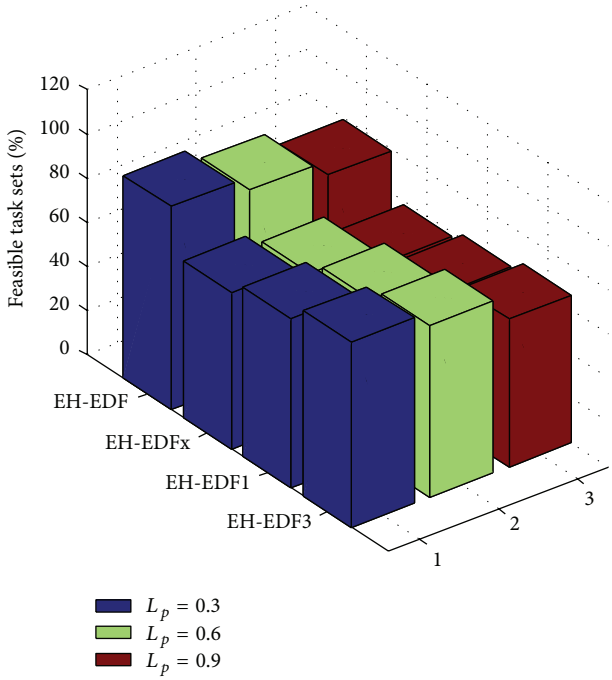

Figure 10: Percentage of feasible task sets ($E_{\max} = 200$).

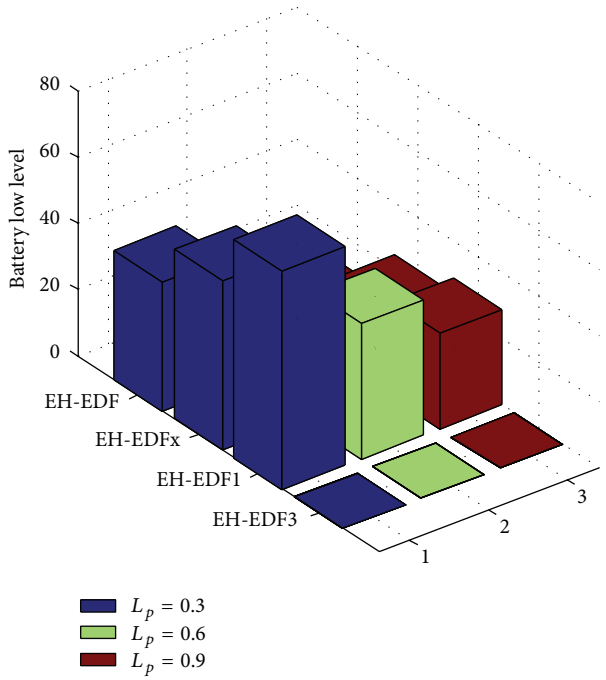

Figure 11: Battery low level ($E_{\max} = 100$).

empty under EH-EDF is, respectively, 38% and 49% less than under EH-EDFx and EH-EDF1.

Furthermore, we note that EH-EDF3 cannot reach the empty state $E_{\min}$ since the system is put in the empty state when the energy capacity reaches $E_{\text{th}\,\min}$ that is greater than $E_{\min}$ (by default equal to zero).

When the energy storage capacity is doubled, EH-EDF still has the lowest number of energy storage low level

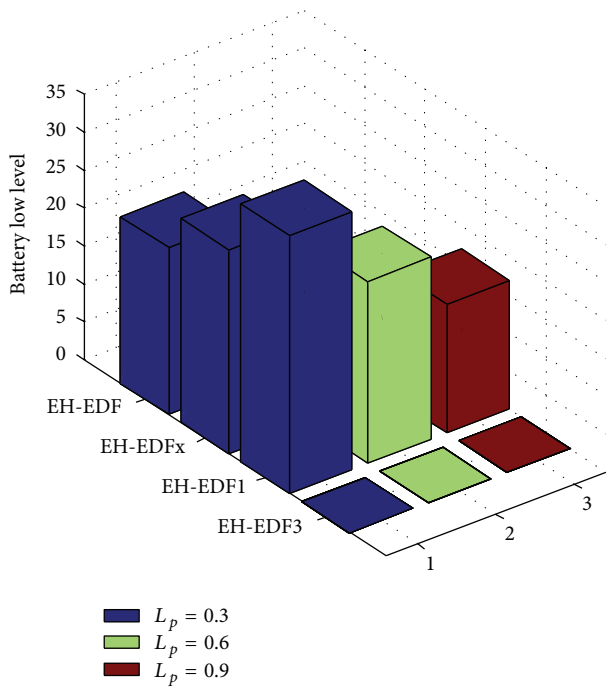

Figure 12: Battery low level ($E_{\max} = 200$).

(Figure 12). The average number of times the energy storage empties under EH-EDF is, respectively, 32% and 45% less than under EH-EDFx and EH-EDF1. As the energy storage capacity increases, the number of energy storage low levels decreases since the energy storage has a higher ability to execute tasks.

7.5. Average Idle Time. The average idle time has a great impact when studying the efficiency of EH-EDF especially in systems that use the Dynamic Power Management mechanism (DPM). DPM provides efficiency only if the idle times are sufficiently long because of inherent time and energy overhead induced by state switching. Consequently, the longer is the average idle time, the lower is the impact of the energy and time overheads incurred by DPM on the overall performance.

Moreover, the length of the idle time has a great impact on the life time of the energy storage unit regardless of its type (battery or supercapacitor). Charging any storage unit is not linear and consequently the more it is paused, the more energy it recharges.

In this section, we compute the average idle time by taking two values for $E_{\max}$. When $E_{\max} = 100$ (Figure 13), we observe that EH-EDF maximizes the average idle time, respectively, by 70%, 58%, and 45% when compared with EH-EDFx, EH-EDF3, and EH-EDF1. The reason is that, in EH-EDF, the processor is put into sleep mode as long as the slack time is positive and the energy level is less than $E_{\max}$.

When $E_{\max} = 200$, we observe from Figure 14 that EH-EDF still maximizes the average idle time, respectively, by 77%, 60%, and 49% when compared with EH-EDFx, EH-EDF3, and EH-EDF1.

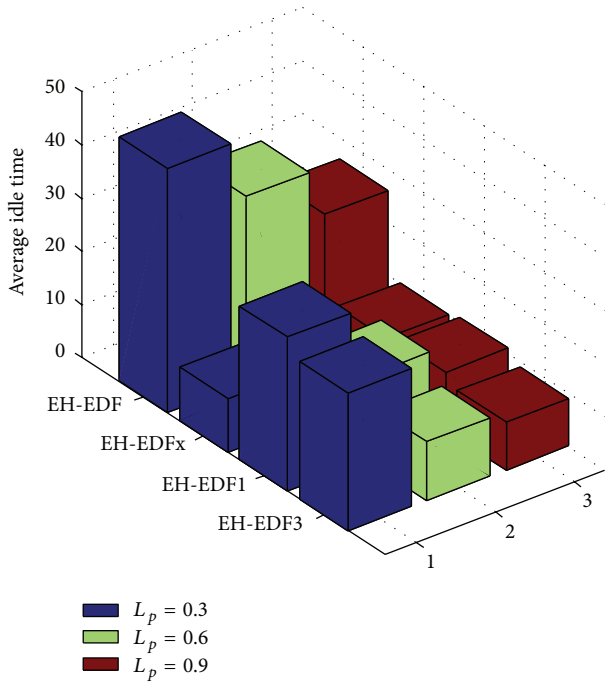

Figure 13: Average idle time ($E_{\max} = 100$).

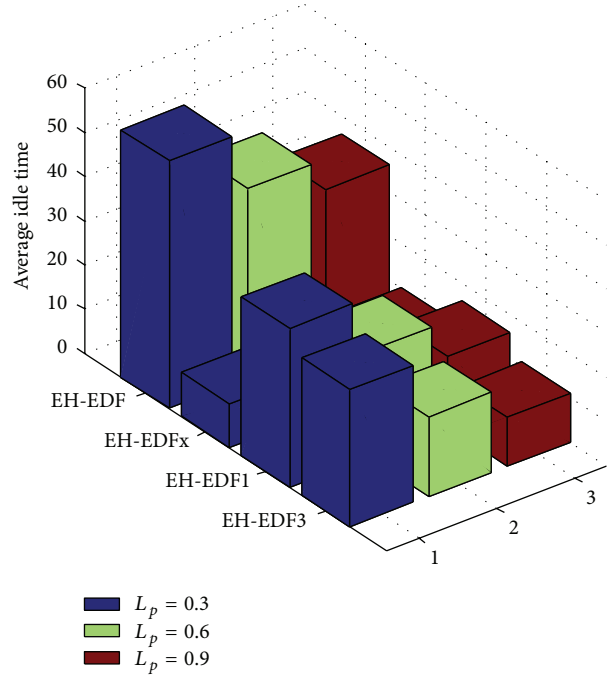

Figure 14: Average idle time ($E_{\max} = 200$).

8. Conclusions

We studied an energy harvesting sensor node which supports a set of aperiodic tasks with real-time constraints. The arrival times, deadlines, and energy demands of the tasks are not known to the node in advance. We focussed on online scheduling with no lookahead including energy production. We presented and analyzed through an experiment an idling-EDF-based scheduling algorithm called EH-EDF.

Traditional online algorithms such as EDF behave poorly because they consume the energy greedily and not adaptively. We recently proved in [11] that EDF remains the best nonidling scheduler but has a zero competitive factor for the energy harvesting model. We consequently propose several variants of EDF to derive more efficient scheduling solutions.

The experiment demonstrates that EH-EDF offers an acceptable and even good performance in a wide range of situations. We study the impact of the slack time and the threshold energy level on the performance of EH-EDF in terms of percentage of feasible task sets. We show that EH-EDF outperforms EH-EDF1, EH-EDF3, and EH-EDFx by, respectively, 7%, 10%, and 27%. Furthermore, EH-EDF proves to be better than EH-EDFx and EH-EDF1, respectively, by 38% and 49% in terms of the number of times the energy storage empties.

Finally, the advantage of the EH-EDF algorithm lies in the average duration of the processor idle times which is higher compared with other heuristics. As a result, leakage and overhead incurred by the implementation of DPM mechanism are avoided under EH-EDF.

The next step of our work will be to extend EH-EDF to the Dynamic Voltage and Frequency Scaling (DVFS) technology.

References

[1] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[2] M. L. Dertouzos, "Control robotics: the procedural control of physical processes," in *Proceedings of the International Federation for Information Processing Congress*, pp. 807–813, 1974.

[3] k. Lin, J. Yu, J. Hsu et al., "Demo abstract: heliomote: enabling long-lived sensor networks through solar energy harvesting," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, November 2005.

[4] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 463–468, April 2005.

[5] H. El Ghor, M. Chetto, and R. H. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," *Computers and Electrical Engineering*, vol. 37, no. 4, pp. 498–510, 2011.

[6] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Operating Systems Review*, vol. 35, no. 5, pp. 89–102, 2001.

[7] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.

[8] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS '06)*, pp. 313–322, December 2006.

[9] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Systems*, vol. 37, no. 3, pp. 233–260, 2007.

[10] M. Chetto, H. El Ghor, and R. Hage Chehade, "Real-time scheduling for energy harvesting sensors," in *Proceedings of the 6th International Conference on Internet Technology and Secured Transactions*, pp. 396–402, 2011.

[11] M. Chetto and A. Queudet-Marchand, "A note on EDF scheduling for real-time energy harvesting systems," *IEEE Transactions on Computers*, 2013.