Real-Time Scheduling with Regenerative Energy

C. Moser [1], D. Brunelli [2], L. Thiele [1] and L. Benini [2]
[1] Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
{moser|thiele}@tik.ee.ethz.ch
[2] University of Bologna, Italy
{dbrunelli|lbenini}@deis.unibo.it

Abstract

This paper investigates real-time scheduling in a system whose energy reservoir is replenished by an environmental power source. The execution of tasks is deemed primarily energy-driven, i.e., a task may only respect its deadline if its energy demand can be satisfied early enough. Hence, a useful scheduling policy should account for properties of the energy source, capacity of the energy storage as well as power dissipation of the single tasks. We show that conventional scheduling algorithms (like e.g. EDF) are not suitable for this scenario. Based on this motivation, we state and prove optimal scheduling algorithms that jointly handle constraints from both energy and time domain. Furthermore, an offline schedulability test for a set of periodic or even bursty tasks is presented. Finally, we validate the proposed theory by means of simulation and compare our algorithms with the classical Earliest Deadline First Algorithm.

1. Introduction

One of the key challenges of wireless sensor networks is the energy supply of the single nodes. As for many other battery-operated embedded systems, a sensor's lifetime, size and cost can be significantly improved by appropriate energy management. In particular, sensor nodes are deployed at places where manual recharging or replacement of batteries is not practical. In order for sensor networks to become a ubiquitous part of our environment, alternative power sources must be employed. Therefore, environmental energy harvesting is deemed a promising approach.

In [6], several technologies have been discussed how, e.g., solar, thermal, kinetic or vibrational energy may be extracted from a node's physical environment. Several prototypes like *Heliomote* [5] or *Prometheus* [3] could demonstrate that solar energy is a viable power source to achieve perpetual operation. However, it remains unclear how real-time responsiveness of these inherently energy constrained devices can be guaranteed. Is it possible to schedule a given set of tasks within their deadlines and – if yes – which scheduling policy guarantees schedulability? – Clearly, the answers to this questions depend on parameters like the capacity of the battery and the impinging light intensity. Unfortunately, the latter is highly unstable and complicates the search for online scheduling algorithms as well as admittance tests.

The Earliest Deadline First (EDF) algorithm has been proven to be optimum with respect to the schedulability of a given taskset in traditional time-driven scheduling. The following example shows why greedy scheduling algorithms (like EDF) are not suitable in the context of this paper.

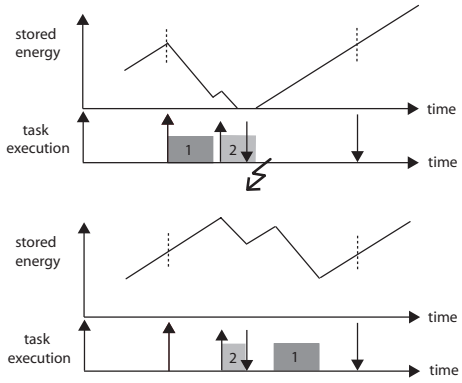

Figure 1. Greedy vs. lazy scheduling

Let us consider a node with an energy harvesting unit that replenishes a battery with constant power. Now, this node has to perform an arriving task "1" that has to be finished until a certain deadline. Meanwhile, a second task "2" has to be executed within a shorter time interval that is again given by an arrival time and a deadline. In Figure 1, the arrival times and deadlines of these tasks are indicated by up and down arrows respectively. As depicted in the top diagrams, a greedy scheduling strategy violates the deadline

Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)
0-7695-2619-5 /06 $20.00 © 2006 IEEE

of task "2" since it dispenses overhasty the stored energy by driving task "1". When the energy is required to execute the second task, the battery level is not sufficient to meet the deadline. In this example, however, a scheduling strategy that hesitates to spend energy on task "1" meets both deadlines. The bottom plots illustrate how the *Lazy Scheduling* paradigm described in this paper outperforms a naive, greedy approach like EDF in the described situation.

The research presented in this paper is directed towards sensor nodes. But in general, our results apply for all kind of energy harvesting systems which must schedule processes under deadline constraints: We claim, that our scheduling algorithms are not only energy-aware, but truly energy-*driven*. This insight originates from the fact, that energy – contrary to the computation resource "time" – is storable. As a consequence, every time we withdraw energy from the battery to execute a task, we change the state of our scheduling system. That is, after having scheduled a first task the next task will encounter a lower energy level in the system which in turn will affect its own execution. This is not the case in conventional real-time scheduling where time just elapses either used or unused.

The following new results are described in this paper:

- We present an energy-driven scheduling scenario for a system whose energy storage is recharged by an environmental source.
- For this scenario, we state and prove optimal online algorithms that dynamically assign energy to arriving tasks. These algorithms are energy-clairvoyant, i.e., scheduling decisions are driven by the future incoming energy.
- We present an admittance test that decides, whether a set of tasks can be scheduled with the energy produced by the harvesting unit. For this purpose, we introduce the concept of energy variability characterization curves (EVCC).
- Finally, we propose practical algorithms which predict the future harvested energy with the help of EVCCs. Simulation results demonstrate the benefits of our algorithms compared to the classical Earliest Deadline First Algorithm.

The remainder of the paper is organized as follows: In the next section, we discuss how our approach differs from related work. Subsequently, Section 3 gives definitions that are essential for the understanding of the paper. In Section 4, we present *Lazy Scheduling Algorithms* for optimal online scheduling. Admittance tests for arbitrary tasksets are the topic of Section 5. Simulation Results are presented in Section 6 and Section 7 concludes the paper.

2. Related work

In [4], the authors use a similar model of the power source as we do. But instead of executing concrete tasks in a real-time fashion, they propose tuning a node's duty cycle dependent on the parameters of the power source. Nodes switch between active and sleep mode and try to achieve sustainable operation. This approach only indirectly addresses real-time responsiveness: It determines the latency resulting from the sleep duration.

The approach in [7] is restricted to a very special offline scheduling problem: Periodic tasks with certain rewards are scheduled within their deadlines according to a given energy budget. The overall goal is to maximize the sum of rewards. Therefore, energy savings are achieved using Dynamic Voltage Scaling (DVS). The energy source is assumed to be solar and comprises two simple states: day and night. Hence the authors conclude that the capacity of the battery must be at least equal to the cumulated energy of those tasks, that have to be executed at night. In contrast, our work deals with a much more detailed model of the energy source. We focus on scheduling decisions for the online case when the scheduler is *indeed* energy-constraint. In doing so, we derive valuable bounds on the necessary battery size for arbitrary energy sources and tasksets.

The research presented in [1] is dedicated to offline algorithms for scheduling a set of periodic tasks with a common deadline. Within this so-called "frames", the order of task execution is *not* crucial for whether the taskset is schedulable or not. The power scavenged by the energy source is assumed to be constant. Again – by using DVS – the energy consumption is minimized while still respecting deadlines. Contrary to this work, our systems (e.g. sensor nodes) are predominantly energy constrained and the energy demand of the tasks is fixed (no DVS). We propose algorithms that make best use of the available energy. Provided that the average harvested power is sufficient for continuous operation, our algorithms minimize the necessary battery capacity.

3. System model and assumptions

The paper deals with a scheduling scenario depicted in Fig. 2. At some time t, an energy source harvests ambient energy and converts it into electrical power $P_S(t)$. This power can be stored in a device with capacity C. The stored energy is denoted as $E_C < C$. On the other hand, a computing device drains power $P_D(t)$ from the storage and uses it to process tasks with arrival time a_i, energy demand e_i and deadline d_i. We assume that only one task is executed at time t and preemptions are allowed. The following subsections define the relations between these quantities in more detail.

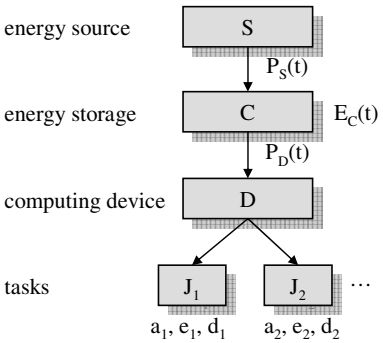

Figure 2. Scheduling scenario

3.1. Energy source

Many environmental power sources are highly variable with time. Hence, in many cases some charging circuitry is necessary to optimize the charging process and increase the lifetime of the storage device. In our model, the power P_S incorporates all losses caused by power conversion as well as charging process. In other words, we denote $P_S(t)$ the charging power that is actually fed into the energy storage. The respective energy E_S in the time interval $[t_1, t_2]$ is given as

$$E_S(t_1, t_2) = \int_{t_1}^{t_2} P_S(t)\mathrm{d}t \quad .$$

In order to characterize the properties of an energy source, we define now energy variability characterization curves (EVCC) that bound the energy harvested in a certain interval Δ: The EVCCs $\epsilon^l(\Delta)$ and $\epsilon^u(\Delta)$with $\Delta \geq 0$ bound the range of possible energy values E_S as follows:

$$\epsilon^l(t_2 - t_1) \leq E_S(t_1, t_2) \leq \epsilon^u(t_2 - t_1) \quad \forall t_2 > t_1$$

Given an energy source, e.g., a solar cell mounted in a building or outside, the EVCCs provide guarantees on the produced energy. For example, the lower curve denotes that for any time interval of length Δ, the produced energy is at least $\epsilon^l(\Delta)$ (see Fig. 3). Two possible ways of deriving an EVCC for a given scenario are given below:

- A sliding window of length Δ is used to find the minimum/maximum energy produced by the energy source in any time interval $[t_1, t_2)$ with $t_2 - t_1 = \Delta$. To this end, one may use a long power trace or a set of traces that have been measured. Since the resulting EVCC bounds only the underlying traces, these traces must be selected carefully and have to be representative for the assumed scenario.
- The energy source and its environment is formally modeled and the resulting EVCC is computed.

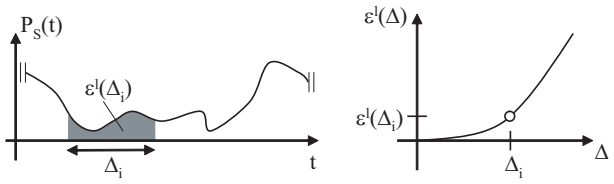

Figure 3. Power trace and energy variability characterization curve (EVCC)

3.2. Energy storage

We assume an ideal energy storage (e.g. a battery) that may be charged up to its capacity C. According to the scheduling policy used, power $P_D(t)$ and the respective energy $E_D(t_1, t_2)$ is drained from the storage to execute tasks. If no tasks are executed and the storage is consecutively replenished by the energy source, an energy overflow occurs. Consequently, we can derive the following relations:

$$\begin{aligned} E_C(t) &\leq C && \forall t \\ E_D(t_1, t_2) &\leq E_C(t_1) + E_S(t_1, t_2) && \forall t_2 > t_1 \\ E_C(t_2) &\leq E_C(t_1) + E_S(t_1, t_2) - E_D(t_1, t_2) && \forall t_2 > t_1 \end{aligned}$$

3.3. Task scheduling

As illustrated in Fig. 2, we utilise the notion of a computing device that assigns energy E_C from the storage to dynamically arriving tasks. Here, we limit the power consumption $P_D(t)$ of the system to the maximum value P_{max}. In other words, the processing device determines at any point in time how much power it uses, that is

$$0 < P_D(t) < P_{max} \, .$$

We assume tasks to be independent from each other and preemptive. That is, the currently active task may be interrupted at any time and continued at a later time. If the node decides to assign power $P_i(t)$ to the execution of task J_i during the interval $[t_1, t_2]$, we denote the corresponding energy $E_i(t_1, t_2)$. The effective starting time s_i and finishing time f_i of task i are dependent on the scheduling strategy used: A task starting at time s_i will finish as soon as the required amount of energy e_i has been consumed by it. We can write

$$f_i = \min \{t : E_i(s_i, t) = e_i\} \, .$$

We see that the actual running time $(f_i - s_i)$ of a task i is strongly dependent on the amount of power $P_i(t)$ which is driving the task during $s_i \leq t \leq f_i$. In the best case, a task may finish after the execution time $w_i = \frac{e_i}{P_{max}}$ if it is processed without interrupts and with the maximum power P_{max}.

4. Lazy Scheduling Algorithms LSA

After having described our modeling assumptions, we will now state and prove optimal scheduling algorithms. In subsection 4.1 and 4.2, we will introduce two scheduling algorithms that dynamically assign energy and time to arriving tasks. Theorems which prove optimality of both algorithms will follow in subsection 4.3.

4.1. LSA-I for unlimited power P_{max}

We start with a hypothetical system with infinite power $P_{max} = +\infty$. As a result, a task's execution time w_i collapses to 0 if the available energy E_C in the storage is equal to or greater than the task's energy demand e_i. This fact clearly simplifies the search for advantageous scheduling algorithms but at the same time contributes to the understanding of our problem. Furthermore, it should be mentioned that a theoretical node which runs a task in zero time can be a good approximation for many practical scenarios. Whenever processing times w_i are negligible compared to the time to recharge the battery (i.e. $P_{max} \gg P_S(t)$), the assumed model can be regarded as reasonable.

For optimal task scheduling, the processing device has to select between three power modes. The node may process tasks with the maximal power $P_D(t) = P_{max}$ or not at all ($P_D(t) = 0$). In between, the node may choose to spend only the currently incoming power $P_S(t)$ from the harvesting unit on tasks. Altogether, we consider a node that decides between $P_D(t) = P_S(t)$, $P_D(t) = 0$ and $P_D(t) = +\infty$ to schedule arriving tasks.

As already indicated in the introduction, the naive approach of simply scheduling tasks with the EDF algorithm may result in unnecessary deadline violations (cp. Fig. 1). It may happen, that after the execution of task "1" another task "2" with an earlier deadline arrives. If now the required energy is not available before the deadline of task "2", EDF scheduling produces a deadline violation. If task "1" would wait instead of executing directly, this deadline violation might have been avoidable. These considerations directly lead us to the principle of *Lazy Scheduling*: Gather environmental energy and process tasks only if it is necessary.

The *Lazy Scheduling Algorithm* (LSA-I) for $P_{max} = \infty$ shown below attempts to assign energy to a set of tasks J_i, $i \in Q$ such that deadlines are respected. It is based on the two following rules:

- **Rule 1**: If the current time t equals the deadline d_j of some arrived but not yet finished task J_j, then finish its execution by draining energy $(e_j - E_j(a_j, t))$ from the energy storage instantaneously, i.e. with $P_D(d_j) = +\infty$.
- **Rule 2**: We must not waste energy if we could spend it on task execution. Therefore, if we hit the capacity limit ($E_C(t) = C$) at some times t, we execute the task with the earliest deadline using $P_D(t) = P_S(t)$.
- **Rule 3**: Rule 1 overrules Rule 2.

Lazy Scheduling Algorithm LSA 1 ($P_{max} = \infty$)

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i
$P_D(t) \Leftarrow 0$;
while (true) **do**
  $d_j \Leftarrow \min\{d_i : i \in Q\}$;
  process task J_j with power $P_D(t)$;
  $t \Leftarrow$ current time;
  if $t = a_k$ **then** add index k to Q;
  if $t = f_j$ **then** remove index j from Q;
  if $t = d_j$ **then** $E_C(t) \Leftarrow E_C(t) - e_j + E_j(a_j, t)$;
    remove index j from Q;
    $P_D(t) \Leftarrow 0$;
  if $E_C(t) = C$ **then** $P_D(t) \Leftarrow P_S(t)$;

The above algorithm is solely designed for schedulable tasksets and doesn't account for possible deadline violations. If there is not enough energy available at the deadline of task ($E_C(d_j) < e_j - E_j(a_j, d_j)$) then the taskset is assumed to be not schedulable.

Note that the LSA degenerates to an *earliest deadline first* (EDF) policy, if $C = 0$. On the other hand, we find an *as late as possible* (ALAP) policy for the case of $C = +\infty$.

4.2. LSA-II for limited power P_{max}

Now, we focus on how LSA-I must be adapted to handle limited power consumption and finite execution times. In doing so, we determine an optimal starting time s that balances the time *and* energy constraints for our scheduling scenario.

The LSA-I algorithm instantaneously executes a task at its deadline. However, after the introduction of a finite, minimal computation time $w_i = \frac{e_i}{P_{max}}$, LSA-II has to determine an earlier starting time in order to hold the respective deadline. The upper plots in Fig. 4 display a straightforward ALAP-translation of the starting time for task "1": To fulfill its time condition, task "1" begins to execute at starting time $s_1 = d_1 - \frac{e_1}{P_{max}}$. As illustrated, it may happen that shortly after s_1 an unexpected task "2" arrives. Assume that this unexpected task "2" is nested in task "1", i.e., it also has an earlier deadline than "1". This scenario inevitably leads to a deadline violation, although plenty of energy is available. This kind of timing conflict can be solved by shifting s_1 to earlier times and thereby reserving time for the unpredictable task "2" (see lower plots Fig. 4). But starting earlier, we risk to "steal" energy that might be needed at

later times (compare Fig. 1). So – according to the "lazy" principle – we have to take care that we don't start *too* early.

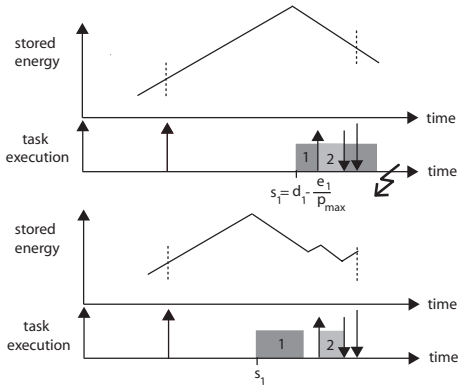

Figure 4. ALAP vs. lazy scheduling

From the above example, we learned that it may be disadvantageous to prearrange a starting time in such a way, that the stored energy E_C cannot be used before the deadline of a task. If the processing device starts running at time s with P_{max} and cannot consume all the available energy before the deadline d, time conflicts may occur. On the other hand, if we remember the introductory example in Fig. 1, energy conflicts are possible if the stored energy $E_C(t) = 0$ at some time $t < d$. Hence we can conclude the following: The scheduler has to conserve the energy E_C as long as required, but must start processing the stored energy duly. Consequently, the optimal starting time s must guarantee, that the processor *could* continuously use P_{max} in the interval $[s, d]$ and empty the energy storage at time d.

A necessary prerequisite for the calculation of the optimal starting time s_i is the knowledge of the incoming power flow $P_S(t)$ for all future times $t \leq d_i$. Finding useful predictions for the power $P_S(t)$ can be done, e.g., by analyzing traces of the harvested power, as we will see in Section 6. In addition, we assume that

$$P_S(t) < P_{max} \quad \forall t\,,$$

that is, the incoming power $P_S(t)$ from the harvesting unit never exceeds the power consumption P_{max} of a busy node. Besides from being a realistic model in many cases, this assumption prevents us from dealing with possible energy losses.

To calculate the optimal starting time s_i, we determine the maximum amount of energy $E_C(a_i) + E_S(a_i, d_i)$ that may be processed by the node before d_i. Next, we compute the minimum time required to process this energy without interruption and shift this time interval of continuous processing just before the deadline d_i. In other words, we calculate the starting time s_i^* as

$$s_i^* = d_i - \frac{E_C(a_i) + E_S(a_i, d_i)}{P_{max}}.$$

If now the energy storage was filled before s_i^*, starting execution at s_i^* could yield $E_C(t) = 0$ at some time $t < d_i$. Thus starting at s_i^* means starting too early and one can find a better starting time s_i' by solving the following equation numerically:

$$E_S(a_i, s_i') - C = E_S(a_i, d_i) + (s_i' - d_i)P_{max}.$$

We denote the optimal starting time

$$s_i = \max(s_i', s_i^*).$$

The pseudo-code of the Lazy Scheduling Algorithm LSA-II is shown below. It is based on the following rules:

- **Rule 1**: EDF scheduling is used at time t for assigning the processor to all waiting tasks with $s_i \leq t$. The currently running task is powered with $P_D(t) = P_{max}$.
- **Rule 2**: If there is no waiting task i with $s_i \leq t$ and if $E_C(t) = C$, then all incoming power P_S is used to process the task with the smallest deadline($P_D(t) = P_S(t)$). If there is no waiting task at all, the incoming power is wasted.

Lazy Scheduling Algorithm LSA 2 (P_{max} = const.)

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i
$P_D(t) \Leftarrow 0$;
while (true) **do**
  $d_j \Leftarrow \min\{d_i : i \in Q\}$;
  calculate s_j;
  process task J_j with power $P_D(t)$;
  $t \Leftarrow$ current time;
  if $t = a_k$ **then** add index k to Q;
  if $t = f_j$ **then** remove index j from Q;
  if $E_C(t) = C$ **then** $P_D(t) \Leftarrow P_S(t)$;
  if $t \geq s_j$ **then** $P_D(t) \Leftarrow P_{max}$;

The calculation of s_i must be performed once the scheduler selects the task with the earliest deadline. If the scheduler is not energy-constraint, i.e., if the available energy is more than the device can consume with power P_{max} within $[a_i, d_i]$, the starting time s_i will be before the current time t. Then, the resulting scheduling policy is EDF, which is reasonable, because only time constraints have to be satisfied.

In summary, LSA-II can be classified as an energy-clairvoyant adaptation of the Earliest Deadline First Algorithm. It changes its behaviour according to the amount of

available energy, the capacity C as well as the maximum power consumption P_{max} of the device. For example, the lower the power P_{max} gets, the greedier LSA-II gets. On the other hand, high values of P_{max} force LSA-II to hesitate and postpone the starting time s. For $P_{max} = \infty$, all starting times collapse to the respective deadlines, and we identify LSA-I as a special case of LSA-II.

4.3. Optimality proof for LSA I+II

In this section, we will show that the LSA-II algorithm is optimal in the following sense: If the algorithm can not schedule a given set of tasks no other algorithm is able to do so. Since LSA-I is just a special case of LSA-II, the considerations in this section are restricted to the case of LSA-II.

The scheduling scenario presented in this paper is inherently energy-driven. Hence, a scheduling algorithm yields a deadline violation if it fails to assign the energy e_i to a task before its deadline d_i. We distinguish between two types of deadline violations:

- A deadline cannot be respected since the time is not sufficient to execute available energy with power P_{max}. At the deadline, unprocessed energy remains in the storage and we have $E_C(d) > 0$. We call this the **time limited** case.
- A deadline violation occurs because the required energy is simply not available at the deadline. At the deadline, the battery is exhausted (i.e., $E_C(d) = 0$). We denote the latter case **energy limited**.

For the following theorems to hold we suppose that at initialization of the system, we have a full capacity, i.e., $E_C(t_i) = C$. Furthermore, we call the computing device *idle* if no task i is running with $s_i \leq t$.

Theorem 1 *Let us suppose that the LSA-II algorithm schedules a set of tasks. At time d the deadline of a task J with arrival time a is missed and $E_C(d) > 0$. Then there exists a time t_1 such that the sum of execution times $\sum_{(i)} w_i = \sum_{(i)} \frac{e_i}{P_{max}}$ of tasks with arrival **and** deadline within time interval $[t_1, d]$ exceeds $d - t_1$.*

Proof 1 *Let us suppose that t_0 is the maximal time $t_0 \leq d$ where the processor was idle. Clearly, such a time exists.*

We now show, that at t_0 there is no task i with deadline $d_i \leq d$ waiting. At first, note that the processor is constantly operating on tasks in time interval $(t_0, d]$. Suppose now that there are such tasks waiting and task i is actually the one with the earliest deadline d_i among those. Then, as $E_C(d) > 0$ and because of the construction of s_i, we would have $s_i < t_0$. Therefore, the processor would actually process task i at time t_0 which is a contradiction to the idleness.

Because of the same argument, all tasks i arriving after t_0 with $d_i \leq d$ will have $s_i \leq a_i$. Therefore, LSA-II will attempt to directly execute them using an EDF strategy.

*Now let us determine time $t_1 \geq t_0$ which is the largest time $t_1 \leq d$ such that the processor continuously operates on tasks i with $d_i \leq d$. As we have $s_i \leq a_i$ for all of these tasks and as the processor operates on tasks with smaller deadlines first (EDF), it operates in $[t_1, d]$ only on tasks with $a_i \geq t_1$ and $d_i \leq d$. As there is a deadline violation at time d, we can conclude that $\sum_{(i)} w_i > d - t_1$ where the sum is taken over all tasks with arrival **and** deadline within time interval $[t_1, d]$.*

Theorem 2 *Let us suppose that the LSA-II algorithm schedules a set of tasks. At time d the deadline of a task J with arrival time a is missed and $E_C(d) = 0$. Then there exists a time t_1 such that the sum of task energies $\sum_{(i)} e_i$ of tasks with arrival **and** deadline within time interval $[t_1, d]$ exceeds $C + E_S(t_1, d)$.*

Proof 2 *Let time $t_1 \leq d$ be the largest time such that (a) $E_C(t_1) = C$ and (b) there is no task i waiting with $d_i \leq d$. Such a time exists as one could at least use the initialization time t_i with $E_C(t_i) = C$. As t_1 is the last time instance with the above properties, we can conclude that everywhere in time interval $[t_1, d]$ we either have (a) $E_C(t) = C$ and there is some task i waiting with $d_i \leq d$ or we have (b) and $E_C(t) < C$.*

It will now be shown that in both cases a) and b), energy is not used to advance any task j with $d_j > d$ in time interval $[t_1, d]$. Note also, that all arriving energy $E_S(t_1, d)$ is used to advance tasks.

In case a), all non-storable energy (i.e. all energy that arrives from the source) is used to advance a waiting task, i.e., the one with the earliest deadline $d_i \leq d$. In case b), the processor would operate on task J with $d_j > d$ if there is some time $t_2 \in [t_1, d]$ where there is no other task i with $d_i \leq d$ waiting and $s_j \leq t_2$. But s_j is calculated such that the processor could continuously work until d_j. As $d_j > d$ and $E_C(d) = 0$ this can not happen and $s_j > t_2$. Therefore, also in case b) energy is not used to advance any task j with $d_j > d$.

*As there is a deadline violation at time d, we can conclude that $\sum_{(i)} e_i > C + E_C(t_1, d)$ where the sum is taken over all tasks with arrival **and** deadline within time interval $[t_1, d]$.*

From the above two theorems we draw the following conclusions: First, in the **time limited** case, there exists a time interval before the violated deadline with a larger accumulated computing time request than available time. And second, in the **energy limited** case, there exists a time interval before the violated deadline with a larger accumulated energy request than what can be provided at best. Therefore, we conclude the following:

Corollary 1 (Optimality of Lazy Scheduling) *If LSA-II cannot schedule a given taskset, then no other scheduling algorithm can. This holds even if the other algorithm knows the complete taskset in advance.*

If we can guarantee that there is no time interval with a larger accumulated computing time request than available time **and** no time interval with a larger accumulated energy request than what can be provided at best, then the taskset is schedulable. This property will be used to determine the admittance test described next.

5. Admittance test

In this section, we will determine an offline schedulability test in case of periodic, sporadic or even bursty tasksets. In particular, given an energy source with lower EVCC $\epsilon^l(\Delta)$, an energy storage with capacity C and a set of periodic tasks J_i, $i \in I$ with period p_i, *relative* deadline d_i and energy demand e_i, we would like to determine whether all deadlines can be respected.

To this end, let us first define for each task its arrival curve $\alpha(\Delta)$ which denotes the maximal number of task arrivals in any time interval of length Δ. The notion of arrival curves to describe the arrival patterns of tasksets is well known and has been used explicitly or implicitly in, e.g., [2], or [8]. To simplify the discussion, we limit ourselves to periodic tasks, but the whole formulation allows to deal with much more general classes (sporadic or bursty) as well.

In case of a periodic taskset, we have for periodic task J_i, see also Fig. 5:

$$\alpha_i(\Delta) = \left\lceil \frac{\Delta}{p_i} \right\rceil \quad \forall \Delta \geq 0$$

In order to determine the maximal energy demand in any time interval of length Δ, we need to maximize the accumulated energy of all tasks having their arrival and deadline within an interval of length Δ. To this end, we need to shift the corresponding arrival curve by the relative deadline. In case of a periodic task J_i, this simply leads to:

$$\overline{\alpha}_i(\Delta) = \begin{cases} e_i \cdot \alpha_i(\Delta - d_i) & \Delta > d_i \\ 0 & 0 \leq \Delta \leq d_i \end{cases}$$

In case of several periodic tasks that arrive concurrently, the total demand curve $A(\Delta)$ can be determined by just adding the individual contributions of each periodic task, see Fig. 5:

$$A(\Delta) = \sum_{i \in I} \overline{\alpha}_i(\Delta)$$

Using the above defined quantities, we can formulate one of the major results of the paper which determines the schedulability of an arbitrary taskset:

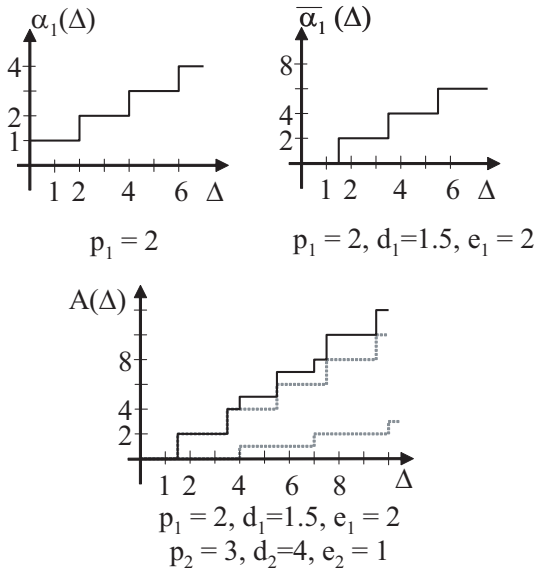

Figure 5. Examples of an arrival curve, a demand curve and a total demand curve in case of periodic tasks.

Theorem 3 (Schedulability Test) *A given set of tasks J_i, $i \in I$ with arrival curves $\alpha_i(\Delta)$, energy demand e_i and relative deadline d_i is schedulable under the energy-driven model with initially stored energy C, if and only if the following condition holds*

$$A(\Delta) \leq \min\left(\epsilon^l(\Delta) + C\,,\, P_{max} \cdot \Delta\right) \quad \forall \Delta > 0$$

Here, $A(\Delta) = \sum_{i \in I} e_i \cdot \alpha_i(\Delta - d_i)$ denotes the total energy demand of the taskset in any time interval of length Δ, $\epsilon^l(\Delta)$ the energy variability characterization curve of the energy source, C the capacity of the energy storage and P_{max} the maximal processing power of the system. In case of periodic tasks we have $A(\Delta) = \sum_{i \in I} e_i \cdot \left\lceil \frac{\Delta - d_i}{p_i} \right\rceil$.

Proof 3 *The proof of the if direction is omitted, since it is a direct consequence of Theorems 1 and 2. We just prove the only-if direction.*

Remember that the total demand curve $A(\Delta)$ denotes the maximal energy demand of tasks in any interval $[t_1, t_2]$ of length Δ. It equals the maximal accumulated energy of tasks having their arrival **and** *deadline within $[t_1, t_2]$. Therefore, in order to satisfy all deadlines for these tasks, at least energy $A(t_2 - t_1)$ must be available.*

Let us suppose that the condition in Theorem 3 is violated for some Δ due to missing energy. Let us suppose also that the task arrival curve and the energy variability characterization curve are strict, i.e., there exists some time interval $[t_1, t_2]$ where the energy demand is $A(t_2 - t_1)$ and

at the same time the energy $E_S(t_2 - t_1)$ is received. Then in time interval $[t_1, t_2]$ with $\Delta = t_2 - t_1$ the difference between the energy demand and the received energy is larger than the maximal stored energy C as $A(\Delta) > \epsilon^l(\Delta) + C$. As a result, the taskset is not schedulable.

On the other hand, whenever the demanded computation time $\frac{A(\Delta)}{P_{max}}$ of a taskset in the interval Δ is larger than the interval itself, a taskset is not schedulable. Therefore it is evident, that both the energy condition $A(\Delta) \leq \epsilon^l(\Delta) + C$ and the time condition $A(\Delta) \leq P_{max} \cdot \Delta$ must be fulfilled in order to avoid deadline violations.

Theorem 3 tells us that we can decouple energy and time constraints if we have to decide whether a taskset is schedulable or not. On the one hand, a taskset has to respect the time constraint imposed by P_{max}. Only if

$$P_{max} \geq \max_{0 \leq \Delta} \left(\frac{A(\Delta)}{\Delta} \right) ,$$

the system is fast enough to process the given taskset. This condition is independent of the energy provided by the environmental source (i.e. ϵ^l) and the capacity of the storage device. Even increasing the capacity does not help. If a taskset however satisfies the time constraint, the role of the capacity C as a design parameter for the energy harvesting system becomes important.

Suppose now that the time constraint is fulfilled. For this case, Theorem 3 claims that the minimum capacity C_{min} needed to schedule a taskset with $A(\Delta)$ using a source with $\epsilon^l(\Delta)$ is given by

$$C_{min} = \max_{0 \leq \Delta} \left(0, A(\Delta) - \epsilon^l(\Delta) \right) .$$

From Corollary 1 we derive, that LSA scheduling guarantees schedulability with C_{min}. Hence, among all algorithms, LSA is the one with requires the minimum capacity C_{min} to successfully schedule a given taskset.

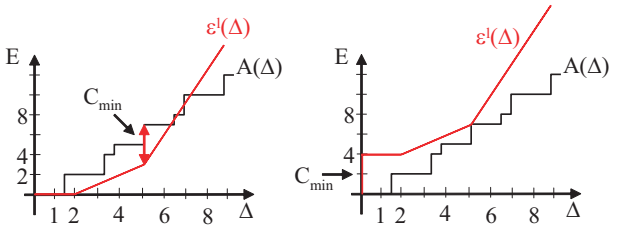

Figure 6. Determining C_{min} for the schedulability test in theorem 3.

Fig. 6 illustrates an example for such a schedulability test. The left diagram displays the total demand curve $A(\Delta)$ for two periodic tasks with p_1=2, d_1=1, e_1=2 and p_2=3, d_2=4, e_2=1. Furthermore, the EVCC $\epsilon^l(\Delta)$ is given by a piecewise linear function using three pieces (0,0,0),(2,0,1),(5,3,3), where each piece i is defined by the triple of the form (initial Δ_i, initial $\epsilon^l(\Delta_i)$, slope of piece i). Now, one can calculate that the maximal difference between the total demand curve A and the EVCC ϵ^l has value 4 which is obtained at $\Delta = 5$. Therefore, one can conclude that the set of tasks with associated deadlines can be scheduled by LSA using a minimal capacity $C_{min} = 4$. The respective schedulability test with C_{min} is shown in the right diagram of Fig. 6.

Regarding the slope of the curves in Fig. 6, we can guess that A and ϵ^l won't intersect after the critical time interval of length 5. Formally, this is because the minimum average power $\lim_{\Delta \to \infty} \frac{\epsilon^l(\Delta)}{\Delta}$ is higher than the maximum average power demand $\lim_{\Delta \to \infty} \frac{A(\Delta)}{\Delta}$ of the taskset. In other words, for large values of the interval Δ the incoming energy is always greater than the energy needed for task execution. This is precisely the condition for perpetual operation, which is included in our admittance test, too. If, however

$$\lim_{\Delta \to \infty} \frac{\epsilon^l(\Delta)}{\Delta} < \lim_{\Delta \to \infty} \frac{A(\Delta)}{\Delta} ,$$

only a hypothetical capacity of $C_{min} = +\infty$ guarantees schedulability.

6. Simulation results

6.1. Simulation setup

The trace of the power source P_S is generated by a random number generator (see Fig. 7). From this trace we compute the average power $\overline{P_S}$ as well as upper and lower EVCCs ϵ^u and ϵ^l.

A taskset consists of an arbitrary number of periodic tasks. Here, periods p are taken from a set {10, 20, 30, ... , 100}, each value having an equal probability of being selected. The initial phases φ are uniformly distributed between [0,100]. For simplicity, the relative deadline d is equal to the period p of the task. The energies e of the periodic tasks are generated according to a uniform distribution in $[0, e_{max}]$, with $e_{max} = \overline{P_S} \cdot p$. All simulations have been run for a total simulation time T and repeated for N tasksets.

We define the utilisation $U \in [0, 1]$ of a scheduler as

$$U = \sum_i \frac{\frac{e_i}{\overline{P_S}}}{p_i} .$$

One can interpret U as the percentage of processing time of the device if tasks are solely executed with the average incoming power $\overline{P_S}$. A system with, e.g., $U > 1$ is processing

more energy than it scavenges on average and will deplete its energy reservoir.

6.2. Admittance test

At first, we are interested in the tightness of the capacity bound C_{min}. To avoid timing conflicts, we set the maximum power $P_{max} = \infty$.

Fig. 7 depicts the power P_S we used for the first 5000 time units. Based on this curve, $N = 5000$ tasksets are generated which yield a processor utilisation U between 0.1 and 0.9. For that purpose, the number of periodic tasks in each taskset is successively incremented until the intended utilisation U is reached. For each taskset, the value of C_{min} is computed with the help of ϵ^l and the demand curve A of the respective taskset (cp. Fig. 6). Next, we start simulations with varying capacities C between $0.8C_{min}$ and $1.2C_{min}$. Initially, the energy storage is full and we count the number of tasksets that survive a simulation period $T = 20000$ without deadline violations.

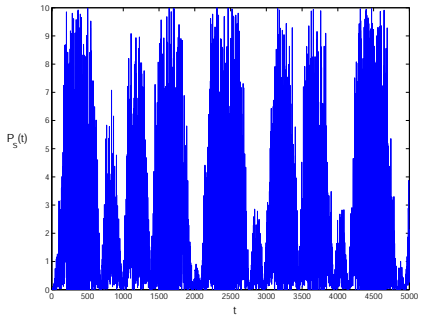

Figure 7. Power trace

Fig. 8 illustrates the percentage of tasks without deadline violation over the capacity C. Obviously, for capacities equal or greater than C_{min}, all tasksets turn out to be schedulable. In the simulated time, however, not all tasksets with $C < C_{min}$ produced deadline violations. Therefore we increased the simulation time T from 20000 to 100000 time units for single values of C. At, e.g., $C = 0.95C_{min}$ we measured only 9% passed tasks at $T = 100000$ instead of 20% at $T = 20000$.

6.3. Approximation methods for P_S

For a realistic scenario, the knowledge of the future energy E_S is not a very practical assumption. Therefore, we implemented two LSA algorithms that calculate starting times s_i based on the EVCCs ϵ^l and ϵ^u, respectively. That is, we assume an offline characterization of the harvested energy. Taking into account the maximum power

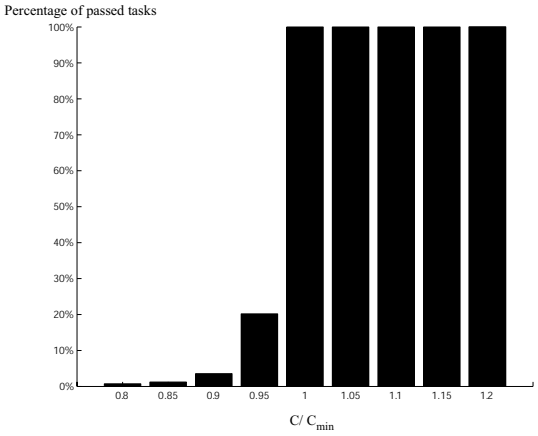

Figure 8. Tightness of C_{min}

$P_{max} = 10$ of the device, scheduling decisions remain online for an unknown taskset. The EDF algorithm – which is optimal for time-driven scheduling – serves as benchmark for this simulation.

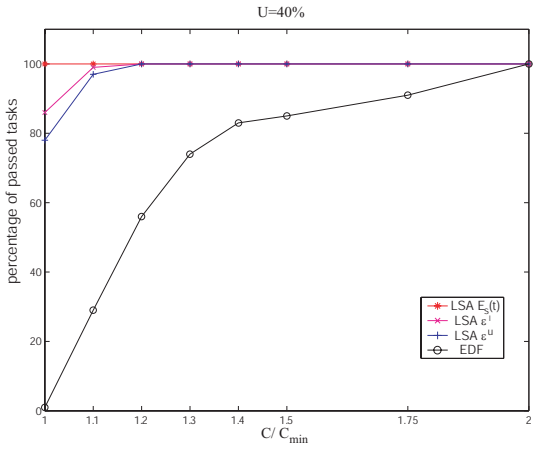

Figure 9. Comparison of EDF with LSA (with $P_S(t)$, ϵ^l and ϵ^u) at $U = 40\%$

The performance of the four algorithms is very similar for different values of U and we decided to show only the result for $U = 40 \pm 1\%$. Fig. 9 shows the percentage of $N = 5000$ tasksets that could be scheduled without deadline violation for $T = 10000$. Again, we calculated C_{min} for every taskset to be able to show the average behaviour in on plot. Clearly, no deadline violations occur for energy-clairvoyant LSA scheduling and values of $\frac{C}{C_{min}} \geq 1$. Both approximations of LSA with the EVCCs outperform the EDF algorithm, whereat the lower curve ϵ^l seems to be the better approximation. For a capacity of $C = C_{min}$ almost no taskset is schedulable with EDF. Here, LSA with ϵ^u is

able to schedule $\approx 78\%$ of all tasksets, LSA with ϵ^l even $\approx 85\%$. The capacity needed for EDF to avoid deadline violations ($2 \cdot C_{min}$) is even 67% higher than the one needed with the combination LSA and ϵ^u ($1.2 \cdot C_{min}$).

From a theoretical point of view, LSA with ϵ^u can never violate a deadline if EDF doesn't. Hence, the curve for LSA with ϵ^u is above the EDF curve as expected.

6.4. Improvements in stored energy

Finally, we are interested in the average energy stored in the system during a simulation period of $T = 10000$. For a fair comparison of the four algorithms, all simulations were performed with the respective capacities necessary to avoid any deadline violations. For, e.g., $U = 40\%$ and the EDF algorithm, the capacity C must be at least $2 \cdot C_{min}$ to satisfy this constraint (cp. Fig. 9). For $N = 1000$ randomly generated tasksets, Fig. 10 displays the ratio of average energy E_C to the respective capacity. Again, all Lazy Algorithms outperform EDF, but for increasing utilisation the differences decrease. Interestingly, LSA with ϵ^l stores even more energy than LSA with E_S since LSA with ϵ^l exhibits the latest starting times s of all four algorithms.

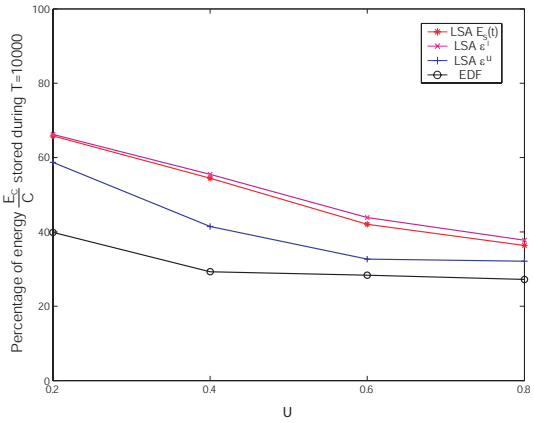

Figure 10. Average stored energy $\frac{E_C}{C}$

7. Conclusions

The paper studies the case of a sensor node which – equipped with an energy harvesting unit and a rechargeable battery – executes tasks defined by an arrival time, an energy demand and a deadline. We prove the optimality of two online scheduling algorithms that are based on the principle of laziness. For periodic, sporadic or bursty tasksets, a schedulability test is derived that is both, necessary and sufficient for the given model assumptions. This schedulability test shads light on the fundamental question of how to dimension the battery size of the device. We showed how to compute the minimum battery size required to maintain perpetual operation of the sensor node. Finally, simulation results are provided that validate the results of the paper and show the improvement in comparison to classical EDF scheduling. By introducing an appropriate characterization of the energy source, we found a useful approximation of the future produced energy.

Acknowledgements

The work presented in this paper was partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. In addition, this research has been founded by the European Network of Excellence ARTIST2.

References

[1] A. Allavena and D. Mossé. Scheduling of frame-based embedded systems with rechargeable batteries. In *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS 2001)*, 2001.

[2] S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[3] X. Jiang, J. Polastre, and D. E. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 463–468, UCLA, Los Angeles, California, USA, April 25-27 2005.

[4] A. Kansal, D. Potter, and M. B. Srivastava. Performance aware tasking for environmentally powered sensor networks. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2004*, pages 223–234, New York, NY, USA, June 10-14 2004. ACM Press.

[5] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. B. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 457–462, UCLA, Los Angeles, California, USA, April 25-27 2005.

[6] S. Roundy, D. Steingart, L. Frechette, P. K. Wright, and J. M. Rabaey. Power sources for wireless sensor networks. In *Wireless Sensor Networks, First European Workshop, EWSN 2004, Proceedings*, Lecture Notes in Computer Science, pages 1–17, Berlin, Germany, January 19-21 2004. Springer.

[7] C. Rusu, R. G. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. In *15th Euromicro Conference on Real-Time Systems, ECRTS 2003*, pages 95–104, Porto, Portugal, July 2-4 2003.

[8] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems, Springer Science+Business Media B.V.*, 9(2):205–225, 2005.