# An optimal energy aware aperiodic task server for autonomous IoT sensors

**Rola El Osta[1], Maryline Chetto[1,*], Hussein El Ghor[2]**

[1]LS2N Laboratory - UMR CNRS 6004, University of Nantes, Nantes, France
*Email: rola.elosta@ul.edu.lb, maryline.chetto@univ-nantes.fr*

[2]LENS Laboratory, Lebanese University, Saida, Lebanon
*Email: hussain@ul.edu.lb*

[*]*Corresponding author*

***Abstract*—A real-time applicative software consists of both aperiodic and periodic tasks. The periodic tasks have regular arrival times and strict deadlines. The aperiodic tasks have irregular arrival times and no deadline. The objective of an optimal aperiodic task server is to guarantee minimal response times for the aperiodic tasks and no violation of hard deadlines for periodic tasks. We consider a real-time energy harvesting system composed of energy harvester, energy storage unit, uniprocessing unit and the real-time tasks. We introduce a novel aperiodic task scheduler, namely SSP which is an extension of the slack stealing server so as to cope with fluctuations in energy availability. Experimental results show that the proposed algorithm achieves better performance in terms of aperiodic responsiveness. Simulations have been conducted for various settings of workloads and harvested energy profiles.**

***Index Terms*—Earliest Deadline First, energy harvesting, aperiodic servicing, preemptive scheduling, energy management.**

## I. INTRODUCTION

One of the key features for the next generation of portable wireless devices used for sensing, environmental/infrastructure monitoring, etc. is for them to become self-powered. Harvesting ambient energy from the environment to provide power for these devices has been recognized as one of the most promising technologies [1] [2]. Energy harvesting (EH) allows sensors to be placed where a connected network infrastructure is not available or practical. Photovoltaic cells, thermoelectric converters, and other energy harvesting techniques are available to harvest energy. Nevertheless, new problems arise because perpetual operation of an autonomous system implies *energy neutrality*: The system should never consume more energy than available. In contrast to classical battery-operated embedded systems, batteries are solely used as energy buffers and not as primary energy sources. As a positive consequence, the cost and size of batteries are reduced significantly.
In this paper, we consider an application software with real-time requirements where tasks are given by computation times as well as amount of energy required by their execution. Specifically, periodic tasks are given by periods and deadlines whereas aperiodic tasks have unpredictable arrival times and have no deadline. Scheduling periodic tasks with no energy limitation has been extensively studied for a long time [3]. The most famous result is certainly the proof of optimality of the Earliest Deadline First (EDF) scheduler reported in [4]. We recently proved that EDF which is greedy and non clairvoyant is no more suitable if the processor is supplied thanks to fluctuating regenerative energy [5]. Nonetheless, a clairvoyant and idling variant of EDF called ED-H has been proved optimal for uniprocessor energy harvesting systems where all the tasks, periodic or not, have hard deadlines to meet [6]. In this paper, we will focus on the scheduling issue where the application software is a mixed task set composed of both periodic hard deadline tasks and soft aperiodic tasks. The paper will address the following question: how to reduce the response time of aperiodic tasks while guaranteeing no deadline miss for the periodic tasks when all the tasks execute on a monoprocessor self-powered device? The paper is organized as follows: First, we describe a novel aperiodic task server which is an extension of the Slack Stealing server [7]. Second, by means of simulations, we demonstrate significant system performance improvement in aperiodic responsiveness comparing to Background servicing. Third, to provide insights for system designers, we report simulation results that show the impact of the aperiodic server on the response time for the soft aperiodic tasks.

This work is partially supported by a French grant from the Isite NExT (Project entitled "Optimisation et gestion en Temps Réel de la consommation Energétique d'un système Cobotique") within the *New Partnerships* framework.

## II. BACKGROUND AND RELATED WORK

### A. Aperiodic task servicing

The so-called TBS (Total Bandwidth Servicing) algorithm permits to execute aperiodic tasks with efficient responsiveness [8]. Once any aperiodic task arrives, it is assigned a virtual deadline. Then, it is jointly scheduled by EDF with the periodic tasks. The virtual deadline depends on processor utilization available for the aperiodic tasks called capacity of the server. An improved version of TBS called $TB^*$ was proved to be optimal [8]. An iterative process is applied so as

to shorten the virtual deadline and consequently improve the aperiodic response time. Such approach was extended so as to adapt to energy harvesting settings [9]. However, no theoretical analysis permits to state the relative performance of the so-called TB-H server.

Using the available slack of periodic tasks for advancing the execution of aperiodic requests is the basic principle adopted by the Earliest Deadline Late Server (EDL) [7]. The idea is to postpone the execution of periodic tasks as long as possible. The processor idle times of the so-called dynamic EDL schedule serve to perform the aperiodic tasks as soon as possible, profiting from the immediate available surplus of processing time called slack time. The EDL Server was proved optimal, that is, the response times of aperiodic tasks are the best achievable. In summary, whenever at least one aperiodic task enters the system, the dynamic EDL schedule is updated to predict future time intervals where periodic activities should be executed and future time intervals where aperiodic tasks should be executed for optimal responsiveness.
The pseudo-code of Algorithm 1 describes the outline of the EDL server.

**Algorithm 1** The Slack Stealing server EDL [9]

**Require:**
t: current time
$\Gamma(t)$: list of periodic tasks
$Ap(t)$: list of aperiodic tasks
**while** True **do**
  **if** $Ap(t)$ is not empty **then**
    Update the EDL schedule in order to execute the periodic tasks of $\Gamma(t)$ in the EDL busy periods
    **if** SlackTime(t)>0 **then**
      Schedule the tasks in $Ap(t)$ according to First Come First Serve
    **else**
      Schedule the tasks in $\Gamma(t)$ according to EDF
    **end if**
  **else**
    Schedule the tasks in $\Gamma(t)$ according to EDF
  **end if**
  $t := t + 1$
**end while**

### *B. Hard deadline scheduling with energy consideration*

As EDF, the energy harvesting aware ED-H algorithm chooses the ready task with the shortest deadline for execution. [6]. Nevertheless, ED-H may choose to postpone the completion of this task in order to avoid energy depletion in the future. As a result, the ED-H processor has Dynamic Power Management, which determines when the processor should be in the busy state and when it should be idle. At each instant, the choice is based on two dynamic variables respectively called *slack time* and *preemption slack energy*. The slack time is the longest period of time during which the processor can be left unoccupied without causing a deadline violation. The preemption slack energy is the maximum amount of energy that the active task can spend while ensuring that no task suffers from energy depletion.
Optimality of the ED-H scheduler has been established in [6]. If any algorithm can schedule a hard real-time task set on a platform with a specified processor, energy harvester, and energy reservoir, then the ED-H algorithm can schedule it on the same platform.

## III. SYSTEM MODEL AND ASSUMPTIONS

The energy harvesting system consists of four parts: energy harvester, energy storage, the processor and the real-time tasks (see figure 1). The energy harvester draws the energy

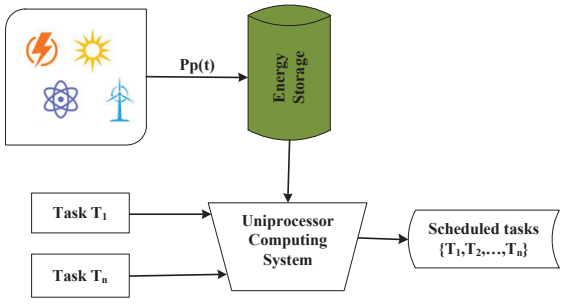


Fig. 1. Framework of an energy harvesting device

from an environmental source and feeds into the energy storage unit with instantaneous charging rate $P_p(t)$ at time t. We postulate that energy production and energy consumption coexist. Any task's instantaneous power consumption is not less than the source's instantaneous power consumption. The term "energy storage" refers to a reservoir that can be used to store energy. The energy level of the store at the time instant $t$ is given by $E(t)$. The stored energy can be utilised at any moment in the future. We are assuming an ideal storage unit with no leaks. When the storage unit is fully charged (i.e. $E(t) = E$ at time $t$), energy is dissipated. When the processor's energy storage capacity is spent, it can no longer function and shuts down. We assume that energy production and consumption can occur simultaneously.

A four-tuple $(C_i, E_i, D_i, T_i)$ is connected with a periodic task $tau_i$ and represents its Worst Case Execution Time (WCET), Worst Case Energy Consumption (WCEC), relative deadline, and period. We assume that $E_i$ and $C_i$ are not really proportional [10]. A task makes a request, which is represented by a job. The first job of $\tau_i$ is released at time 0 and the subsequent ones at times $kT_i, k = 1, 2, ...$ called release times. $H$ is the least common multiple of the request periods $T_i$, called the hyper-period. The processor utilization of the set of periodic tasks $\tau$ is $U_{pp} = \sum_{\tau_i \epsilon \tau} \frac{C_i}{T_i}$ which is lower than 1.
In addition, we consider $Ap$ the stream of $m$ soft aperiodic requests, defined as $Ap = \{Ap_i | 1 \leq i \leq m\}$ and $Ap_i =$

$(r_i, c_i, e_i)$. $r_i$ is the arrival time of the soft aperiodic task $Ap_i$. $c_i$ and $e_i$ are respectively the worst case execution time and the worst case energy requirement of $Ap_i$.

## IV. Energy harvesting aware aperiodic task servicing

In this section we will introduce a novel scheduling algorithm which is drawn from EDL and is adapted to mixed task sets in real-time systems with energy harvesting constraints. We suppose that the hard deadline periodic tasks are scheduled using to the optimal scheduler ED-H.

### A. SSP: Slack Stealing servicing

SSP (Slack Stealing with Energy Preserving) is a Slack Stealing server that works similarly to the EDL server but takes into consideration fluctuations in energy availability. The notion of slack will be discussed here in terms of both time and energy. The optimality of the SSP server is established in the sense that among all possible aperiodic servers for real-time energy harvesting systems, SSP gives the shortest aperiodic response time [9]. The main principle of the slack stealer SSP for aperiodic servicing with ED-H is to allow aperiodic job executions as long as it does not result in a deadline violation for all the jobs generated by the periodic task set $\tau$. It is recognized that a deadline violation can be caused by either processing time starvation (not having enough time to complete a task before deadline) or energy starvation (not have enough energy to complete a task before deadline).

Consequently, we can think of the system slack at current time $t$ as a pair of elements denoted slack time and slack energy, respectively. The slack time of $tau$ at time $t$ is defined as the maximum processing time available at $t$ after completing the tasks of $tau$ on time. Slack time is a variable value that represents the variation of processing surplus. Its computation allows it to determine how long the processor can be idle or busy executing additional tasks, such as aperiodic ones, at any time.

The slack energy of $tau$ at time $t$ is defined as the maximum energy available at $t$ after completing the tasks of $tau$ on time. Slack energy is a dynamic value that expresses changes in energy surplus. Its computation allows it to determine how much energy could be wasted or consumed by executing additional tasks, such as aperiodic ones, at any time.

When the aperiodic queue is not empty, the slack stealer SSP can be seen as a task that is ready to execute. When the queue is empty, this task is paused. When there is slack, such as slack time or slack energy, the slack stealer is given top priority. Details on how to calculate slack time and slack energy are given in [9]. It receives the lowest priority whenever there is either no slack time or no slack energy. The slack stealer SSP selects the aperiodic tasks in FCFS order.

The framework of the SSP server is described by the pseudo-code of Algorithm 2.

**Algorithm 2** The Slack Stealing server SSP

**Require:**
t: current time
$L(t)$: list of periodic tasks at t
$Ap(t)$: list of aperiodic tasks at t
**while** TRUE **do**
  **if** Ap(t) is not empty AND energy reservoir is not empty AND $SlackEnergy(t) > 0$ AND $SlackTime(t) > 0$ **then**
    Schedule the tasks in $Ap(t)$ according to First Come First Serve
  **else**
    Schedule the tasks in $\Gamma(t)$ according to ED-H
  **end if**
  $t := t + 1$
**end while**

### B. Implementation considerations

SSP has a $O(m.n)$ complexity, where $m$ is the number of iterations and $n$ is the number of periodic tasks. The complexity of the algorithm is pseudo-polynomial since the number of iterations, $m$, is dependent on the periods and deadlines of the hard deadline tasks. The frequent slack time and slack energy calculations in SSP cause rather substantial time overheads. However, under the optimal slack stealing strategy, extra processing time and energy are leveraged whenever possible for aperiodic activities by procrastinating periodic operations.

## V. Simulations and discussion

### A. Introduction to the experiment

When implementing an aperiodic task algorithm, many factors affect aperiodic response time performance:

- The ratio of processor utilization and the ratio of harvested energy used by the periodic tasks determine the quantity of energy and processing times available for the aperiodic tasks.
- The aperiodic processing load and the ratio of energy needed by the aperiodic tasks have a direct effect upon aperiodic response time performance. The smaller both the quantity of processing times and energy required by the aperiodic tasks are, then the more likely it is that the aperiodic tasks are served immediately.

The experiment compares and evaluates the performance of aperiodic task servers using simulations. The SSP server will be pitted against two background servers: Background with Energy Surplus (BES) and Background with Energy Preserving (BEP). When no periodic tasks are present in the system and the energy storage is fully replenished, BES serves aperiodic tasks. BEP, the enhanced version of BES, allows

any aperiodic task to run if its execution does not cause an energy shortage for future occurring periodic tasks. In terms of processing loads and energy requirements, we will consider various application profiles. In addition, the experiment will show how different parameters (such as the power provided by the environmental source) affect the response time of soft aperiodic tasks.

The ED-H algorithm, as previously stated, is used to schedule periodic tasks. The FCFS policy governs how aperiodic tasks are handled. The metrics used to evaluate the performance of the aperiodic servers SSP, BEP, and BES are now presented. The obtained results here are based on the assumption that

- The overall processing load $U_p$ is made up of $50\%$ of periodic processor utilization $U_{pp}$ and $50\%$ of aperiodic processor utilization $U_{ps}$.
- In the same way, the total energy load $U_e$ includes $50\%$ of the periodic energy usage $U_{ep}$ and $50\%$ of the aperiodic energy utilization $U_{es}$.

We will explain how each aperiodic task server behaves from various angles, including the average response time of aperiodic tasks. It is worth emphasizing that the goal is to reduce the mean response time of soft aperiodic tasks while maintaining the schedulability of periodic tasks with the lowest possible implementation costs, i.e. the fewest possible preemptions and computing operations.

### B. Simulation Environment

We used Matlab to create a simulator. The two-dimensional and three-dimensional graphs that result are plotted with high resolution. Our code is capable of producing any simulation with parameters specified by the user. The generator of periodic tasks takes as input the number of tasks *n*, the hyperperiod *H*, processing utilization $U_{pp}$, and energy utilization $U_{ep}$.

The simulator automatically generates a periodic task set $\tau$ where each task $\tau_i$ is characterized by quadruple $(C_i, E_i, D_i, T_i) \mid 1 \leq i \leq n\}$. Periods and computation times are uniformly distributed, in dependance of $U_{pp} = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Energy consumption of every task is proportional to its period and depends on the setting of $U_{ep} = \sum_{i=1}^{n} \frac{E_i}{T_i}$. Periodic task sets are generated so as to guarantee feasibility in terms of processing time and energy consumption i.e. $U_{pp} \leq 1$ and $U_{ep} \leq P_p$ where $P_p$ is the average recharging power.

The input data of the aperiodic tasks are number of desired tasks *m*, processing utilization factor $U_{ps}$ and energy utilization $U_{es}$. A stream of aperiodic tasks with uniform distribution is generated by simulating a Poisson aperiodic arrival pattern.

A simulation run consists of one task set composed of 20 periodic tasks. Simulations are performed on 10 hyperperiods to reduce the bias effect of random generation procedure. Each point on the curves corresponds to 100 runs. The energy storage unit is assumed to be initially full. Its capacity is equal to $E_{min}$, defined as the minimum size of the energy reservoir that guarantees feasibility. The energy produced by the source is uncontrollable. We assume that it is possible to accurately estimate short-term energy within some prediction errors margin. Four different energy profiles extracted from [11] have been considered in our performance evaluation: constant, sine wave signal of period $\pi = 2$, a rectifier, and a pulse signal with a $20\%$ duty-cycle (Figure 2). The power for the constant profile was supposed constant and equal to 5. The output power of the three profiles is supposed variable between 2 and 17. It is worth mentioning that $E^p_{min}$ of each profile $p$ should not be less than the area $A_p$.

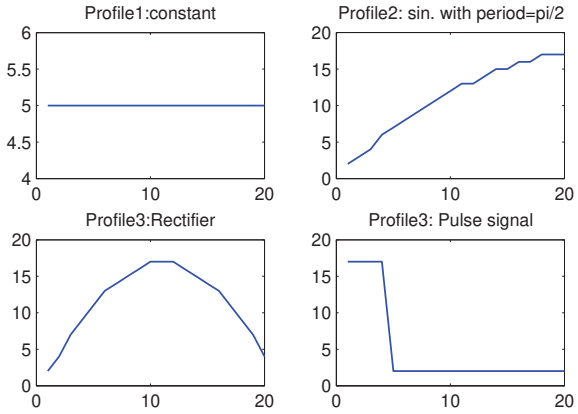


Fig. 2. Energy source profiles under study.

The following metrics are addressed in our study:

1) *Average response time of aperiodic tasks* defined as average time between arrival and completion, normalized with regards to average computation time. It is defined as:
$$ART_i = \frac{f_i - a_i}{c_i} \quad (1)$$
For example, a y-axis value of 10 indicates that the average response time is 10 times greater than the computation time.
2) *Average jitter of aperiodic tasks*: Real-time systems, especially software control systems, are developed to meet the requirements of real-time automation systems. One issue is to minimize the delay and jitter of tasks. Jitter represents the induced offset between the release time of the aperiodic task and its actual starting time. Therefore, we evaluate the average jitter which is normalized with respect to its response time.
As a result, a jitter of 1 on the y-axis corresponds to its response time; a jitter of 0 relates to the shortest possible jitter time and indicates that the task was completed without being blocked.

### C. Experiment Results for Average response time of aperiodic tasks

The evaluations are carried out in this initial set of experiments for a total energy utilization applied to the

system that ranges from $5\%$ to $100\%$, while the total processing utilization load remains constant ($Up = 0.6$). Figures 3, 4, 5 and 6 show the results for a constant profile, a Sine wave with period $pi = 2$, a Rectifier signal, and a Pulse signal with a $20\%$ duty-cycle, respectively. They show the normalized mean aperiodic response time in relation to the aperiodic computation time.

SSP clearly beats the other algorithms across all energy profiles, as it takes advantage of time slack stealing to maximize CPU utilization. This demonstrates that our theoretical research was carried out without making any assumptions about energy production through time. Under the Pulse signal profile, BEP outperforms BES by a tiny margin and shows a considerable decline in comparison to the BEP method (Figure 6). The results of the Pulse model are predicted to reveal that the performance of the three algorithms, especially BES, is slightly lower than the three other models. BES, for example, is at least $12.5\%$ percent less responsive under the Pulse signal profile than under the other modes. The reason for this is that power is only gathered on a $20\%$ duty-cycle of the overall signal, and BES allows aperiodic tasks to be performed only when the energy reservoir is full, resulting in increased aperiodic responsiveness.

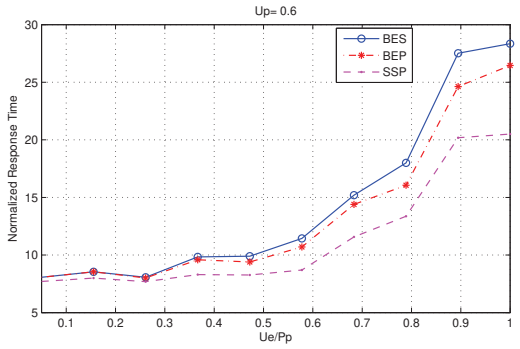


Fig. 3. Aperiodic response time with respect to $U_e/P_p$, for $U_p$=0.6 under constant profile.

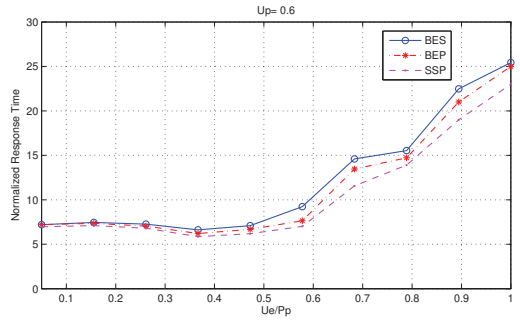


Fig. 4. Aperiodic response time with respect to $U_e/P_p$, for $U_p$=0.6 under sinusoidal signal of period $\pi/2$.

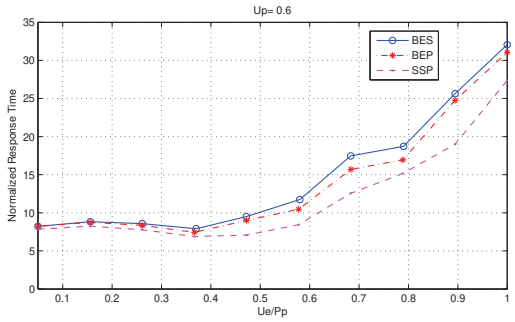


Fig. 5. Aperiodic response time with respect to $U_e/P_p$, for $U_p$=0.6 under rectifier signal.

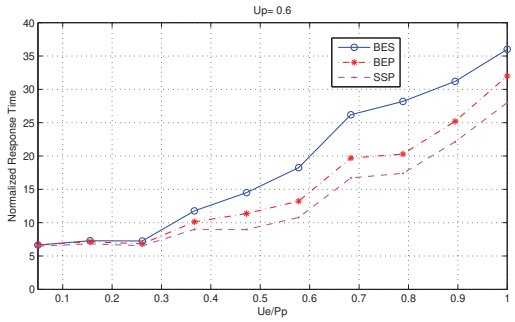


Fig. 6. Aperiodic response time with respect to $U_e/P_p$, for $U_p$=0.6 under pulse signal.

### *D. Experiment Results for Average jitter of aperiodic tasks*

This section includes a set of experiments that demonstrate how SSP minimizes aperiodic task delays and jitters in energy-constrained real-time systems, as enforced by the operating system, control tasks, kernel mechanisms,etc. The induced offset between the aperiodic task's release time and its execution start is referred to as jitter.

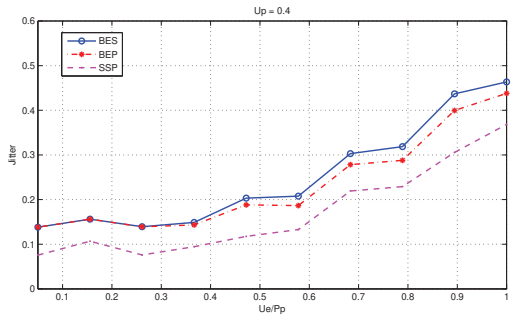


Fig. 7. Jitter time with respect to $U_e/P_p$, for $U_p$=0.4 under constant profile.

For a constant total processing utilization of 0.4, SSP is evaluated as a function of total energy consumption $U_e/P_p$ and jitter is compared to background policies. Average jitter time is normalized with respect to its response time. Figures 7, 8, 9, and 10 show the simulation results for a constant profile, a Sine wave with a period of $\pi = 2$, a Rectifier signal,

and a Pulse signal with a duty-cycle of 20% duty-cycle respectively.

From the graphs, we observe that the SSP server outstands the background servers by reducing the delay and jitter of the aperiodic tasks for all energy loads and under all energy profiles. For example, the results in Figure 7 show that the jitter of SSP is at least 16% lower than BEP and BES. Furthermore, higher is the energy load $U_e/P_p$, more important is this advantage.

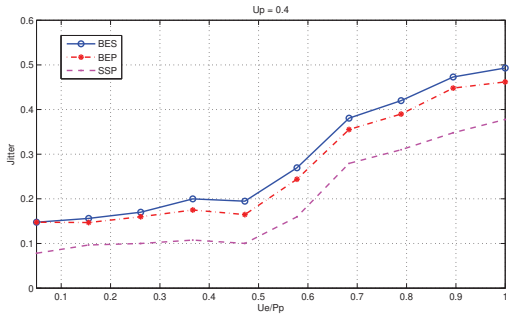


Fig. 8. Jitter time with respect to $U_e/P_p$, for $U_p$=0.4 under sinusoidal signal of period $\pi/2$.

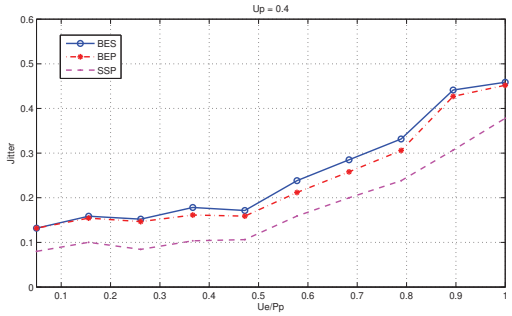


Fig. 9. Jitter time with respect to $U_e/P_p$, for $U_p$=0.4 under rectifier signal.

It is worth mentioning that BES exemplifies the inferior server for all energy profiles, and shows a significant degradation compared to BEP (Figure 10) under the Pulse signal profile owing to the power harvested by the Pulse profile and to the performance of the BES algorithm. For example, the jitter time variance between BES and BEP under the Pulse signal profile is 26% at $U_e/P_p$=6 (Figure 10), while it is equal to 10% under the constant profile (Figure 7).

## VI. CONCLUSION

Energy autonomous systems are becoming an important class of embedded real-time systems that utilize ambient energy to perform their computations. They do not need to charge the battery cyclically since they rely on an external energy supply. In this paper, we investigated the problem of scheduling real-time tasks implemented on a single processor

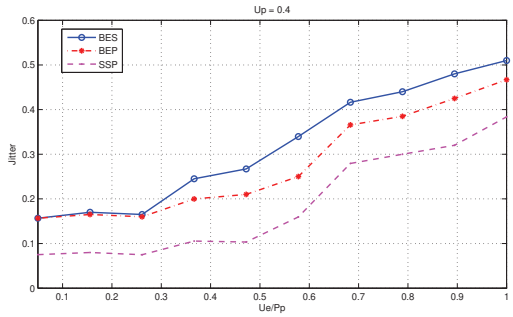


Fig. 10. Jitter time with respect to $U_e/P_p$, for $U_p$=0.4 under pulse signal.

supplied with energy harvesting. We considered hard periodic and soft aperiodic real-time tasks. Periodic tasks are scheduled according to the optimal scheduler ED-H. We have described a novel aperiodic task server, proved to be theoretically optimal in terms of aperiodic responsiveness. The so-called aperiodic task servicing algorithm, SSP, suggests to steal both processing time and environmental energy so as to execute the aperiodic tasks as soon as possible with no deadline violation and no energy starvation. The experimental study reported in the paper permits to measure the actual performance gain of SSP, reduced by at least 25%, in comparison to classical Background servicing techniques for all energy conditions .

## REFERENCES

[1] S. Chalasani and J. M. Conrad, "A survey of energy harvesting sources for embedded systems", *In Proceedings of the IEEE Southeastcon 2008*, pp. 442–447, April 2008.
[2] F. Yildiz, "Potential ambient energy-harvesting sources and techniques", *The Journal of Technology Studies*, vol. 35, no. 1, pp. 40–48, 2009.
[3] J. Liu, *Real-Time Systems*, Prentice Hall, 2000.
[4] C.-L. Liu and J.-W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, 1973.
[5] M. Chetto and A. Queudet, "A Note on EDF Scheduling for Real-Time Energy Harvesting Systems", *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 1037–1040, April 2014.
[6] M. Chetto, "Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems", *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 122–133, 2014.
[7] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, 1989.
[8] G.C. Buttazzo and F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Enviroments," in *Proc. of the 3rd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'97)*, Como, Italy, pp. 39–48, September 1997.
[9] R. El Osta, M. Chetto and H. El Ghor, "Optimal Slack Stealing Servicing for Real-Time Energy Harvesting Systems", *The Computer Journal*, vol. 63, no. 10, pp. 1537-–1546, October 2020.
[10] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the Worst-Case Energy Consumption of Embedded Software", *Proc. of 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 81–90, 2006.
[11] S. Liu, J. Lu, Q. Wu and Q. Qiu, "Harvesting-Aware Power Management for Real-Time Systems With Renewable Energy , *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1473—1486, August 2012.