Contributions to Real Time Scheduling for Energy Autonomous Systems.

Rola El Osta

Thèse de Doctorat

Rola EL OSTA

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Automatique et Informatique Appliquée, section CNU 61
Spécialité : Informatique
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le 26 octobre 2017
Thèse n° : ED xyz

Contributions to Real Time Scheduling for Energy Autonomous Systems.

JURY

Président :	**M. Abdelhamid MELLOUK**, Professeur, Université de Paris 12, France
Rapporteurs :	**M. Mathieu JAN**, Ingénieur de Recherche, HDR, CEA, France
	M. Daniel CHILLET, Professeur, Université de Rennes 1, France
Examinateur :	**M. Hussein EL GHOR**, Maître de conférences, IUT de Saida, Université libanaise, Liban
Directrice de thèse :	**M^{me} Maryline CHETTO**, Professeur, Université de Nantes, France
Co-directeur de thèse :	**M. Rafic HAGE**, Professeur, IUT de Saida, Université libanaise, Liban

ACKNOWLEDGEMENTS

This thesis is the result of hard work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

First and foremost, I would like to thank my supervisor Prof. Maryline Chetto, not to comply with customs, but to owe my deepest gratitude to her. I would never have completed this effort without her common-sense, knowledge, perceptiveness and being able to fully rely on her professional and personal support. The comments and time given by her have greatly improved and clarified this work.

I would like also to express my profound gratitude to my second supervisor Prof. Rafic Hage who deserves the great deal of thanks for his fruitful guidance and continuous support throughout these years. My deep appreciation goes out to Dr. Hussein El Ghor. His guidance and expertise into the world of real-time systems have been a valuable input for this thesis. Many thanks for him for being patiently listening to me and for being available whenever I needed a meeting with him despite his busy schedule.

I would like also to thank the committee members: Dr. Mathieu JAN, Prof. Daniel CHILLET and Prof. Abdel Hamid MELLOUK for the time they spent in reading my thesis.

Beside my advisors, I greatly acknowledge all permanent and temporary team of the LS2N laboratory.

I am indebted to my friends Amina in Lebanon and Fatat in Nantes who were always so helpful in numerous ways.

I would also like to say a heartfelt thank you to my family for always believing in me and encouraging me to follow my dreams. Thank you my parents for your unconditional love and support and for giving me the opportunities and experiences that have made me who I am.

Thank you my husband for having been by my side throughout this PhD, living every single minute of it, and without whom, I would not have had the courage to embark on this journey. I will never forget his supportive and understanding attitude when I was nervous or overloaded with work. His presence in my life was a great bless. I owe special thanks to my children who have strewn my life with thousands of roses.

I would like to thank my brother who helped me in whatever way he could during this challenging period and my sister who were always listening to me with the big heart of a sister and the attentive ear of a confidante.

Finally, I am thankful for Allah the Almighty, my success can only come from Him.

To the energy of my soul ...
To my family ...

CONTENTS

List of Figures

LIST OF TABLES

GENERAL INTRODUCTION

Thesis objectives

Battery-powered embedded devices are characterized by limited battery lifespan. In most of these devices and in some applications, such as military, health, and environmental, replacing batteries makes them less affordable. Consequently, green solutions based on environmental energy have become economically conceivable for many application fields. Furthermore, energy harvesting for small devices such as wireless sensors offers many technological advantages, such as the continuous replenishment of battery or capacitor, better flexibility and reliability for remote monitoring, complete autonomy and efficiency [4], [5]. As a consequence, energy harvesting technology has been facing a spectacular growth in global interest in the last decade and this development will continue in the years to come.

This thesis seeks to settle a crucial scheduling problem in hard real-time systems with energy harvesting considerations often called autonomous real-time systems. In general, such systems consist of a set of programs called tasks with periodic executions, each one in charge of controlling/monitoring an external environment [6]. In addition, some tasks said to be aperiodic may occur and require to be executed as soon as possible while preserving feasibility of the periodic tasks. Under fixed priority as well as dynamic priority settings, many research efforts have been made from the eighties in order to propose solutions to solve the underlying scheduling problem [7]. This problem can be simply described as follows: how to execute jointly the hard periodic tasks set mixed with the dynamically occurring soft aperiodic tasks? Many scheduling solutions i.e. aperiodic task servers, have been proposed in the literature, including several ones proved to be optimal. However, all of them have been designed for systems with no energy limitation and they do not take energy into consideration. Providing an optimal solution for the energy harvesting context represents the central objective of this work for dynamic priority systems. Accordingly, this thesis provides a novel set of approaches to cope with the problem of minimizing aperiodic responsiveness while guaranteeing the schedulability of periodic tasks. ED-H (earliest deadline for energy harvesting) has been proved, in 2014, the optimal scheduling scheme for real-time energy harvesting (RTEH) systems composed of hard deadline tasks [8]. Consequently, it will represent the bearing of our research.

Thesis contributions

When processor time and energy resource are limited, they have to be exploited as efficiently as possible to respect timing constraints. In RTEH systems, ED-H is the optimal uniprocessor scheduling strategy that is responsible for scheduling hard deadline jobs by smartly using both processor time and energy resource so as to meet the deadlines and to avoid energy starvation. ED-H is based on computing two key data, on line, respectively called slack time and slack energy. According with these requirements, the existing ED-H scheduling algorithm would be merged with other strategies in order to tackle the scheduling problem that concerns minimization of aperiodic response times while guaranteeing the timing constraints of jobs that issue from the periodic tasks. Our study has lead to three novel aperiodic task scheduling algorithms often known as aperiodic servers. Firstly, we propose two servers, namely BEP (Background with Energy

Preserving) and BES (Background with Energy Surplus) that both rely on the classical background approach [9]. Our experimental evaluation shows that BEP outperforms BES in terms of aperiodic response time but with much more overhead. The major disadvantage of these two servers lies in their limited performance even if they offer relatively simple implementation.

For that reason, the SSP (Slack Stealing with energy Preserving) aperiodic task server is proposed. Based on the slack stealing mechanism, it profits whenever possible from available extra processing time and available extra energy so as to service the aperiodic tasks as soon as possible [10], [11], [12], [13]. Our key contribution here is the optimality proof for this new server in a context of energy harvesting. We illustrate the performance improvements, in terms of aperiodic responsiveness and other criteria, which are brought by this approach, compared to the previous background approaches.

To cope with the time complexity issue that characterizes the previous servers, we finally propose an approach which is based on the bandwidth preserving technique that was initially proposed at the end of the eighties by Buttazzo et all [13], [14], [15]. In other words, we show how to extend the famous TB server to the energy harvesting context. We show how to adequately assign a fictive deadline to any occurring aperiodic task and permit to schedule it according to the ED-H algorithm together with the periodic tasks. The experimentation permits to state its good behaviour in terms of implementation complexity and scheduling performance.

Thesis organization

The **first** chapter of this manuscript starts with description of the basic and necessary concepts relative to real-time computing systems.

Chapter 2 focuses on new generation systems which are powered through the environment, called energy harvesting systems. This chapter presents a description of the main energy storage devices and available energy sources dedicated to energy harvesting technology.

Chapter 3 treats the scheduling problem in energy autonomous systems. It describes the model for the energy harvesting system under study. And we recall the principles of ED-H the scheduling policy for periodic tasks that was adopted in this thesis. The main relevant approaches for scheduling aperiodic tasks with no energy limitation, which were established previously in the literature, are also reviewed.

Chapter 4 is the first contribution. It formalizes the problem of jointly scheduling soft aperiodic tasks and hard deadline periodic tasks under energy constraints. Two novel solutions for aperiodic servicing with energy harvesting considerations are proposed. Both are based on background approaches.

Chapter 5 investigates the problem of improving the response times of aperiodic tasks due to limitations of background servers. Based on the slack stealing technique, a novel aperiodic task server, called SSP is described. The theoretical performance analysis is presented and proves the theoretical optimality of this server.

In **Chapter 6**, the simulations that were carried out effectively show that the proposed slack stealing-based mechanism satisfies the optimal responsiveness of aperiodic tasks. The performance evaluation of SSP promises and demonstrates that it reduces the aperiodic responsiveness among different conditions when compared to background approaches.

In order to further improve the run-time overhead incurred by the use of SSP, the TB-H server and its main properties are introduced in **Chapter 7**. Experimental results show that this server provides good performance with reasonable complexity. This improvement is realized by giving the total bandwidth of the server, whenever possible, to each aperiodic task that enters in the system. Consequently, this chapter

shows that the TB-H is a good candidate for aperiodic servicing.

Finally, a general summary of our work is presented and recalls the main results that have been obtained for aperiodic task scheduling in a real time energy harvesting context. Some perspectives are additionaly given for the continuation of our work.

PART I

STATE OF THE ART

CHAPTER 1

FUNDAMENTALS OF REAL-TIME COMPUTING

Summary

This chapter presents an introduction to real-time computing systems. It includes description of the classical techniques or approaches proposed by the literature for the analysis and scheduling of hard real-time systems. The main idea is to give an overview of the state of the art in this domain, mainly dedicated to mono-processor architectures. Our objective is to justify some choice for our analytical approach and assumptions that we will describe later on.

Contents

1.1 Real-time Systems

1.1.1 Basic Concepts

"The actual time during which a process or event occurs" is the definition of *real-time* by the Oxford dictionary. Technically, the real-time systems are computing systems that have timing constraints, i.e. they are characterized by the fact that they require temporal correctness as well as logical correctness. In other words, "the correctness of the systems doesn't depend on the computational results only, but also on the

time when they are produced" [16]. Today, real-time computing plays a crucial role in our society, since an increasing number of complex systems relies, in part or completely, on computer control/monitoring. Examples of applications that require real-time computing include the following:

- Aircraft Monitoring applications: systems are responsible of automatic navigation, detection of hardware malfunctions and of monitoring a set of smoke detectors on the aircraft board. Warning lamps are illuminated with red when consecutive non valid readings are received from sensors or when smoke is detected.
- Automotive management applications: systems monitor and control the speed of the car using a cruise control function. It also monitors the mileage, average speed, and fuel consumption.
- Multimedia and entertainment applications: systems periodically perform the following jobs: read, decompress, and display video and audio streams.
- Metal industry applications: systems are typically used in controlling processes such as casting, hot rolling, cold rolling, finishing, annealing, soaking, and other metal processing functions.
- Electric utility monitoring and control applications: computers are needed in this application to monitor and control plant equipment, to ensure optimal operation and safety, and to prevent costly unscheduled outages. Large quantities of coal or oil are typically consumed by the boilers in the utilities plants. A slight deviation from the optimal efficient performance of these boilers, even for a short period of time, can seriously impact the cost of electrical energy.
- Petrochemical applications: the production of high commodity chemicals such as ethylene and propylene is supervised and controlled by computers. These systems require high performance real-time features and should provide interfaces to regulatory control instrumentation systems and Programmable logic controllers. In petrochemical applications in general, safety related requirements are very important since a system failure may cause an environmental catastrophe.
- Mobile and data communication applications: systems are used as communication processors to provide key features of packet switched networks such as high speed real-time communication, performance capabilities that can handle both on-line, concurrently with background communication tasks, as well as extensive line handling for communication protocols.

Based on these examples of real-time applications, we can notice that the concept of time is not an intrinsic property of a control system, either natural or artificial, but it is strictly related to the environment in which the system operates. It does not make sense to design a real-time computing system for flight control without considering the timing characteristics of the aircraft. More precisely, a real-time computing system should be predictable i.e. every real-time computation must be completed in an interval of certain length. The beginning of the interval is called *release time* and the end is called *deadline*. Depending on the consequences that may occur because of a missed deadline, three categories of real time system can be distinguished:

Hard: producing the results after a given timing constraint (deadline) may cause catastrophic consequences on the system under control.

Firm: producing the results after the deadline is useless for the system, but does not cause any damage.

Soft: producing the results after the deadline has still some utility for the system, although it causes a performance degradation.

For instance, aircraft monitoring and control systems require a strict respect of timing constraints, whereas media and communication systems can tolerate timing delays without major consequences.

1.1.2 States of a Task

The main software part of a real-time system consists of tasks, i.e., computing processes. A task is a computation code that has to be executed by the CPU. In that thesis, we consider that tasks are totally independent, do not synchronize with each others and do not suspend except for preemption of the scheduler.

The main objective of a real-time scheduler is to guarantee the correctness of the results while respecting the timing constraints of the tasks (no deadline miss). It is important to clarify that real-time scheduling

does not necessarily mean executing tasks as soon as possible, but instead taking scheduling decisions that guarantee that the timing constraints are satisfied. Based on the decisions of the scheduler, a real-time task can be in one of the three following states:

Running A task enters this state as it executes on the processor.

Ready A task is ready when it has received its release signal but is waiting (volontarily or not) for the processor. All descriptors (defined as a data structure associated to every task) of ready tasks are maintained in a queue, called the ready queue.

Waiting A task is waiting when blocked until the occurrence of a specific event such as synchronization event or a release event.

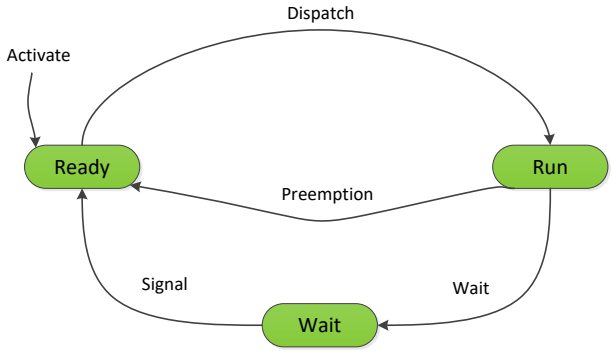


Figure 1.1: States transition for a real-time task

The different states of tasks are shown in Figure 1.1. Moreover, a real-time scheduler controls the transitions between the ready and running states of tasks, but it has no control over the external events that block the execution of tasks.

1.1.3 Real-time Task Model

A real-time application is composed of a set of tasks denoted by τ. It is composed of n tasks where $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task τ_i is assumed to generate one or more identical instances which are called jobs. In general, a real-time task τ_i can be characterized by the following parameters:

Arrival time a_i is the time at which a task becomes ready for execution; it is also referred as request time or release time and indicated by r_i.

Computation time C_i is the time necessary for the processor to execute the task without interruption.

Absolute Deadline d_i is the time before which a task should be completed to avoid damage to the system.

Relative Deadline D_i is the difference between the absolute deadline and the request time: $D_i = d_i - r_i$.

Start time s_i is the time at which a task starts its execution.

Finishing time f_i is the time at which a task finishes its execution.

Response time R_i is the difference between the finishing time and the request time: $R_i = f_i - r_i$.

Criticality is a parameter related to the consequences of missing the deadline (typically, it can be hard, firm, or soft).

Value v_i represents the relative importance of the task with respect to the other tasks in the system.

Lateness L_i: $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline (note that if a task is completed before the deadline, its lateness will be negative).

Tardiness or Exceeding time E_i: $E_i = max(0, L_i)$ is the time a task stays active after its deadline.

Laxity or Slack time X_i: $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its activation to complete within its deadline.

There are possibly additional important parameters that characterize real-time tasks [17, 18]. For example, it could be:

- Resources Constraints i.e. those derived from the access with mutual exclusion to critical resources.
- Synchronization Constraints that can be described by a set of precedence relations (or prior) which determine the order in which tasks have to undergo their treatment. When there is no precedence relation between tasks, tasks are said to be independent.
- Execution Constraints are based on two types of tasks, preemptible and non-preemptible. When any task is preemptible, its execution can be interrupted at any moment and resumed later. In contrast, a non preemptive task executes with no interruption from its start time to its finishing time.

The real-time computing theory also allows for the classification of tasks as periodic or aperiodic. *Periodic tasks* consist of an infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate. For the sake of clarity, from now on, a periodic task set τ will be denoted as follows: $\tau = \{\tau_i \mid 1 \leq i \leq n\}$. Each periodic task τ_i has a period T_i, a relative deadline D_i and a constant worst case execution time (WCET) C_i (normalized to processor computing capacity). We consider a constrained-deadline task set τ in which $0 < C_i \leq D_i \leq T_i$. Task τ_i generates jobs which are released at times 0, T_i, $2T_i$,... and must complete by times D_i, $T_i + D_i$, $2T_i + D_i$,... The hyper-period H of a periodic task set is defined as the least common multiple (LCM) of the request periods T_i, that is $H = LCM(T_1, T_2, ..., T_n)$. The processor utilization of the periodic task set τ is $U_{pp} = \sum_{\tau_i \epsilon \tau} \frac{C_i}{T_i}$ and is less than or equal to 1. A job is any request that a task makes. (r_j, C_j, d_j) is associated with a job J_j and gives its release time, WCET and (absolute) deadline, respectively.

Aperiodic tasks also consist of an infinite sequence of identical jobs (or instances); however, their activations are not regularly interleaved. An aperiodic task, where consecutive jobs are separated by a minimum inter-arrival time, is called a sporadic task. We will use the following notation throughout the thesis: Ap is a stream of aperiodic occurrences defined as $Ap = Ap_i(a_i, c_i), i = 1..m$, where a_i is the arrival time and c_i is the worst case execution time. The finish time of Ap_i will be denoted by f_i. Figures 1.2 and 1.3 show an example of task instances for a periodic and an aperiodic task.

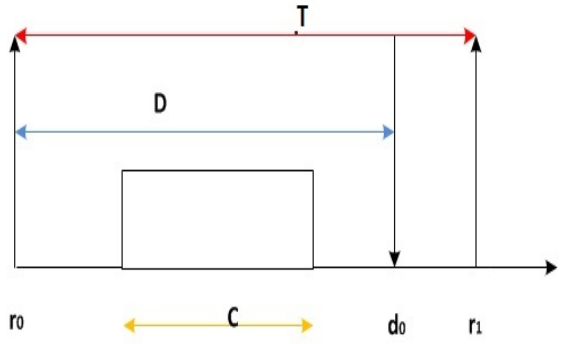


Figure 1.2: Model of a periodic job

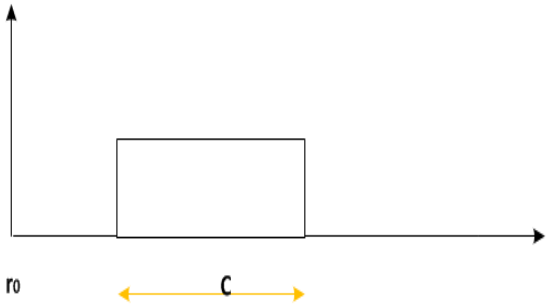


Figure 1.3: Model of an aperiodic job

1.2 Scheduling Problems

A real-time scheduling problem describes the conflicts due to concurrent accesses to the processor by the tasks. Typically, a real-time scheduler takes its decision based on the timing parameters of the ready tasks. The scheduling function is a service of the operating system (the scheduler), which allocates the

processor along time in accordance with the tasks in the ready state. According to the performance criteria in general maximization of deadline success, the scheduler determines for every time interval, identity of the task to execute on the processor. A scheduler may implement one or more scheduling algorithms that specify the rules for selecting the ready task that will be running. The time schedule of the tasks which is constructed by a scheduling algorithm is called *sequence* or *schedule* and is generally represented by a Gantt chart. Each line is associated to a given task. An additional line may be used to describe busy vs idle periods of the processor.

1.2.1 Classification of Scheduling Algorithms

Real-time scheduling is divided into several categories based on various criteria: the scheduling rule and instants where to apply the scheduling rule. Most of real time schedulers are priority driven ones and preemptive. They can be classified as follows:

Preemptive vs. Non-preemptive – In preemptive algorithms, the running task can be interrupted at any time to assign the processor to another ready task, according to a predefined scheduling policy.
- In non-preemptive algorithms, a task, once started, is executed by the processor until completion. In this case, all scheduling decisions are taken when any task terminates its execution.

Static vs. Dynamic – Static priority algorithms are those in which scheduling decisions are based on fixed parameters, assigned to tasks before the activation starts.
- Dynamic priority algorithms are those in which scheduling decisions are based on dynamic parameters that may change along time during the application lifetime.

Off-line vs. Online – A scheduling algorithm is off-line if it constructs the sequence on the entire task set before the system starts operation. The schedule generated in this way is stored in a table and is later executed by a dispatcher.
- A scheduling algorithm is online if scheduling decisions are taken at runtime every time a new task enters the system, when a running task terminates or more generally when an event should be taken into account. In a next chapter, we will see that such an event may connected to the energy storage.

Optimal vs. Heuristic – An algorithm is said to be optimal if it minimizes some given cost function defined over the task set. When no cost function is defined and the only concern is to achieve a feasible schedule, an algorithm is said to be optimal if it is able to find a feasible schedule, if one exists.
- An algorithm is said to be heuristic if it is guided by a heuristic function when taking its scheduling decisions. A heuristic algorithm tends toward the optimal schedule, but does not guarantee finding it.

Monoprocessor vs. Multiprocessor – An algorithm is said to be monoprocessor, when all tasks can only run on the same processor.
- An algorithm is said to be multiprocessor when multiple processors are available in the system.

Idling vs. Non-idling – An algorithm is said to be non-idling or work-conservative if a processor performs the highest priority task as soon as it is ready for execution and can not delay it if it has nothing to do, i.e., it works without the insertion of idle time.
- An algorithm is said to be idling; when a task is ready, it may be elected or it may wait for a period of time before running, even if the processor is free.

Centralized vs. Distributed – An algorithm is said to be distributed if the scheduling decisions are taken by an algorithm locally at each node.
- An algorithm is said to be centralized when the scheduling algorithm for the whole system, whether distributed or not, is extracted on a privileged node.

Clairvoyant vs. Non Clairvoyant – An algorithm is said to be clairvoyant if it knows the future; that is, it knows in advance the arrival times of all the tasks. Although such an algorithm does not exist in

reality, it can be used for comparing the performance of real algorithms against the best possible one.

1.2.2 Real-time Feasibility and Schedulability

We present here a description of the properties of the scheduling algorithms presented in terms of schedulability analysis and feasibility:

Definition 1 *A schedule Γ for τ is said to be valid if the deadlines of all jobs of τ are met in Γ.*

Definition 2 *A system is feasible if there exists at least one valid schedule for τ.*

Definition 3 *An algorithm is optimal if it finds a valid schedule whenever one exists*

Definition 4 *A schedulability analysis or feasibility analysis is performed to test the validation of a system on an off-line analysis.*

1.2.3 Complexity of Scheduling Algorithms

Associated with the metrics described above, the *efficiency* of a scheduling algorithm is also evaluated based on its *computational complexity* [19, 20]. In general, the overhead incurred by scheduling at run time is calculated by evaluating the number of implemented elementary operations. It means that the number of basic instructions of any programming language (addition, subtraction, assignment, test,...) and this, according to the number of input data of the problem. Let ξ be the complexity function of an algorithm representing the largest number of elementary operations that this algorithm requires to solve a problem Π, and n the size of the problem Π, that means the amount of input data required to write Π. This leads to the following definitions [21]:

Definition 5 *An algorithm is said to be **polynomial** time if its complexity function ξ is O(p(n)) where p is polynomial. When p is linear, the algorithm is called of linear complexity.*

Definition 6 *An algorithm is said to be **pseudo-polynomial** time if its complexity function ξ has the form of a polynomial function. The execution time depends not only on the length of the inputs of the problem but also on the size of them.*

Definition 7 *An algorithm is said in **exponential** time if its complexity function ξ is O(n!) or $O(k^n)$, $k > 1$ or else $O(n^{log(n)})$. Polynomial algorithms are of course the most interesting because they lead to the solution of a scheduling problem in a reasonable considered time, unlike the exponential complexity. Note again that, more the algorithmic complexity is high, more their overhead implementation will be important.*

Polynomial algorithms are of course the most interesting ones because they lead to the implementation of a scheduling algorithm in a reasonable time, unlike the exponential complexity. Note again that, higher is the algorithmic complexity, more important and costly is the overhead incurred by the on line execution of the scheduler.

1.3 Periodic Task Scheduling

The majority of real time schedulers rely on the notion of priority. If the priority is set at the initialization time for all tasks, the algorithm is said to be fixed priority driven. If the priority is not constant over time, the algorithm is said to be dynamic priority driven. The study reported in this thesis only deals with preemptive dynamic priority scheduling, and does not consider resource and precedence constraints. We now recall the famous EDF (Earliest Deadline First) scheduling algorithm with a description of its behaviour with its associated performance and feasibility conditions [22].

1.3.1 The EDF scheduling algorithm

By definition, at each instant, the EDF algorithm gives the highest priority to the task with the closest absolute deadline d_i [23, 24, 22]. In case of conflicts, the task with the earliest arrival time may be the first for execution.

Example 1 *Principles of the EDF algorithm are illustrated with a set of tasks* $\Gamma = \{\tau_i(C_i, D_i, T_i), i = 1 \text{ to } 2\}$. *Let* $\tau_1 = (4, 9, 9)$ *and* $\tau_2 = (3, 12, 12)$.
The EDF *schedule produced on* Γ *during the first hyperperiod is depicted in Figure 1.4:*

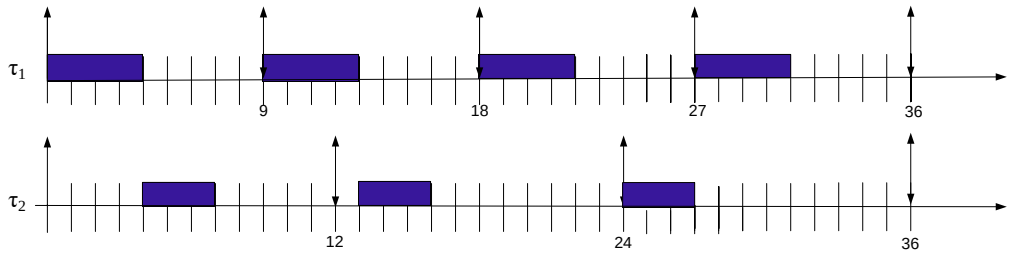


Figure 1.4: Illustration of EDF schedule

1.3.1.1 Optimality Results

EDF has been proved to be optimal for preemtively scheduling independent hard deadline tasks [25]. In particular, that signifies that if a set of independent periodic tasks is schedulable by any algorithm then it is also schedulable by EDF.

In the non-preemptive case, the scheduling problem is known to be $\mathcal{NP} - hard$ [26]. However, if we consider only non-idling scheduling, the problem is again solvable. In this sub-class of non-preemptive schedulers, EDF algorithm is optimal as it is shown in George et al. [27].

1.3.1.2 Schedulability Conditions

A schedulability test, based on a necessary and sufficient condition [22], can be reported for the EDF algorithm:

Theorem 1 *A periodic task system* Γ *such that* $\forall i = \{1 \cdots n\},\ D_i = T_i$ *is schedulable by* EDF *if and only if the utilization factor* U *satisfies (see [22]):*

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1.1}$$

Later, Dertouzos [25] proved that EDF is optimal among all preemptive scheduling (periodic, sporadic and aperiodic task systems). This means that if a task set is not schedulable by EDF, then it cannot be scheduled by any other algorithm.

Under EDF, the schedulability analysis of periodic tasks with deadlines less than or equal to periods can also be conducted using the processor demand criterion proposed by Baruah et al. [28].

Theorem 2 *A task set is schedulable by EDF if and only if, in every interval of length L, the overall computational demand is not greater than the available processing time, that is, if and only if $U \leq 1$ and*

$$\forall L > 0, \sum_{i=1}^{n} \lfloor \frac{L + T_i - D_i}{T_i} \rfloor C_i \leq L \tag{1.2}$$

Where $\lfloor x \rfloor$ denotes the floor of a rational number, that is, the highest integer less than or equal to x (see [28]).

The complexity of this feasibility test is *pseudo-polynomial*. However, in some restrictive hypotheses, the same authors show that the schedulability analysis under EDF has a complexity in $O(n)$ [28].

1.4 Aperiodic Task Scheduling

Real-time scheduling algorithms that deal with a combination of mixed sets of periodic real-time tasks and aperiodic tasks have been extensively studied in the literature, both under fixed and dynamic priority assignments for about 30 years. By definition, an aperiodic task server is optimal if it minimizes the response times of aperiodic tasks while guaranteeing that the deadlines of the periodic tasks are met. Several important approaches for servicing aperiodic tasks are discussed in what follows.

1.4.1 Background Scheduling

A Background (BG) Server executes the aperiodic tasks at the lowest priority level. In other terms, it makes use of any extra CPU cycles for aperiodic servicing. Any Background Server executes whenever the processor is idle (i.e. not executing any periodic tasks and no periodic tasks are pending). Consequently, the schedule produced on periodic tasks is identical with and without aperiodic tasks. We notice that when the processor utilization of the periodic task set is high, the processing utilization left for aperiodic servicing is low. With background servicing, this will result in high response times for the aperiodic tasks because opportunities are relatively infrequent and the periodic schedule is not flexible [9].

Example 2 *The illustrative example of Figure 1.5 shows the schedule produced by the BG server. We consider a task set composed of two periodic tasks and two aperiodic tasks as imparted in Tables 1.1 and 1.2, respectively. The periodic tasks are scheduled according to the EDF algorithm.*

Table 1.1: Parameters of periodic tasks

Task	C_i	D_i	T_i
τ_1	4	9	9
τ_2	3	12	12

Table 1.2: Parameters of aperiodic tasks

Task	a_i	c_i
Ap_1	9	1
Ap_2	18	3

The response times of Ap_1 and Ap_2, offered through the Background servicing mechanism, are 8 and 14 time units respectively, which do not reveal a good performance for this servicing approach.

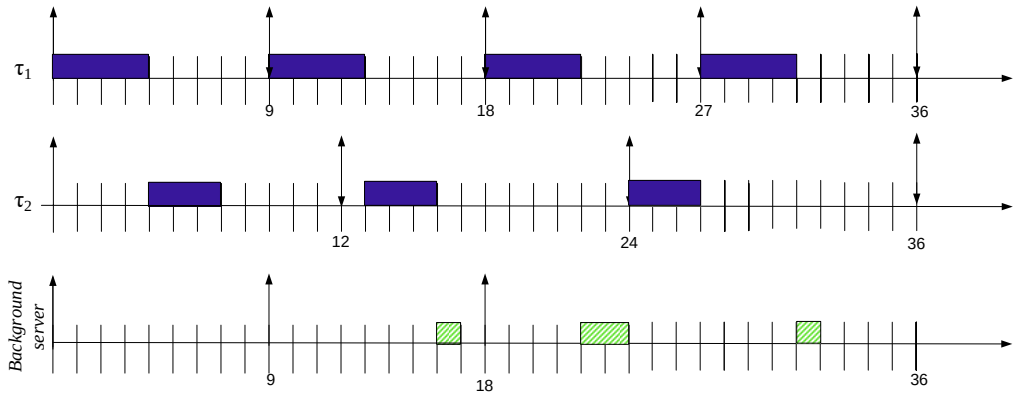


Figure 1.5: Illustration of the Background Server

1.4.2 Dynamic priority servers

Most of the methods studied for fixed priority algorithms have been extended to dynamic priority algorithms. Several approaches have been proposed under EDF by Ghazalie and Baker [29] (Deadline Deferrable Server, Deadline Sporadic Server, and Deadline Exchange Server), by Spuri and Buttazzo [13, 14] (Dynamic Sporadic Server, Dynamic Priority Exchange Server, and Improved Priority Exchange Server), by Gutiérrez and al. [30] (Minimum Deadline Assignment Server), and by Butazzo and Sensini [15] (Improved Total Bandwidth Server)

We will not describe their behavior in this part. We shall consider the algorithms studied in the framework of the thesis: EDL [10], TBS [13, 14], and TB* [31], taking into account their basic properties and their optimality.

1.4.2.1 EDL Server

Using the available slack of periodic tasks for advancing the execution of aperiodic requests is the basic principle adopted by the EDL server [13, 14]. This aperiodic servicing algorithm can be viewed as a dynamic version of the Slack Stealing algorithm described in [11]. The definition of the Earliest Deadline as Late as possible (EDL) server makes use of some results presented by Chetto and Chetto [10] to determine the location and length of idle time in any window of a sequence generated by the two different implementations of EDF: EDS and EDL.

Under EDS the active tasks are processed as soon as possible, whereas they are processed as late as possible under EDL. An important property of EDL is that in any interval $[0, t]$ it guarantees the maximum available idle time.

Let us present the terminology used by the authors in [10]. f_Y^X is the availability function defined for a task set Y and a scheduling algorithm X.

$$f_Y^X(t) = \begin{cases} 1, & \text{if the processor is idle at t} \\ 0, & \text{else} \end{cases} \tag{1.3}$$

where $f_Y^X(t)$ is defined with respect to a task set Y that is scheduled according to the scheduling algorithm X in the time interval $[0, t]$. So, for two instants t_1 and t_2, the integral $\int_{t_1}^{t_2} f_Y^X(t)dt$ gives the total idle time available in the interval $[t_1, t_2]$. Denote this quantity as $\Omega_Y^X(t_1, t_2)$.

1.4.2.1.1 Static Idle Times under EDL The function f^{EDL} computes the static EDL schedule *off-line* for the task set Γ. In this case, we have to estimate the localization and duration of idle times within the EDL schedule from time $t = 0$ till the end of the hyperperiod. Let $H = lcm(T_1, T_2, \cdots, T_n)$, the

hyperperiod be equal to the least common multiple of the task periods. The EDL schedule for the interval $[0, H]$ can be described by means of the two following vectors:

– *Static Deadline Vector* $\mathcal{K}$: The static deadline vector $\mathcal{K} = \{k_0, k_1, \cdots, k_i, k_{i+1}, \cdots, k_q\}$ represents the time instants from 0 to the end of the first hyperperiod at which idle times occur and is constructed from the distinct deadlines of tasks. We note that $q \leq N+1$ where N denotes the number of instances within $[0, H]$. Consequently, instances k_i of vector $\mathcal{K}$ are defined as follows:

$$k_i = x.d_j \tag{1.4}$$

Where $x = \{1, \cdots, \frac{H}{T_j}\}$, $k_i < k_{i+1}$, $k_0 = 0$ and $k_q = H - min\{T_j; 1 \leq i \leq n\}$

– *Static Idle Time Vector* $\mathcal{D}$*:* it represents the lengths of the idle times which start at time instants given by $\mathcal{K}$. $\mathcal{D} = \{\Delta_0, \Delta_1, \cdots, \Delta_i, \Delta_{i+1}, \cdots, \Delta_q\}$.
The recurrence formula for the calculation of vector $\mathcal{D}$ is:
The duration of idle time of vector $\mathcal{D} = \{\Delta_0, \Delta_1, \cdots, \Delta_i, \Delta_{i+1}, \cdots, \Delta_q\}$ calculated on $[0, H[$ is defined as:

$$\Delta_q = \min\{T_i | 1 \leq i \leq n\} \tag{1.5}$$

$$\Delta_i = \max(0, F_i), \ \ for\ i = q-1\ down\ to\ 0 \tag{1.6}$$

where

$$F_i = (H - k_i) - \sum_{j=1}^{n} \lceil \frac{H - k_i}{T_j} \rceil C_j - \sum_{k=i+1}^{q} \Delta_k \tag{1.7}$$

The complexity for computing the EDL static schedule is $O(N)$ where N is the total number of periodic instances in the hyperperiod [18].

Example 3 *Let us consider the previous task set* Γ*. By applying the above formulas, we obtain:* $K = (0, 9, 12, 18, 24, 27)$ *and* $D = (5, 0, 2, 2, 0, 2)$.

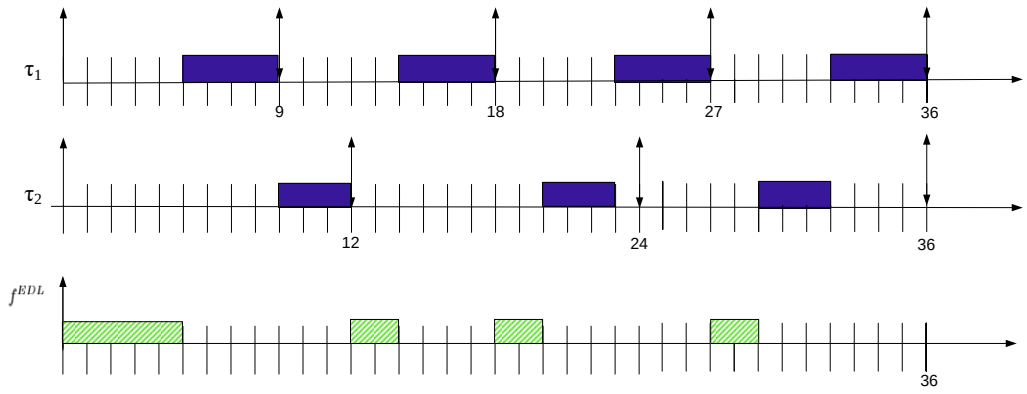


Figure 1.6: Illustration of the static EDL Schedule

1.4.2.1.2 Dynamic Idle Times under EDL

– *Dynamic Deadline Vector* $\mathcal{K}(t)$: We denote by t the current time at which we want to apply the EDL scheduling algorithm. We must compute the dynamic deadline vector $\mathcal{K}(t)$ which represents the time instants that are greater than or equal to t in the current hyperperiod, at which idle times occur. As in the static case, it is constructed from the distinct deadlines of periodic tasks.

Let h be an index such that $k_h = sup\{d, d\epsilon\mathcal{K}\ and\ d < t\}$. So, this vector contains the instances such that $\mathcal{K}(t) = (t, k_{h+1}, \cdots, k_i, \cdots, k_q)$.

– *Dynamic Idle Time Vector* $\mathcal{D}(t)$: The dynamic idle time vector $\mathcal{D}(t) = (\Delta_h(t), \Delta_{h+1}(t), \cdots, \Delta_i(t), \cdots, \Delta_q(t))$ represents the length of the idle times that starts at time instants given by $\mathcal{K}(t)$. $\Delta_i(t)$ denotes the length of idle time that follows time k_i for $h < i \leq q$.
At time t, each periodic task τ_i is characterized by $A_j(t)$ which denotes the amount of processing completed on the current request of τ_i. Let M be the greatest deadline of the current periodic request, which means the greatest deadline among the ready task instances. Let index f be such that $k_f = \min\{k_i \mid k_i > M\}$ [10].

The dynamic idle time vector $\mathcal{D}(t) = (\Delta_h(t), \Delta_{h+1}(t), \cdots, \Delta_i(t), \cdots, \Delta_q(t))$ is defined as:

$$\Delta_i(t) = \Delta_i \quad for\ i = q\ down\ to\ f \tag{1.8}$$

$$\Delta_i(t) = \max(0, F_i(t)) \quad for\ i = f - 1\ down\ to\ h + 1 \tag{1.9}$$

with

$$F_i(t) = (H - k_i) - \sum_{j=1}^{n} \lceil \frac{H - k_i}{T_j} \rceil C_j + \sum_{j=1(d_j > k_i)}^{n} A_j(t) - \sum_{k=i+1}^{q} \Delta_k(t) \tag{1.10}$$

$$\Delta_h(t) = (H - \tau) - \sum_{j=1}^{n} \lceil \frac{H - t}{T_j} \rceil C_j + \sum_{j=1(d_j > k_i)}^{n} A_j(t) - \sum_{k=h+1}^{q} \Delta_k(t) \tag{1.11}$$

The overall time complexity of dynamic EDL schedule is $O(K.n)$ and *K* is equal to $\lceil \frac{R}{p} \rceil$, where *n* is the number of periodic tasks, R is the longest deadline and p is the shortest period [18].

In summary, the EDL server consists in executing the periodic tasks as soon as possible whenever there are no pending aperiodic tasks then maximizing the processing time which could be available for processing aperiodic tasks in the future. Whenever at least one aperiodic task is present in the system, the dynamic EDL schedule is computed so as to determine the precise time intervals where to execute the periodic tasks and the other ones where to execute the aperiodic ones.

The scheduling outline of the EDL server can be described by the pseudo-code (Algorithm 1):

Example 4 *Let us assume that the previous periodic task set has been scheduled according to* EDF *(i.e. EDS) from time zero until time* $t = 9$*. Suppose that an aperiodic request occurs at this time. We want to execute the aperiodic tasks as soon as possible and consequently the periodic ones as late as possible. This requires that at run time, at the arrival of the aperiodic task, at* $t = 9$*, we compute the dynamic idle time vector in order to know when to execute the periodic tasks and when to execute the aperiodic ones.*

So we wish to calculate the maximum CPU time which is free in the interval $[9, 36]$. Thanks to equations 1.8, 1.9, 1.10 and 1.11, we get $\mathcal{K}(t) = (9, 12, 18, 24, 27)$, and $\mathcal{D}(t) = (3, 2, 2, 0, 2)$. This is illustrated in Figure 1.7:

In this example, the idle times of an EDL schedule recomputed at $t = 9$, as shown in Figure 1.7, permit to provide an optimal response time for the aperiodic request. We observe in Figure 1.8 that two aperiodic tasks which are depicted in Table 1.2 arrive into the system at two different times. We can see that if there is no aperiodic tasks in the system, the periodic tasks would be scheduled according to EDS. Upon the arrival

Algorithm 1 EDL schedule

Require:
t: current time
$\Gamma(t)$: list of ready periodic tasks
$Ap(t)$: list of ready aperiodic tasks
INITIALISATION($\Gamma(t)$, $\mathcal{D}$, $\mathcal{K}$) {Calculation of the static idle vector $\mathcal{D}$ and of static deadline vector $\mathcal{K}$ from $\Gamma(t)$}
while True **do**
  if $Ap(t)$ is not empty **then**
    Update the dynamic vectors $\mathcal{K}$ and $\mathcal{D}$ in order to schedule the periodic tasks of $\Gamma(t)$ in the EDL busy periods
    if SlackTime(t)>0 **then**
      Schedule $Ap(t)$ according to FCFS
    else
      Schedule $\Gamma(t)$ according to EDF
    end if
  else
    Schedule $\Gamma(t)$ according to EDF
  end if
  $t := t + 1$
end while

of Ap_1 for example, the two dynamic vectors are updated as described above. As the slack time at time $t = 9$ is equal to 3, Ap_1 is serviced immediately within one unit of time. At that time, the highest periodic task is serviced. The maximum delay invoked in terms of periodic activities gives the optimal aperiodic responsiveness; 1 and 3 units of time for Ap_1 and Ap_2, respectively.

We can conclude that the EDF scheduler can be used with high flexibility by executing tasks as soon as possible (EDS) or as late as possible (EDL), according to the deadlines. We shall see in a next chapter how to use such flexibility to execute the tasks neither strictly ASAP or ALAP, but depending on energy availability.

1.4.2.1.3 EDL **Server Properties** Applying the EDL scheduling rule to the periodic tasks at a given time instant leads to maximize the available processing time from this time instant. This is precisely stated in the following Theorem 3.

Theorem 3 *[18] Let X be any on-line preemptive algorithm. At any time t,*

$$\Omega_{\mathcal{T}}^{X}(0,t) \leq \Omega_{\mathcal{T}}^{EDL}(0,t) \tag{1.12}$$

The optimality of the EDL server is a direct consequence of the previous theorem. Whenever there are soft aperiodic tasks pending for execution, the periodic tasks are executed as late as possible while guaranteeing their schedulabilty with providing a minimal response time for every aperiodic task. The optimality of the EDL server is stated in the following Corollary.

Corollary 1 *[18] The EDL server minimizes the response time of every soft aperiodic task.*

We now present another well known aperiodic server which also provides optimal responsiveness.

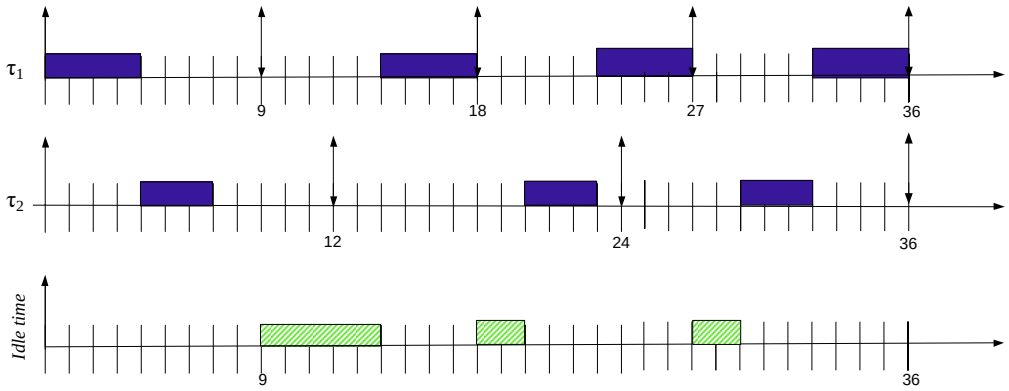


Figure 1.7: New EDL idle times at $t = 9$.

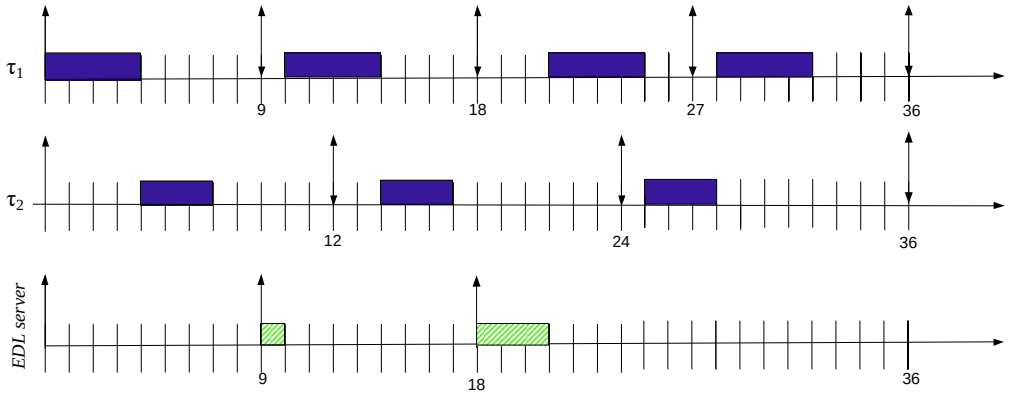


Figure 1.8: Schedule produced with EDL server at $t = 9$.

1.4.2.2 TBS Server

The Total Bandwith Server (TBS) [13], [14] is a simple mechanism for servicing aperiodic tasks in conjunction with periodic tasks in a dynamic priority environment using EDF. Whenever an aperiodic request enters the system, the TBS server assigns to it a deadline according to the time bandwidth reserved for the server (i.e. the available CPU capacity). When the kth aperiodic request arrives at time $t = r_k$, a fictive deadline d_k is assigned to it as follows:

$$d_k = max(r_k, d_{k-1}) + \frac{c_k}{U_{ps}} \tag{1.13}$$

where c_k is the execution time of the newly occurring aperiodic task and U_{ps} is the server utilization factor (i.e. its time bandwidth). By definition, $d_0 = 0$. Moreover, when a new deadline d_k is assigned, the bandwidth already allocated to the previous requests is taken into account by the value d_{k-1}. Once this fictive deadline is assigned to the aperiodic request, it is scheduled jointly with the periodic tasks according to the EDF algorithm.

1.4.2.2.1 Schedulability Plainly, the assignment must be done in such a way that the overall processor utilization of the aperiodic requests never exceeds U_{ps}. Thus, the schedulability of a periodic task set in the presence of a TBS server can be tested by verifying condition of theorem 4 that is formally proved in [14]. But let us first prove that the periodic processor utilization does not exceed U_{ps}.

Lemma 1 *In each interval of time $[t_1, t_2]$, if C_{ape} is the total execution time demanded by aperiodic tasks arrived at t_1 or later and served with deadlines less than or equal to t_2, then*

$$C_{ape} \leq (t_2 - t_1)U_{ps} \tag{1.14}$$

Proof:
By definition,

$$C_{ape} = \sum_{t_1 \leq r_k, d_k \leq t_2} C_k.$$

Given the deadline assignment of the TB server, there must be two indexes k_1 and k_2 such that

$$\sum_{t_1 \leq r_k, d_k \leq t_2} C_k = \sum_{k=k_1}^{k_2} C_k.$$

It follows that

$$\begin{aligned} C_{ape} &= \sum_{k=k_1}^{k_2} C_k \\ &= U_{ps} \sum_{k=k_1}^{k_2} [d_k - max(r_k, d_{k-1})] \\ &\leq U_{ps}[d_{k_2} - max(r_{k_1}, d_{k_1})] \\ &\leq U_{ps}(t_2 - t_1). \end{aligned}$$

□

Theorem 4 *[14] Given a set of n periodic tasks with processor utilization U_{pp} and a set of aperiodic tasks served by TBS with processor utilization U_{ps}, the whole set is schedulable if and only if*

$$U_{pp} + U_{ps} \leq 1. \tag{1.15}$$

Proof: "If". Suppose there is an overflow at time t. The overflow is preceded by a period of continuous utilization of the processor. Furthermore, from a certain point $t^{'}$ on, only jobs (periodic or aperiodic) ready at $t^{'}$ or later and having deadlines less than or equal to t are running. Let C be the total execution time demanded by these jobs. Since there is an overflow at time t, we must have

$$t - t^{'} < C.$$

We also know that

$$\begin{aligned} C &\leq \sum_{i=1}^{n} \lfloor \frac{t - t^{'}}{T_i} \rfloor C_i + C_{ape} \\ &\leq \sum_{i=1}^{n} \lfloor \frac{t - t^{'}}{T_i} \rfloor C_i + (t - t^{'}) U_{ps} \\ &\leq (t - t^{'})(U_{pp} + U_{ps}). \end{aligned}$$

It follows that

$$U_{pp} + U_{ps} \leq 1,$$

a contradiction.

"Only If". If an aperiodic request enters the system periodically, say each $T_s > 0$ units of time, and has execution time $C_s = T_s U_{ps}$, the server behaves exactly as a periodic task with period T_s and execution time C_s. Being the processor utilization $U = U_{pp} + U_{ps}$, again from Theorem 7 of [22] we can conclude that $U_{pp} + U_{ps} \leq 1$. □

Example 5 *The following example shows two schedules produced by the EDF algorithm, using the deadline assignment by the TBS server as shown in Figure 1.9. Let us consider the same task set of two periodic tasks and two aperiodic tasks imparted in Tables 1.1 and 1.2 respectively. The periodic tasks are scheduled according to the EDF algorithm. It is worth noting that the periodic utilization is $U_{pp} = 0.7$ leaving a low bandwidth for the aperiodic tasks. The condition $U_{pp} + U_{ps} \leq 1$ is verified, hence the system is considered feasible.*

When the aperiodic task Ap_1 arrives at time 9, it receives a fictive deadline computed by equation 1.13 ($d_1 = 13$) according to the TBS server. As 13 is the earliest deadline, Ap_1 is immediately serviced. A second aperiodic task Ap_2 arrives at time 18. Identically, it receives a fictive deadline ($d_2 = 28$), is executed at time 22 as τ_1 with nearest deadline, and is is active at this point. We note through the resulting schedule that the response time of Ap_1 and Ap_2 is 1 and 7 units of time respectively, which shows a moderate performance with respect to the low bandwidth dedicated to aperiodic tasks.

1.4.2.2.2 Implementation Complexity Among the servers presented in the literature, the TB server is considered to be the simplest one. We only need to keep track of the deadline assigned to the last occurring aperiodic task ($d_k - 1$) to properly assign the deadline to the new occurring task. Therefore, the task can be inserted in the ready queue and processed by EDF as any other periodic job. Consequently, the overhead is practically negligible.

Besides its simplicity, TBS was extended towards optimality to reduce the aperiodic response time. In the following section, we will briefly recall the optimal algorithm TB^*.

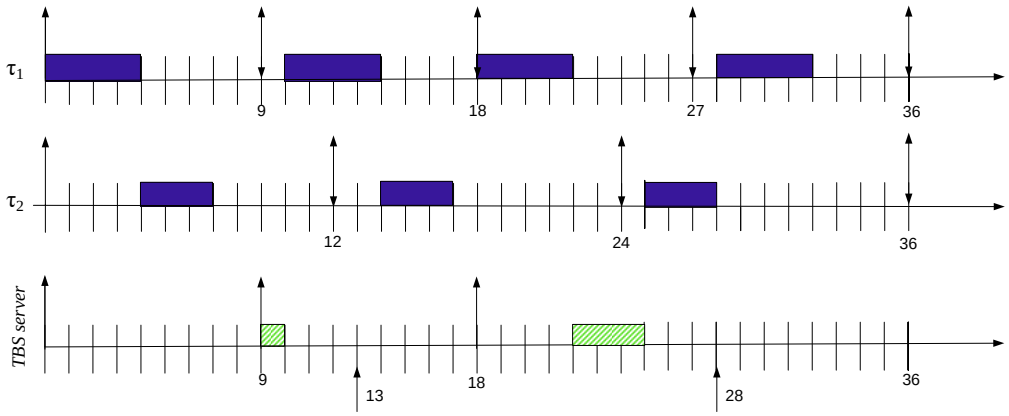


Figure 1.9: Illustration of the TBS server

1.4.2.3 TB^* Server

The Total Bandwidth Server algorithm TBS is capable of achieving good aperiodic responsiveness with high simplicity. However, its performance is perfectible. How? If we look at the example in Figure 1.9, we can argue that, if we assign a shorter deadline, the second aperiodic task Ap_2 may be better served without jeopardizing the schedulability of periodic tasks, especially that the periodic jobs have enough laxity to be preempted.
Thus, the optimal Total Bandwidth Server algorithm, namely TB^* [15] is an optimization of the TB server in the sense that it assigns to each aperiodic request a shorter fictive deadline than that provided by TBS. Thus, whenever an aperiodic task arrives, TB^* first assigns to it a deadline according to the TBS algorithm. Then, it will try to shorten this deadline to the maximum so as to improve the response time of the aperiodic requests without compromising the execution of the periodic tasks. If d_k^0 corresponds to the first deadline assigned by TBS, the new deadline d_k^1 will be set at the estimated worst-case finishing time of the aperiodic task f_k^0, scheduled with d_k^0. The process of shortening this deadline is applied iteratively until no improvement is possible, while guaranteeing the schedulability of the periodic task set.
Consequently, if d_k^s is the deadline assigned to the aperiodic request Ap_k at step s, the schedulability is guaranteed by assigning to Ap_k a new deadline given by:

$$f_k^s = d_k^{s+1} = t + c_k + I_p(t + d_k^s) \tag{1.16}$$

where t is the current time (corresponding to the maximum between the release r_k of the request Ap_k and the completion time of the previous request), c_k is the worst case execution time required by Ap_k, and $I_p(t, d_k^s)$ is the interference due to periodic jobs in the interval $[t, d_k)$.
The periodic interference $I_p(t, d_k^s)$ is given by the sum of two terms, $I_a(t, d_k^s)$ and $I_f(t, d_k^s)$. $I_a(t, d_k^s)$ corresponds to the interference due to the periodic jobs which are active at the current time with deadlines strictly less than d_k^s. $I_f(t, d_k^s)$ corresponds to the future interference due to the periodic jobs having their arrival date greater than the current time t with deadlines less than d_k^s.
Formulas are given below:

$$I_a(t, d_k^s) = \sum_{\tau_{active}; d_i < d_k^s} c_i(t) \tag{1.17}$$

and

$$I_f(t, d_k^s) = \sum_{i=1}^{n} max(0, \lceil \frac{d_k^s - next_{r_i}(t)}{T_i} \rceil) C_i \tag{1.18}$$

where $next_{r_i}(t)$ is the next arrival of task τ_i greater than or equal to t.

1.4.2.3.1 TB^* **Server Properties** If the actual execution time of tasks is equal to their worst-case execution time, the TB* deadline assignment algorithm will achieve optimality. In that case, it minimizes the response time of each aperiodic task among all scheduling algorithms which meet all periodic task deadlines, assuming that aperiodic requests are processed in FIFO order, and that deadlines are broken in favour of aperiodic tasks. This result is summarized in the following theorem.

Theorem 5 *[31] For any periodic task set and any aperiodic arrival stream processed in FIFO order, the TB* algorithm is optimal because it minimizes the response time of every aperiodic task among all scheduling algorithms which meet all periodic task deadlines.*

Example 6 *Let us assume the same task set* Γ *of Table 1.1. The periodic utilization ratio is* $U_{pp} = 0.7$. *The bandwidth allocated to the TB* server is such that* $U_{ps} = 1 - U_{pp} = 0.3$. *Condition* $U_{pp} + U_{ps} \leq 1$ *is verified, hence the system is considered feasible. An aperiodic task* Ap_1 *occurs at* $t = 9$. *It receives a fictive deadline* $d_1 = 13$ *(optimal) according to the TBS algorithm. A second aperiodic task* Ap_2 *occurs at* $t = 18$. *It receives a fictive deadline (*$d_2 = 28$*) according to the TBS algorithm. Using equations 1.17 and 1.18, we get:* $I_a(18, 28) = c_1(18) = 4$ *and* $I_f(18, 28) = 0$.
By equation 1.16, we obtain $d_2^1 = t + c_1 + I_a + I_f = 25$. *We repeat the process until the deadline coincides with the finish time. In our example, only two steps are performed in order to shorten the first deadline assigned by TBS. These steps are shown in Table 1.3. The schedule produced by EDF using the shortest deadline* $d_2^* = d_2^1 = 25$ *is shown in Figure 1.10.*

Table 1.3: Deadlines and finishing times computed by the TB* algorithm

step	d_k^s	f_k^s
0	28	25
1	25	25

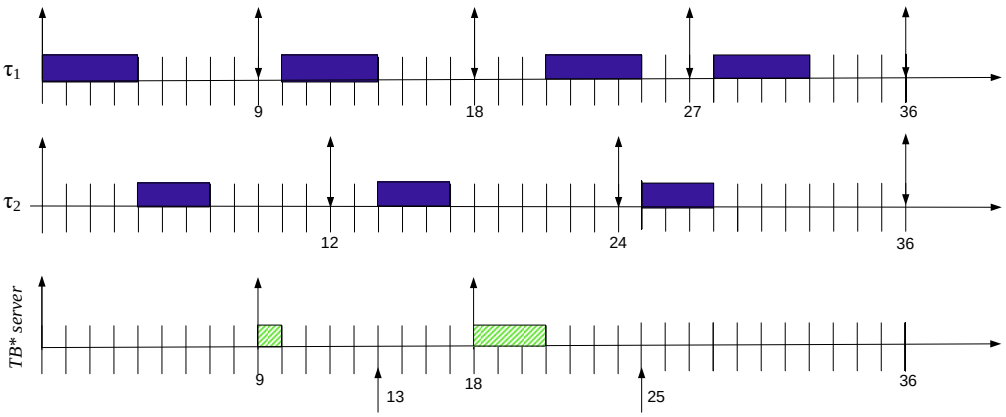


Figure 1.10: Illustration of the TB* server

A significant result of the TB* deadline assignment rule is that I_a and I_f can be computed in $O(n)$. The overall complexity of the TB^* server is $O(N.n)$, where N is the maximum number of steps that have to be done by the algorithm to shorten the initial deadline assigned by TBS[31]. So balancing performance vs complexity is the main feature of the TB* server, which allows it to be adapted to different application requirements.

1.5 Conclusion

An introduction to scheduling in real time systems was presented in this chapter. Terminology and concepts related to the real-time domain were first addressed. Secondly, we presented the EDF scheduling algorithm for optimally scheduling periodic tasks, noting in particular the conditions of feasibility. Next, we examined the state of the art in terms of aperiodic task servicing. We have presented the basic Background server due to its very simple implementation and despite its inefficiency. And the two optimal dynamic priority servers were described: first, the EDL server based on the slack stealing mechanism and second, the TB* server based on the adequate attribution of a deadline to each aperiodic task.

Although EDL and TB servers are difficult to implement, they present the best achievable response time for soft aperiodic tasks which make them good candidates for scheduling mixed task sets. Our objective in a next chapter will be to show how to modify these servers so as to use them in the energy harvesting context.

In the next chapter, we will focus on a state of the art on energy harvesting techniques that may be applied to design autonomous real-time systems which draw their energy from environmental sources.

CHAPTER 2

RENEWABLE ENERGY FOR COMPUTING SYSTEMS

Summary

In this chapter, we set the stage for our study of energy harvesting technology. We start by describing the architecture of a EHWSN (Energy Harvesting-based Wireless Sensor Network) node. Then, we review the main harvesting techniques to draw and convert energy from ambient sources in order to power wireless devices. In the following sections, we describe the common storing devices used to buffer these devices and we have a look on harvesters found in the market.

Contents

2.1 Introduction

Recently, Internet of Things (IoT) and machine to machine (M2M) have been the focus of the attention of consumers and markets. IoT and M2M are not only adduced to cellular phones and personal computers connected through the Internet, but also to the wireless interconnection of all of the billions of "things" such as sensors through the Internet or local area networks.

Realizing that supplying the power needed to maintain all these sensors operating for their expected lifespan was a linchpin that could potentially short-circuit the IoT, scavenging energy from the IoT node's environment comes to offer a straightforward solution for easily powering those remote devices using clean green energy. The umbrella term for the group of harvesting technologies is called Energy Harvesting (EH). Energy harvesting techniques use power generating elements to convert light (solar), heat (thermoelectric), vibration (piezoelectric), or RF energy (such as energy emitted from cellphone towers) into electricity in a stable way and without a lot of loss. Energy harvesting allows the design of systems that are able to function for years on these ambient power sources, getting rid of the battery-change problem.

That does not mean that there is no need for a battery in these systems. When the ambient energy is collected, it must be temporarily stored to provide the required current at a time when: 1) it is needed (IoT nodes have low duty cycles, which is a sensor taking periodic air temperature samples that might only be active a few milliseconds per hour, and can be in sleep mode the remaining time) or 2) when the source of energy is not available (e.g., no sun's rays at night). Classical rechargeable coin cell batteries can be used in this manner and so can thin film rechargeable solid-state batteries as well as supercapacitors.[32]

A key consideration that affects power management in an energy harvesting system is that instead of minimizing the energy consumption and maximizing the lifetime achieved, as in classical energy storage operated devices, the system operates in a so-called energy neutral mode by consuming only as much energy as harvested.

2.2 Wireless Sensor Networks

Wireless sensor networks (WSNs) are typically embedded real-time systems. They are composed of a collection of sensor nodes designed to perform a common duty and implement tasks in charge of sensing, processing, and communicating data. Today, most of sensor nodes are powered by limited-energy sources, typically small batteries. This makes energy efficiency a vital criterion in the development of WSNs. So, the main emphasis has been placed on prolonging the lifetime of WSNs.
Any sensor node is equipped with four components as depicted in Figure 2.1: 1) a sensing unit to capture data such as temperature, pressure, humidity, etc. 2) a micro-processor (or micro-controller) to process the data; 3) a transceiver for transmitting data and 4) an energy storage unit to supply energy to all the components. It is well-understood that data communication in a sensor node consumes much more energy than data computing.

The major sources of existing environmental energy for WSNs are solar, wind, vibration and thermal. In order to enable sensor nodes to take advantage from EH technology, a new type of sensor node equipped with an EH unit has been developed to perpetuate the lifetime of WSNs [33].

Figure 2.2 presents the system architecture of a wireless sensor node which is composed of the following components: 1) The energy harvester(s), responsible of converting external ambient or human-generated energy to electricity; 2) a power management unit, that draws electrical energy from the harvester. Then the energy is stored or delivered to the other system components for immediate usage; 3) energy storage, for conserving the harvested energy for future usage; 4) a microcontroller that consumes the energy due to processing activities; 5) a radio transceiver which consumes energy by transmitting and/or receiving data; 6) sensing equipment; 7) A/D converter to digitize analog signals generated by the sensors and makes it available to the microcontroller for additionnal processing, and 8) memory to store data and code.

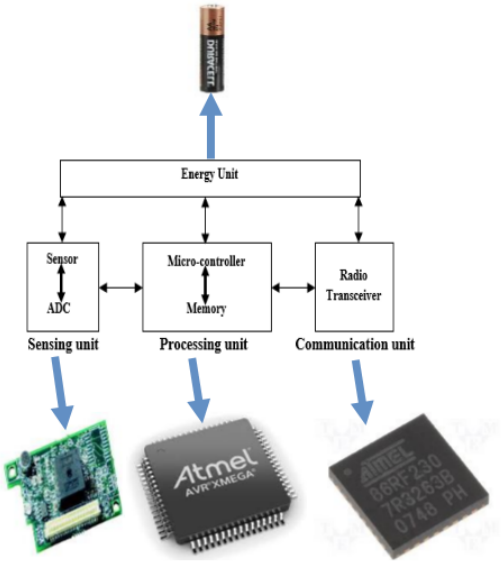


Figure 2.1: Architecture of a wireless sensor node [1].

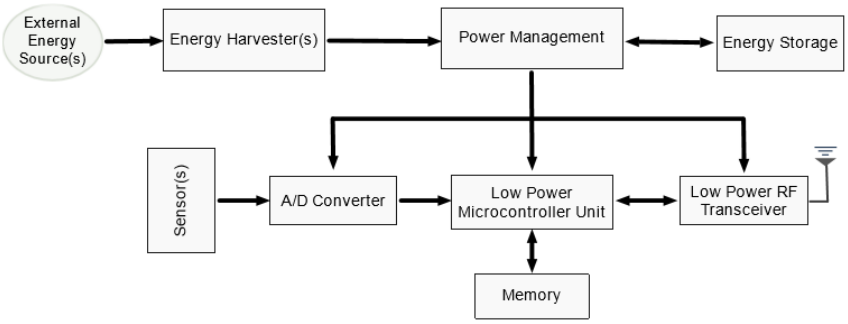


Figure 2.2: An Energy Harvesting sensor node.

2.3 Environmental energy sources

Energy harvesting devices have to capture small amounts of energy over a long time from sources, such as ambient light, wind, vibration, linear motion, temperature differential, radio frequency (RF) energy, etc. We now present the most usual types of energy source and we provide a brief description of them and relevant references. Before going into details, the variety of energy sources are shown in Figure 2.3.

2.3.1 Mechanical energy

Mechanical energy harvesting denotes the technique of converting mechanical energy into electricity by using vibrations, mechanical stress and pressure, strain from the surface of the sensor, high-pressure motors, waste rotational movements, fluid, and force. Converting the energy of the displacements and oscillations of a spring mounted mass component inside the harvester into electrical energy is the main pillar behind mechanical energy harvesting [34, 35].
Mechanical energy harvesting can be: piezoelectric, electrostatic and electromagnetic.

Piezoelectric energy harvesting is based on the piezoelectric effect for which mechanical energy from pressure, force or vibrations is transformed into electrical power by straining a piezoelectric material. The technology of a piezoelectric harvester is usually based on a cantilever structure with a seismic mass attached into a piezoelectric beam. The latter has contacts on both sides of the piezoelectric material [35].

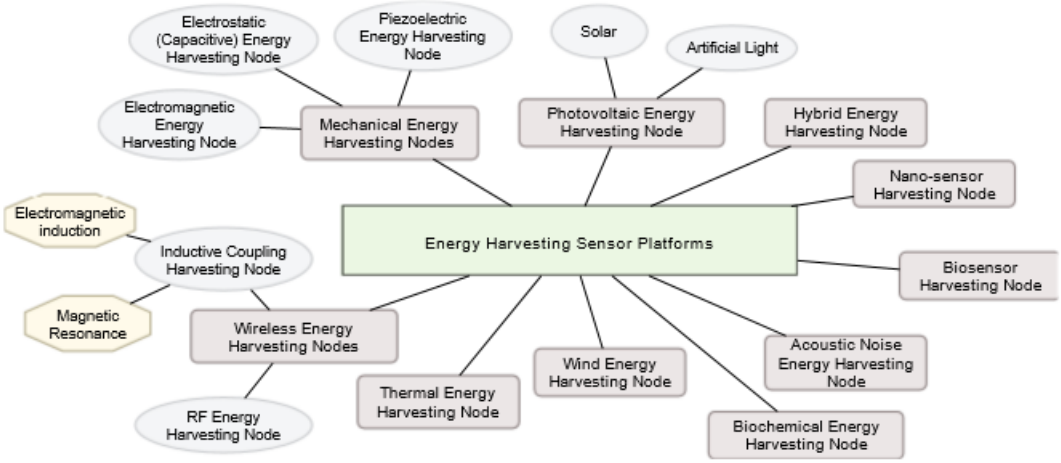


Figure 2.3: Summary of energy harvesting systems [1]

In particular, strains in the piezoelectric material generate charge separation across the harvester, creating an electric field, and hence voltage, proportional to the stress generated [36, 37]. The variation of voltage depends on the strain and time, and an irregular AC signal is generated. Piezoelectric energy conversion has the benefit that it produces the desired voltage directly, without the need for a separate voltage source. However, piezoelectric materials are breakable and can suffer from charge leakage [38][39][35]. Examples of piezoelectric energy harvesters are presented in [40][41][42]

Electrostatic energy harvesting is based on varying the capacitance of a vibration dependent variable capacitor [43][39]. In order to scavenge the mechanical energy, a variable capacitor is created by opposing two plates, one fixed and one moving, and is initially charged. When vibrations separate the plates, mechanical energy is transformed into electrical energy from the capacitance variation. This type of harvesters can be integrated into microelectronic-devices due to their incorporated circuit-compatible nature [44]. However, an additional voltage source is required to initially charge the capacitor [39]. Recent efforts to prototype sensor-size electrostatic energy harvesters are presented in [45][46].

Electromagnetic energy harvesting is based on Faraday's law of electromagnetic induction. An electromagnetic harvester utilizes an inductive spring mass system for converting mechanical energy to electrical one. It induces voltage by moving a mass of magnetic material through a magnetic field created by a stationary magnet. Particularly, vibration of the magnet attached to the spring inside a coil changes the flux and generates an induced voltage [43][35][36]. The benefits of this process don't include mechanical contact between parts nor separate voltage source, which improves the reliability and reduces the mechanical damping in this type of harvesters [38, 39]. However, it is difficult to integrate them in sensor nodes because of the large size of electromagnetic materials [38].
Some examples of electromagnetic energy harvesting systems can be found in [47][48].

Photovoltaic energy harvesting is the technique of converting incoming photons from sources such as solar or artificial light into electricity. Photovoltaic energy can be harnessed by using photovoltaic (PV) cells. These ones consist of two different types of semiconducting materials called n-type and p-type. An electrical field is formed in the area of contact between these two materials. Photovoltaic energy conversion is a traditional, mature, and commercially established energy-harvesting technology. It provides higher power output levels compared to other energy harvesting techniques and is suitable for larger-scale energy harvesting systems. However, its generated power and the system efficiency strongly depend on the availability of light and on environmental conditions. Other factors, including the materials used for

the photovoltaic cell, affect the efficiency and level of power produced by photovoltaic energy harvesters [38][49]. Some recent examples of photovoltaic harvesters are presented in [50][51][52][53].

2.3.2 Thermal energy

Thermal energy harvesting is implemented by *thermoelectric energy harvesting* and *pyroelectric energy harvesting*.
Thermoelectric energy harvesting is the technique of creating electrical energy from temperature difference (thermal gradients) using thermoelectric power generators (TEGs). The main element of a TEG is a thermopile formed by arrays of two dissimilar conductors, i.e., a p-type and n-type semiconductor (thermocouple), placed between a hot and a cold plate and connected in series. A thermoelectric harvester harvests the energy based on the Seebeck effect, which states that electrical voltage is generated when two dissimilar metals joined at two junctions that are kept at different temperatures [54]. This is because the metals respond differently to the temperature difference, creating heat flow through the thermoelectric generator. This generates a voltage difference that is proportional to the temperature difference between the hot and cold plates. Energy is scavenged as long as the temperature difference is maintained.
Pyroelectric energy harvesting is the technique of producing voltage by heating or cooling pyroelectric materials. These materials do not need a temperature gradient similar to a thermocouple. As alternative, they need time-varying temperature modifications. Modifications in temperature change the locations of the atoms in the crystal structure of the pyroelectric material, which generates voltage. To keep producing power, the whole crystal should be continuously affected by temperature change. Otherwise, the produced pyroelectric voltage gradually disappears due to the leakage current [55]. Pyroelectric energy scavenging attains greater performance compared to thermoelectric scavenging. It supports scavenging from high temperature sources, and is much simpler to get to work using limited surface heat exchange. On the other hand, thermoelectric energy harvesting supplies higher scavenged energy levels. The maximum performance of thermal energy harvesting is limited by the Carnot cycle [43]. Because of the various sizes of thermal harvesters, they can be placed on the human body, on structures, and on equipment. Some prototypes of this type of harvesters for WSN nodes are described in [56][57].

2.3.3 Wireless energy

Wireless energy harvesting techniques can be categorized into two main categories: *RF energy harvesting* and *resonant energy harvesting*.
RF energy harvesting is the technique of converting electromagnetic waves into electricity by a rectifying antenna, or rectenna. Energy can be scavenged from either ambient RF power from sources such as radio and television broadcasting, mobile phones, WiFi communications and microwaves, or from EM signals produced at a specific wavelength. Although there is a great number of possible ambient RF power, the energy of existing EM waves are exceedingly low because energy quickly decreases as the signal spreads farther from the source. Therefore, in order to harvest RF energy effectively from existing ambient waves, the harvester must remain close to the RF source. Another possible solution is to use a dedicated RF transmitter to produce more powerful EM signals only for the purpose of providing energy to sensor nodes. Such RF energy harvesting is able to efficiently delivers powers from micro-watts to few milliwatts, depending on the distance between the RF transmitter and the harvester.

Resonant energy harvesting, also known as resonant inductive coupling, is the technique of delivering and harvesting electrical energy between two coils, which are extremely resonant at the same frequency. Particularly, an external inductive transformer device, coupled to a primary coil, can send power through the air to a device provided with a secondary coil. The primary coil generates a time-varying magnetic flux that passes through the secondary coil, inducing a voltage. Generally, there are two possible implementations of resonant inductive coupling: weak inductive coupling and strong inductive coupling. The first one requires

that the distance between the coils must be very small (few centimeters). However, if the receiving coil is quite tuned to match the external powered coil, a "strong coupling" between electromagnetic resonant devices can take place and powering is possible over longer distances. It is worth noting that resonant inductive coupling is considered a wireless energy harvesting technique since the primary and secondary coil are not physically connected. Some recent examples of wireless energy harvesting techniques for WSNs can are presented [58] [59].

2.3.4 Wind energy

Wind energy harvesting is the technique of converting wind energy (air flow) into electrical energy. A wind turbine with proper size is used to profit from linear motion coming from wind for generating electrical energy. Small sized wind turbines, that are eligible for producing enough energy to power WSN nodes, exist [60]. However, effective design of miniature wind energy harvesting is still in progress, challenged by very low flow rates, fluctuations in wind strength, unpredictability of flow sources, etc. Furthermore, even though the performance of large-scale wind turbines is extremely efficient, small scale wind turbines show inferior efficiency due to the relatively high viscous drag on the blades at low Reynolds numbers [61][34].

2.3.5 Biochemical energy

Biochemical energy harvesting is the technique of converting oxygen and endogenous substances into electricity through electrochemical reactions [62][63]. Specifically, biofuel cells behaving as active enzymes and catalysts can be exploited to scavenge the biochemical energy in biofluid into electrical energy. Human body fluids comprise many types of substances that have scavenging potential [64]. Amongst these substances, glucose is the most common used fuel source. As known in theory, it releases 24 free electrons per molecule when oxidized into carbon dioxide and water. Even if biochemical energy harvesting can be better than other energy harvesting processes with regards to continuous power output and biocompatibility [62], its efficiency relies on the type and availability of fuel cells. Advantages and disadvantages using enzymatic fuel cells for energy production engenders advantages and disadvantages are presented in [65]. Research efforts such as [62][63] are examples of recent proposed prototypes that use biochemical energy harvesting to provide energy to microelectronic devices.

2.3.6 Acoustic energy

Acoustic energy harvesting is the technique of converting high and continuous acoustic waves from the environment into electrical energy by using an acoustic transducer or resonator. The acoustic emissions can be in the form of longitudinal, transverse, bending, and hydrostatic waves varying from very low to high frequencies [66]. Typically, acoustic energy harvesting is used where local long term power is not available, as in the case of remote or isolated locations, or where cabling and electrical commutations are difficult to use such as inside sealed or rotating systems [67, 66]. However, the performance of harvested acoustic power is low and such energy can only be harvested in real acoustical environments. Harvested energy from acoustic waves theoretically yields 0.96 $\mu W/cm^3$, which is much lower than what is attainable by other energy harvesting techniques. In principle, ongoing researches are still performed to investigate this kind of harvesters.
Implementations of acoustic energy harvesting systems can be found in [68][69].

All the previously described harvesting techniques can be combined and concurrently used on a single platform (hybrid energy harvesting).

A survey of the quantity of energy which can be harvested from different sources is given in Table 2.1. We show the power density for each energy harvesting technique. The power density gives the harvested energy per unit volume, area, or mass. Common unit measures of power density include watts per square centimeter and watts per cubic centimeter.

Table 2.1, taken from [2], shows power outputs for typical energy harvesting devices.

Table 2.1: Power density of energy scavenging techniques [2]

Energy Harvesting technique	Power density
Photovoltaic	Outdoors (direct sun): 15 mW/cm^3 Outdoors (cloudy day): 0.15 mW/cm^3 Indoors: < 10 mW/cm^3
Thermoelectric	Human: 30 $\mu W/cm^3$ Industrial: 1 to 10 $\mu W/cm^3$
Pyroelectric	8.64 $\mu W/cm^3$ at the temperature rate of $8.5°$ C gradient
Piezoelectric	250 $\mu W/cm^3$ 330 $\mu W/cm^3$ (shoe inserts)
Electromagnetic	Human motion: 1 to 4 $\mu W/cm^3$ Industrial: 306 $\mu W/cm^3$ or 800 $\mu W/cm^3$
Electrostatic	50 to 100 $\mu W/cm^3$
RF	GSM 900/1800 MHz: 0.1 $\mu W/cm^3$ WiFi 2.4 GHz: 0.01 $\mu W/cm^3$
Wind	380 $\mu W/cm^3$ at the speed of 5 m/s
Acoustic (noise)	0.96 $\mu W/cm^3$ at 100 dB 0.003 $\mu W/cm^3$ at 75 dB

In our work, we are mainly concerned by solar energy. There are many reasons for using this type of energy: light is present outdoor as well as indoor which makes it accessible by a lot of wireless sensor nodes, high power output compared to other scavenging technologies, maturity of this harvesting technology and absence of moving parts in the harvester which makes solar panels maintenance-free.

For efficient utilization of energy, we need to buffer it for temporary storing. Characteristics of energy storage technologies are presented in the next section.

2.4 Energy Storage Devices

The production of renewable energies strongly depends on the variable environment conditions (sunshine, etc.). It does not follow the growing trend of demand. Therefore, it seems crucial to store this energy at the time it is harvested for future use. Although it is possible in some particular applications to directly consume the energy obtained by the harvester, with no energy storage (harvest-use architecture [33]), in general it does not meet the constraints of most of applications. Let us recall that the application software that executes in a node is composed of real-time tasks with timing constraints. Such tasks should execute and consume energy in specific time intervals. And this imposes that sufficient energy is available during these time intervals.

A reasonable architecture enables the node to directly use the harvested energy. But it also includes a storage component that acts as an energy buffer for the system, with the main objective of preserving the

harvested energy. Once the harvesting rate is greater than the current usage, the buffer unit stores surplus energy for later use, thus supporting variations in the power level emitted by the environmental source.

For energy storage, there are two widely used alternatives: secondary rechargeable batteries and supercapacitors (or ultracapacitors).

2.4.1 Batteries

Primary (non rechargeable) batteries compared to *secondary* (rechargeable) batteries are relatively long lasting. However, a large-scale adoption would result in important environmental issues. Rechargeable batteries require to be accessed for recharging them which is not always possible.

Batteries are the most widely used energy storage technology for all electronic devices [70]. It is due to the generation of energy-hungry portable devices such as digital cameras, camera phones, PDAs, ... etc. Lithium-ion (Li-Ion) batteries are nowadays the secondary batteries leader of the market for powering portable devices. NiCd batteries market is shrinking and is being replaced by NiMH for environmental reasons.
Since the battery is the most important design choice regarding system lifetime, it is vital to investigate the characteristics of a battery type whenever we need to buffer the harvested energy. Several types of battery technologies exist, each with different strengths and weaknesses. Some have a low discharge rate but may be dangerous to the environment. Other ones can store a large amount of energy but have a high discharge rate.

Several factors must be taken into consideration when operational lifetime is estimated [71, 72]:
- *Ampere-hour:* It is the amount of electric charge carried by a current of $1\ A$ flowing during 1 hour.
- *Capacity (C) or Nominal Capacity (NC):* It is the amount of charge expressed in Ampere-hour that can be delivered by a battery.
- *Charge rate:* A charge or discharge current of a battery is measured in C-rate. A discharge current of $1C$ draws a current equal to the rated capacity. For example, a battery rated at $1000\ mAh$ provides $1000\ mA$ for one hour if discharged at $1C$ rate.
- *Cycle life:* It is the number of cycles that a battery can be charged and discharged. Primary batteries or non-rechargeable batteries have a unitary cycle life whereas secondary batteries are also called rechargeable batteries and have a cycle life greater than one, dependent on the battery chemistry.
- *Nominal voltage:* The nominal voltage, also called average discharge voltage, is defined as the mid-point voltage of the battery voltage range during charge or discharge. For example, a battery with a voltage range of $1.8V$ to $2.8V$ has a nominal voltage of $2.3V$.
- *Self-discharge:* Capacity loss during storage due to the internal leakage.

The most commonly used batteries are LiPo and NiMH due to their high energy density, low self discharge, and high number of recharge cycles. Today, processors have a power consumption in sleep mode that is lower than most batteries' self discharge. This makes it increasingly important to choose efficient battery cells. A summary about the most known battery types is found in table 2.2.

To conduct an efficient selection of the energy storage, it is not sufficient to study the characteristics of the device to be powered. It becomes necessary to investigate how to charge the device. For example, some batteries, such as LiPo and NiMH, have relatively sensitive charging and discharging procedures. On the contrary, NiMH batteries can often be recharged up to 1000 times if managed properly, but the charge current can be several hundreds of milliamps (mA) for several hours, which is not a feasible output from solar panels. Moreover, LiPo batteries have similar requirements on charge current. As a result, we must trickle charging the battery, that means keeping the battery fully charged by charging it constantly with a very small current. It can be achieved even by small solar panels.

Table 2.2: Battery types [3]

Type	Voltage	Energy Density	Specific Energy	Self discharge
Lead-acid	$2.0\ V$	$60-75\ Wh/dm^3$	$30-40\ Wh/kg$	$3-20$ % /month
Nickel Cadmium	$1.2\ V$	$50-150\ Wh/dm^3$	$40-60\ Wh/kg$	10 % /month
Nickel Metal Hydrid	$1.2\ V$	$140-300\ Wh/dm^3$	$30-80\ Wh/kg$	30 % /month
Lithium-Ion	$3.6\ V$	$270\ Wh/dm^3$	$160\ Wh/kg$	5 % /month
Lithium-polymer	$3.7\ V$	$300\ Wh/dm^3$	$130-200\ Wh/kg$	$1-2$ % /month

It is also important to note that some battery types have a severely reduced lifetime if they are recharged incorrectly. To address this issue, rechargeable batteries should be only charged under suitable conditions with the correct charge current. It is also good to fully discharge the battery and then recharge it to prolong its lifetime. The problem is that the system will need a secondary energy storage to be powered when the battery is being cycled. *Supercapacitors* can be used for this purpose.

2.4.2 Supercapacitors

Supercapacitors are similar to common capacitors, but they offer very high capacitance with small size. Compared to rechargeable batteries, they offer various benefits [73]. First of all, when typical lifetimes of an electrochemical battery is less than 1000 cycles, supercapacitors can be recharged and discharged virtually with unlimited number of times [74]. Second, they can be quickly charged using simple charging circuits, thus reducing system complexity. They do not need full-charge or deep discharge protection circuits. They also have higher charging and discharging efficiency than electrochemical batteries [73]. A further advantage is reduction of environmental issues related to battery disposal. Thanks to these characteristics, many platforms with harvesting opportunities use supercapacitors as energy storage, either as the unique storage unit [75] or in combination with batteries [76] [77] [78]. Other systems focus on platforms with only rechargeable batteries [79] [80] [49].

Supercapacitors are rated in units of $1\ F$ and higher. The gravimetric energy density is $1-10\ Wh/kg$. This energy density is lower in comparison to batteries. Supercapacitors provide the energy of approximately one tenth of the NiMH battery. The voltage of the supercapacitors drops linearly from full voltage to zero volts. Consequently, supercapacitors are unable to deliver the full charge to the load. The percentage of charge that is available to be given depends on the supply voltage of the load to power [74].

Supercapacitors are insensitive for over and under-charge, and can be recharged a virtually unlimited number of times. These features make supercapacitors *ideal* to be used especially by sensor nodes equipped with energy scavenging devices. This is because energy-scavenging devices often cannot produce a 100% stable charge current over time.

The major disadvantages with supercapacitor are: the extremely high *self-discharge* rate since it can drain itself in only a few days. They are unable to deliver the full energy stored since the voltage discharge curve is not flat. Supercapacitors have low voltages and low energy density.

Advantages of supercapacitors are: *unlimited cycle life* (not subject to the wear and aging experienced by the electrochemical battery), *low impedance* (enhances pulse current handling by paralleling with an electrochemical battery), *rapid charging* (low impedance supercapacitors charge in seconds), *simple charge methods* (voltage-limiting circuit compensates for self-discharge; no full-charge detection circuit needed) and *cost-effective energy storage* (lower energy density is compensated by a very high cycle count).

2.5 Option beyond ambient energy harvesting

The concept of energy harvesting has been around for over a decade. However, the implementation of ambient energy-powered systems in the real-world environment has been cumbersome, complex and costly. For example, solar energy harvesting is a definite possibility if the panel size is dimensioned appropriately. Also, researchers and vendors are concentrating on many indoor applications where solar energy is not a possible solution. Moreover, solar photovoltaic panels are expensive to manufacture. Nevertheless, there are several companies who are focusing on finding efficient energy scavengers for embedded devices.

The main application sectors are: military (battlefield surveillance, reconnaissance of opposing forces and terrain, battle damage assessment), environmental (tracking the movements of animals, forest fire detection, observation of small size bio-diversity, level of air pollution,...) and health (telemonitoring of human physiological data,...).

New companies such as AdaptiveEnergy, EnOcean, Cymbet and Perpetuum, among others are specialized in energy harvesting embedded systems. Also, big electronics' manufacturers, such as Texas Instrument and Analog Devices Inc, are building microcontrollers, digital signal processors and sensors adapted to the energy harvesting technology. In what follows, we describe typical products based on energy harvesting.

– **Micropelt** [81] offers products powered by thermoelectric and inductive energy harvesting. One of them is MPG-D655 2.4; Thin-film Thermogenerator Chip converts heat energy (temperature difference) into electrical energy. The miniaturized dimensions of the MPGD655 makes the thermogenerator chip ideal for ultracompact equipment. Due to the 288 thermoelectric pairs, the MPG-D655 outputs an open circuit voltage of 80 mV/K, which enables extremely efficient voltage converter solutions (DC-Booster) and operation at very low temperature differences (e.g. 5 degrees Celsius).

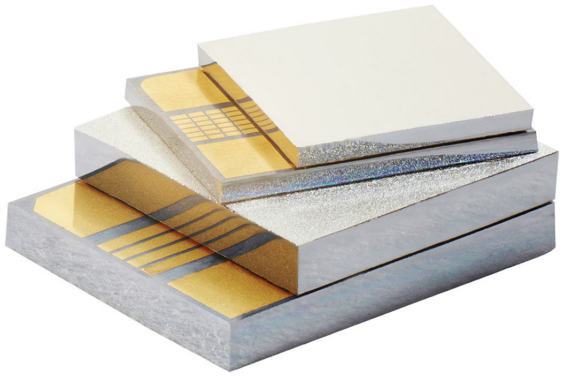


Figure 2.4: Micropelt thin-film thermoelectric chip: MPG-D655

– Vibration harvesters V21BL from **Mide Volture** [82] are attractive because they have two piezo fibers packaged for serial or parallel connection. The parallel connection offers a peak-to-peak of 20 V. Figure 2.5 shows the fiber mounted on the box that houses a motor and cam for generating vibrations. To ensure that the system resonates effectively with the source, a suitable tipping mass is applied. The piezo fiber generates peak power at resonance.
– **EnOceaon** [83] offers the energy module ECO 200, an energy converter for linear motion (Figure 2.6). It can be used to power the PTM 330 radio module or derivates. The energy output at every actuation of the spring is sufficient to transmit 3 sub-telegrams with a PTM 330 module (enables

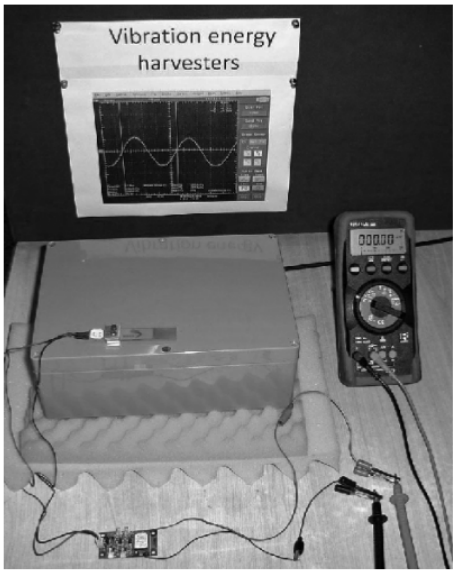


Figure 2.5: Vibration energy generation from Mide Volture's V21BL-piezo fiber.

the implementation of wireless sensors and switches without batteries. Key applications are handled remote controls or industrial switches). Possible applications are miniaturized switches and sensors in building technology and industrial automation.

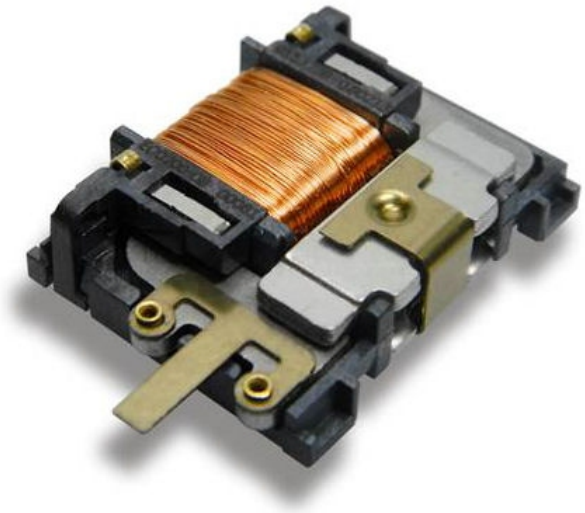


Figure 2.6: ecOcean ECO-200.

2.6 Conclusion

This chapter was concerned by an introduction to the new generation systems said to be energy harvesting since powered through the energy that may be created from a source by some physical process. The Internet of Things will need the energy harvesting technology so as to power all the autonomous objects/sensors that compose it. A wireless sensor is typically an embedded system with real-time constraints. This is why we have focussed on the possible ambient energy harvesting solutions for electronic devices. We also have presented the technology for storing energy i.e. rechargeable batteries and super capacitors since energy needs to be stored before consumption. Finally, we have presented some typical electronic

products that use the energy harvesting technology.

In the next chapter, we will describe the real-time scheduling issue that arises in energy harvesting real-time systems. As introduced in the current chapter, what characterizes energy harvesting systems is both the possible variation in energy production by the source and limitation in the energy that can be stored. Let us recall that we are mainly concerned with wireless sensor nodes. Such devices are constrained in weight, volume and price. All these parameters impact the selection of the energy harvester, the energy storage unit and the computing device. As a consequence, a central question will be to make the best use of the energy and the processing unit under physical limitations and the classical real-time constraints that were described in the previous chapter.

CHAPTER 3

REAL-TIME SCHEDULING WITH ENERGY HARVESTING CONSIDERATIONS

Summary

In this chapter, we give the specificities of the energy harvesting device under study. Next, we describe the issue due to energy autonomy of such device which consists in guaranteeing energy neutrality. We describe scheduling techniques firstly aimed to minimize energy consumption in classical energy powered systems and secondly those which are adapted to energy autonomous systems by achieving energy neutrality.

Contents

3.1 Introduction

The objective of any autonomous system is to ensure perpetual operation without human intervention thanks to batteries (or any other type of energy storage devices), which recharge continuously over time from a renewable energy source. Existing alternative energy sources in our environment described in the previous chapter can be exploited to achieve this goal: it is energy harvesting. It consists of converting energy from the environment and filling an energy storage unit formed by a battery or a supercapacitor. Such an energy storage unit is required because the embedded system needs to operate continuously without missing the available energy (in the storage unit). So using renewable energy (solar, piezoelectric, ..) to power embedded systems requires to reconcile the performances and energy consumption. And this constraint is added to time constraints in real-time systems where the violation of one of them will lead to system failure. Later, we will recall the main scheduling techniques that exist in the literature.

An autonomous system is built around three components (Figure 3.1):
- The energy harvester choosed in dependance on the nature of the environmental energy, the amount of energy required, etc.
- The energy storage unit, such as a battery or a super-capacitor, choosed in dependance on the dynamics of the system, the design constraints and/or cost constraints, etc.
- The energy consumer which here represents the execution support of the real-time tasks. In this chapter, we assume that the energy consumed by the operational part of the embedded device (actuator, LED, etc.) is separately powered, as in the transmitter/receiver module. The energy consumer therefore denotes the electronic card built around a micro-controller or a micro-processor.

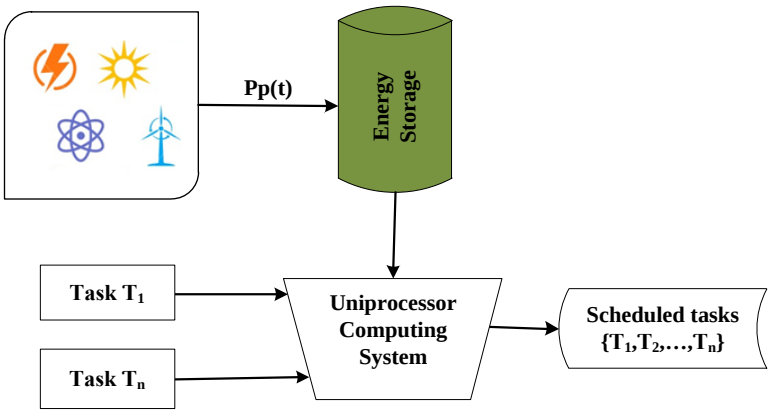


Figure 3.1: Diagram of the ambient energy harvesting system.

3.2 Approaches for minimizing energy consumption

To reduce the energy consumption of a computer system, two types of methods are traditionally applied.
- Methods known as Dynamic Power Management (DPM) allow dynamic management of system activity by switching from idle mode to active mode and vice versa [84]. This is because DPM methods reduce the power consumption of the system without significantly degrading the performance by switching to standby mode when there is no task to be performed, and to switch again to active mode when the processor is requested. These methods may apply to processors that have a sleep function. Therefore, the processor is temporarily switched off whenever necessary. This means that it will consume little energy (called static energy) in this idle state. For example, the processor Intel 80200 has three operating modes with two energy-saving modes that differ by the number of components [84].

– Methods known as DVFS (Dynamic Voltage and Frequency Scaling) allow to dynamically change the frequency of the processor when necessary. Therefore, these methods apply to processors which permit varying the supply voltage and thus the operating frequency [84]. However, let us note that if the processor frequency is reduced, the running job will take longer time to execute and possibly will violate its deadline constraint. For example, if the frequency is halved (reduced to half), the job will take twice as long to execute. Since energy consumption is a quadratic function of frequency, the reduction in energy consumption has a significant impact of energy consumption. In this context, saving energy is done at the expense of stretching the execution times which must be controlled so that the timing constraints of all the hard deadline tasks can still be met. For example, the Transmeta Crusoe processor, the lpARM processor (UC Berkeley) and the Intel Pentium 4M processor are candidate for the DVFS techniques [85, 86] Scheduling real-time tasks on such processors consists to establish an execution order between the tasks, in order that each one of them may be fully executed before deadline. Additionally, the scheduler has an energy management capability in order to determine at all time instants the eligible operating frequency of the processor so to minimize energy consumption.

Techniques for reducing energy consumption using the DPM method have been studied by Benini et al. in [87]. Yao et al. [88] were the first researchers to propose a scheduling algorithm that computes the energy operating frequencies for a set of hard deadline tasks such as periodic ones. The underlying scheduling policy is EDF ans has been proved optimal in the sense of minimal energy consumption. Then, Aydin et al. [89] proposed an approach based on the DVFS technique. In that work, the power received by the source is constant. The algorithm permits to minimize the total energy consumption of periodic tasks by guaranteeing their execution before deadline. Techniques have also been developed for periodic and aperiodic tasks to reduce the energy consumption of the system [90, 91].

3.3 Scheduling approaches for energy neutrality

In this part, we present algorithms which have been presented in the literature for scheduling tasks that operate in real-time energy harvesting systems. Research in that domain is relatively recent since first articles appeared in the beginning of the current century. In most of works, a system is composed of a processing unit that executes tasks with deadlines and consequently consumes the energy that can be stored in a reservoir after production by an environmental source. In some papers, power delivered by this source is invariable along time. In most of papers, the energy storage unit is assumed to be fully charged at the initial time.

From now, it is important to characterize the notion of optimality attached to any scheduling algorithm in the energy harvesting context. Let us assume a given platform which is characterized by the storage unit with given capacity, the energy harvester with given profile of energy production. A scheduling algorithm is optimal if it produces a feasible schedule each time another scheduler produces a feasible schedule under the same conditions i.e. with identical energy harvester and identical energy storage unit.

3.3.1 Algorithm of Allavena and Mossé

In [92], Allavena et al. address the problem of finding a feasible schedule for a set of independent periodic tasks with the same periods (frame based systems). Consequently, the order of task execution within a frame is not crucial for whether the task set is schedulable or not. Moreover, the power scavenged by the energy source is assumed to be constant and all tasks consume energy at a constant rate. Tasks are separated into 2 categories: recharging ones and discharging ones. Whenever a recharging task executes, the energy level of the storage unit increases.

The basic principle of the scheduler is to execute tasks successively from the same category. The scheduler, therefore, chooses discharging tasks so that the available energy level in the storage unit decreases as much as possible to reach the minimum level E_{min}. Then it selects recharging tasks such that the energy level increases as much as possible until it reaches the maximum level E_{max}, thus maintaining the energy level between the two limits E_{min} and E_{max}. Preemption occurs whenever the energy level reaches one of these two values.

Every task is characterized by parameter w_i and belongs to one of the two lists: list of recharging tasks $\mathcal{R}$ or list of discharging tasks $\mathcal{D}$. Furthermore, if the sum of power consumption causes the energy level to finish below its original level ($|\mathcal{R}| < |\mathcal{D}|$), an idle time t_{idle} that causes the system to recharge with rate r is inserted, where $t_{idle} = \frac{|\mathcal{D}|-|\mathcal{R}|}{r}$.

Condition $\mathcal{R} < \mathcal{D}$ is initially verified off-line and then tested on-line each time the energy level reaches one of the two boundaries. If this condition is initially true and if the sum of the execution time (C_i) plus the idle time exceeds the deadline ($\sum_{i=1}^{n} C_i + t_{idle} \leq D_i$), it can be concluded that the system is not feasible. Condition ($\sum_{i=1}^{n} C_i + t_{idle} \leq D_i$) is therefore a necessary and sufficient schedulability condition.

This work is certainly the first one to concentrate on a rechargeable system with hard real-time constraints. The approach is unfortunately unusable because based on a non realistic assumption. As a summary, the main drawbacks of this work are the following:
(i) The model is too restrictive (on energy consumption, task timing parameters), i.e. the solution only deals with frame based systems under the restrictive hypothesis where each task is characterized by an instantaneous power consumption which is constant along time.
(ii) The solution consists of an off-line scheduler which does not provide sufficient flexibility to new generation real-time applications.

3.3.2 Scheduling algorithm LSA

The Lazy Scheduling algorithm (LSA) [93] firstly described in 2006 is one of the most interesting primitive work related to real-time scheduling with energy harvesting considerations. LSA can be described as a variant of the EDF scheduling algorithm since tasks are selected for execution according to their relative urgency. Let us recall that classical EDF is non idling which signifies that the processor is never let idle if at least one task is pending for execution. On the contrary, LSA is an idling version of EDF that may let the processor idle depending on energy availability either currently or in future. LSA specifically intends to keep the energy storage level as high as possible and starts executing task τ_i at time t only if the following conditions are met:

1. τ_i is ready for execution,
2. τ_i has the earliest deadline among ready tasks,
3. The sensor node will not run out of energy if it executes τ_i to completion (at its maximum power),
4. τ_i will not miss its deadline if the node starts executing it at time t.

LSA introduces the concept of energy variability characterization curve (EVCC), which captures the dynamics of the energy source. This concept is used to determine the schedulability of a set of tasks. More particularly, the LSA algorithm uses an off-line schedulability test, given the EVCC of the energy source, the capacity of the energy storage, and the maximum power requirement of a running task. LSA determines whether all the deadlines of a given set of tasks can be met or not.

Although proved to be optimal in [94], LSA suffers several drawbacks: (i) The energy consumed by any task is assumed to be proportional to its execution time. The ratio is given by the speed of the processor. (ii) Optimality is effective only if the processor is able to continuously adjust its consumption power to the source power. (iv) The performance of LSA is highly dependent on the process for predicting future

incoming energy. This aspect is the most challenging one in scheduling for real-time energy harvesting systems. Many applications based on this algorithm were recently proposed:

- LDPC Decoding: a complexity reducing method for iterative message passing decoding algorithms of Low-Density Parity-Check (LDPC) codes in [95],
- L-CSMA/CA scheme: it allows individual nodes to continually estimate the current demand for a broadcast channel and adjust their transmission schedules accordingly in [96],
- Lazy scheduling for assigning jobs in hypercube computers: a scheduling scheme which is uded along with the buddy allocation scheme to process jobs in a multiuser environment in [97].

3.3.3 Multi-Version scheduling algorithm

The multi-version scheduling algorithm introduced in [98] considers that each periodic task has several execution versions with different resulting values. The objective is to execute the most important and valuable versions while meeting all the timing and energy constraints. This static (offline) scheduling solution determines the best task versions, and their execution speeds the maximize rewards. While in [99], the authors proposed dynamic algorithms with Dynamic Voltage Scaling (DVS), according to which the node periodically check the current energy storage and accordingly reschedule the tasks.

3.3.4 EDeg scheduling algorithm

In [100], [101], [102], and [103] EL Ghor et al. introduce a new online scheduling algorithm, called EDeg (Earliest Deadline with energy guarantee) which is a variant of the Earliest Deadline First algorithm. EDeg maintains energy neutrality by making sure that before a task is started, sufficient energy will be available for all future occurring tasks. This scheduler assumes that for taking any decision, it has knowledge on future task arrival times. Edeg is clearly an idling scheduler which may decide when to let the processor in idle state and when to let it busy executing the most urgent ready task. When the stored energy drops below a given threshold, EDeg stops the current running task and starts recharging the storage unit during maximum authorized time. This recharging time is computed so as to keep the system slack time positive and consequently so as to avoid any deadline missing.
The relative performance of the heuristic EDeg has been evaluated in [102]. This study has permitted to prove that EDeg outperforms the classical EDF scheduler in terms of deadline success even if no theoretical evaluation permitted to formally establish its superiority. The requirement to know in advance arrival times, deadlines, and energy demands of all the tasks, seriously limits the applicability of this algorithm in real-life application scenarios. In addition, tasks are assumed to be periodic.

3.3.5 Fixed priority scheduling algorithms

With fixed priority settings, research is recent since the first works appeared in 2011 [104, 105, 106]. A novel fixed priority driven algorithm called (FPC_{ASAP}) was proposed and proved to be optimal (among fixed priority schedulers). It is an idling variant of the well known non-idling FP scheduler that assigns a static priority to every task. As EDeg, the FPC_{ASAP} algorithm requires clairvoyance to predict future occurring tasks and future incoming energy [107].

3.4 Model and Terminology

We have given a brief state of the art on schedulers for energy harvesting real-time systems. Now, we will precisely describe the model studied in our thesis. And we will be prepared to describe the dynamic priority scheduler ED-H introduced in 2014 by M. Chetto which is central for contributing to the aperiodic task servicing issue [8].

3.4.1 System model

Hereafter, we describe the so-called RTEH model (Real Time Energy Harvesting) that comprises a computing element, a set of jobs, an energy storage unit, an energy harvesting unit, and the environmental energy source (Figure 3.2).

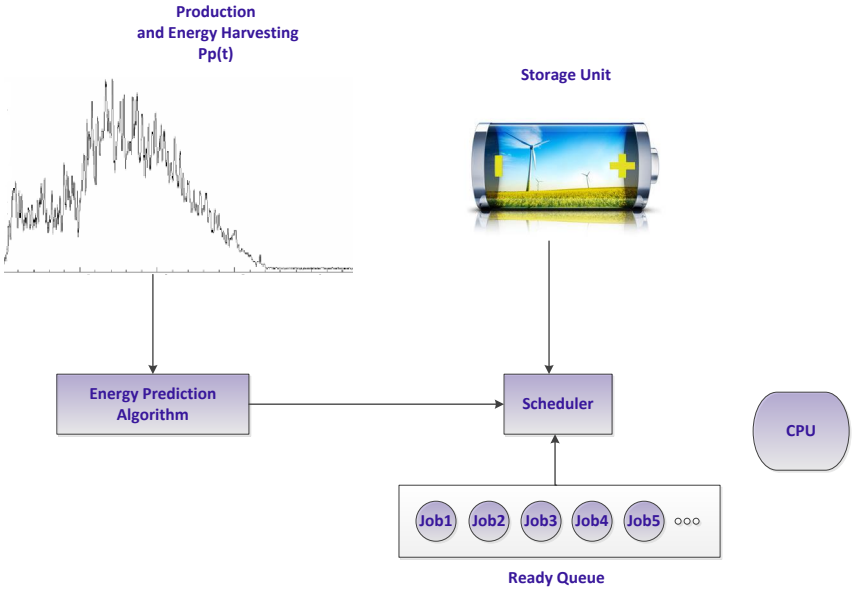


Figure 3.2: The RTEH model.

3.4.1.1 Job model

We will consider the following assumptions: A periodic task set τ can be denoted as follows: $\tau = \{\tau_i \mid 1 \leq i \leq n\}$. Each periodic task τ_i has a period T_i, a relative deadline D_i, a constant worst case execution time C_i (normalized to processor computing capacity), and a constant worst case energy requirement E_i. C_i and E_i can be derived by a static analysis of the source code. So we use a periodic task model with the four-tuple (C_i, E_i, D_i, T_i) associated with τ_i. We consider a constrained-deadline task set τ in which $0 < C_i \leq D_i \leq T_i$. Task τ_i generates jobs which are released at times 0, T_i, $2T_i$,... and must be completed by times D_i, $T_i + D_i$, $2T_i + D_i$,... The hyper-period H of a periodic task set is defined as the least common multiple (LCM) of the request periods T_i, that is $H = LCM(T_1, T_2, ..., T_n)$. The processor utilization of the periodic task set τ is $U_{pp} = \sum_{\tau_i \epsilon \tau} \frac{C_i}{T_i}$ which is less than or equal to 1. Similarly, we define the energy utilization of τ as $U_{ep} = \sum_{\tau_i \epsilon \tau} \frac{E_i}{T_i}$ which characterizes the average energy consumption of τ per time unit.

A job is any request that a task makes. A four-tuple (r_j, C_j, E_j, d_j) is associated with a job J_j and gives its release time, WCET, WCEC, and (absolute) deadline, respectively. The task set τ gives rise to an infinite set of jobs which are scheduled by the optimal uniprocessor scheduler ED-H.
During each time unit, we know an upper bound on the energy consumption of every job equal to e_{Max} energy units. The exact amount of energy effectively drained in every time unit is however not known beforehand.

Throughout the thesis, we will assume that the processor has one operating frequency and its energy consumption is only due to dynamic switching energy.

3.4.1.2 Energy production model

At every time t, the harvester (e.g. solar panel) draws energy from ambient and converts it into electrical power with instantaneous charging rate $P_p(t)$ that incorporates all losses. The energy harvested in the time

interval $[t_1, t_2)$ is thus given by $E_p(t_1, t_2) = \int_{t_1}^{t_2} P_p(t)dt$. We assume that the energy production times can overlap with the consumption times. The energy consumed in any unit time-slot is not less than the energy produced in the same unit time-slot. Consequently, the residual capacity of the energy storage is never increasing every time a job is executed. And the remaining energy needed by any job executing on the processor has to be drawn from the reservoir. The energy produced by the source is not controllable and not necessarily a constant value. It can be predicted with precision in an immediate future and with negligible processing and energy costs.

3.4.1.3 Energy storage model

Our system uses an ideal energy storage unit also called reservoir (e.g. super-capacitor or rechargeable battery) to continue operation even when there is no energy to harvest. Its nominal capacity C corresponds to the maximum amount of energy that can be stored at any time. The energy reservoir receives power from the harvester and delivers power to the processor. The stored energy at any time t is denoted $E(t)$. The energy reservoir does not leak any energy over time. If it is fully charged at time t and we continue to charge it, energy is wasted. In contrast, if it is fully discharged at time t (energy depletion), no job can be executed.
We consider the energy to be wasted when the storage unit is fully charged while we continue to charge it. In contrast, the storage unit is considered fully discharged at time t if $0 \leq E(t) < e_{Max}$ denoted by $E(t) \approx 0$. The application starts with a fully charged storage unit(i.e. $E(0) = C$). The stored energy may be used at any later time and does not leak energy over time.

3.4.2 Types of starvation

According to the RTEH model, a job misses its deadline if one of the two following situations occurs:
- When the job reaches its deadline at time t, its execution is incomplete because the time required to process the job by its deadline is not sufficient.
- When the job reaches its deadline at time t, its execution is incomplete because the energy required to process the job by its deadline is not available. The energy in the reservoir is exhausted when the deadline violation occurs.

3.4.3 Terminology

We now give new definitions peculiar to energy constrained computing systems that we will need throughout the remainder of this thesis.

Definition 8 *A schedule Γ for τ is said to be time-valid if the deadlines of all jobs of τ are met in Γ, considering that $\forall i \in 1, ..., n, E_i = 0$.*

Definition 9 *A job set τ is said to be time-feasible if there exists a time-valid schedule for τ.*

Definition 10 *A schedule Γ for τ is said to be energy-valid if the deadlines of all jobs in τ are met in Γ, considering that $\forall i \in 1, ..., n, C_i = 0$.*

Definition 11 *A job set τ is said to be energy-feasible if there exists an energy-valid schedule for τ.*

Definition 12 *A scheduling algorithm S is said to be energy-clairvoyant if it needs knowledge of the future energy production to take its runtime decisions.*

3.5 Fundamental concepts

3.5.1 Processor demand

Definition 13 *The slack time of a hard deadline job J_i at current time t is given by*

$$ST_{J_i}(t) = d_i - t - h(t, d_i) \tag{3.1}$$

where $h(t, d_i)$ is the total processing demand of uncompleted jobs at t with deadline at or before d_i. $ST_{J_i}(t)$ gives the available processor time after executing uncompleted jobs with deadlines at or before d_i.

Definition 14 *The slack time of a periodic task set τ at current time t is given by*

$$ST_\tau(t) = \min_{d_i > t} ST_{J_i}(t) \tag{3.2}$$

The slack time gives the maximum continuous processor time that could be made available from time t while still guaranteeing the deadlines of all the jobs generated by τ.

3.5.2 Energy demand

Definition 15 *The slack energy of J_i at current time t is given by*

$$SE_{J_i}(t) = E(t) + E_p(t, d_i) - g(t, d_i) \tag{3.3}$$

where $g(t, d_i)$ represents the total energy required by jobs on the time interval $[t, d_i)$. It concerns both jobs which are ready at t but not completed at d_i and future jobs, with deadline less than or equal to d_i.

Clearly, $SE_{J_i}(t)$ represents the maximum energy surplus that could be consumed within $[t, d_i)$ whilst guaranteeing enough energy for jobs with deadline less than or equal to d_i. In other words, if there exists some job J_i such that $SE_{J_i}(t) = 0$, executing any other job with a deadline higher than d_i within $[t, d_i)$ will involve energy starvation for J_i.

Definition 16 *The slack energy of the periodic task set τ at current time t is given by*

$$SE_\tau(t) = \min_{t < d_i} SE_{J_i}(t) \tag{3.4}$$

$SE_\tau(t)$ represents the maximum energy surplus that the system could consume instantaneously at t.

Definition 17 *Let d be the deadline of this active job. The preemption slack energy at the current time t is given by*

$$PSE(t) = \min_{t < d_i < d} SE_{J_i}(t) \tag{3.5}$$

$PSE(t)$ gives the maximum energy that could be consumed by the active job whilst guaranteeing absence of energy starvation for jobs that may preempt it.

3.6 ED-H Scheduling

Let us describe the ED-H scheduler proved to be optimal for the RTEH model.

3.6.1 Informal description

The classic EDF is a greedy scheduler because it runs the jobs as soon as possible and thus spends the energy stored in the storage unit ignoring the future energy needs. In the version of EDF called EDS (Earliest Deadline as Soon as possible), the processor is never inactive if there is at least one job waiting to run. Assuming a set of jobs that can be time-feasible by EDF, the energy starvation for a job J_i can only come from the execution of a job J_j that runs before the arrival of J_i with $d_j > d_i$. The starvation of energy of J_i caused by J_j with $d_j \leq d_i$ could not be avoided by any scheduler. It is clear that the clairvoyance of the arrival of jobs and of the production of energy will help EDF to anticipate an energy starvation and a deadline violation. Therefore, the main idea of the ED-H is to allow the execution of jobs only if no starvation can occur.

3.6.2 Rules of ED-H

Let $L_p(t_c)$ be the list of jobs ready for execution at time t_c. The scheduling algorithm ED-H obeys the following rules:

- **Rule 1:**The order of priority of EDF is used to select the future running job in $L_p(t_c)$.
- **Rule 2:**The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_p(t_c) = \emptyset$.
- **Rule 3:**The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_p(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1. $E(t_c) \approx 0$
  2. $PSE_\tau(t_c) \approx 0$
- **Rule 4:** The processor is imperatively busy in $[t_c, t_c + 1)$ if $L_p(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1. $E(t_c) \approx C$
  2. $ST_\tau(t_c) = 0$
- **Rule 5:**The processor can equally be idle or busy if $L_p(t_c) \neq \emptyset$, $0 < E(t_c) < C, ST_\tau(t_c) > 0$ and $PSE_\tau(t_c) \approx 0$.

Description of ED-H

- The processor must be inactive if the energy storage unit is empty, or if the execution of a job prevents at least one job in the future from being executed because this execution results in an energy starvation; the preemption slack energy being insufficient at time t_c.
- The processor cannot remain inactive if the energy level of the energy storage unit is at its maximum or if the availability of the processor leads to deadline violation due to a zero slack time at time t_c.
- The processor may equally adopt the idle or active state if the energy storage unit is neither full nor empty, and if the system has both a non-null slack time and a non-null preemption slack energy.
- We begin to recharge the energy storage unit when it is empty or when there is not enough energy to guarantee the possible execution of all future jobs.

In the following example, the preemption slack energy is illustrated:

Example 7 *Let us consider two jobs J_1 and J_2 with release times $r_1 = 2$, $r_2 = 0$, execution times C_1=1, $C_2 = 2$, energy consumptions $E_1 = 12$, $E_2 = 15$, absolute deadlines $d_1 = 6$, $d_2 = 8$. Let us consider that the energy level of the battery at time 0 is given by $E(0) = 29$, the capacity of the battery is $C = 40$ and the source power is constant and given by $P_p = 5$.*

At time 0, J_2 is the highest priority task that is ready to be processed. Here, computation of the preemption slack energy is necessary because J_1 with deadline less than deadline of J_2 will be released after time t = 0. Preemption Slack energy is calculated as follows:

$$\begin{cases} SE(J_1,0) = E(0) + \int_0^6 P_p dt - E_1 = 47 \\ SE(J_2,0) = E(0) + \int_0^8 P_p dt - (E_1 + E_2) = 42 \\ PSE(0) = min(SE(J_1,0), SE(J_2,0)) = 42 > 0 \end{cases}$$

Since E(0) = 29 and preemption slack energy is positive, J_2 is authorized to execute immediately. At time $t = 2$, $E(2) = 24$. Now, J_1 has the highest priority. It is executed till $t = 3$, where E(3) = 17 energy units 3.3.

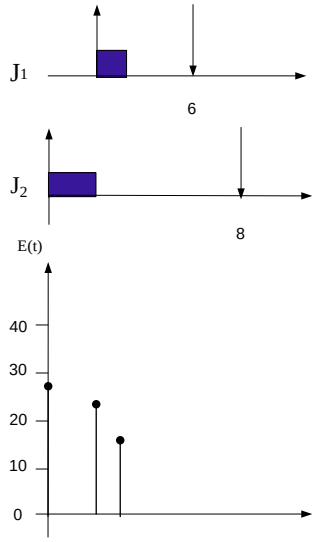


Figure 3.3: Illustration of Preemption Slack Energy.

Now, an example illustrating the ED-H schedule is presented:

Example 8 *Consider a periodic task set Γ that is constituted of two tasks, $\Gamma = \{\tau_i \mid 1 \leq i \leq 2 \ \ and\ \tau_i = (C_i, D_i, T_i, E_i)\}$. Let $\tau_1 = (3, 6, 6, 8)$ and $\tau_2 = (2, 8, 8, 5)$. We suppose that the energy storage capacity is $E = 4$. To simplify, the rechargeable power, P_p, is taking constant along time and equals 2.*

Figure 3.4 represents the resulting schedule under ED-H over the hyperperiod $H = 24$. Let us describe it.

First of all, we have to schedule the periodic task set Γ according to ED-H. Since all the tasks are executed before their deadline and without expending the energy reservoir, we confirm that Γ is schedulable (Figure 3.4). In details:
At time $t = 0$, all tasks are released. τ_1 is the task with highest priority and is executed until $t = 3$ where $E(3) = E(0) - E_1 + P_p * C_1 = 2$ energy units. At time $t = 3$, τ_2 is the task with highest priority and is executed until $t = 5$ where $E(5) = E(3) - E_2 + P_p * C_2 = 1$ energy unit.
From time $t = 5$ up to $t = 6$, the processor remains idle because there are no pending tasks. During that time interval, the energy storage will recharge and the energy level at $t = 6$ is given by $E(6) = E(5) + P_p = 3$ energy units.
τ_1 is now the highest priority task and is executed until $t = 9$ where the capacity of energy storage becomes 1 energy unit. At $t = 9$, τ_2 is the highest priority task ready to be processed but, for execution, there is insufficient energy in the energy storage unit. So, we have to insert an idle time to let the processor passive as long as the energy storage has not replenished completely and the further start time of the next periodic task has not been reached. The slack time is equal to 1. Hence, the processor is let idle till $t = 10$ where the energy reservoir will recharge leading to $E(10) = 3$ energy units. At time $t = 10$, τ_2 is the highest priority task, ready to be processed, runs and finishes at $t = 12$. $E(12) = 2$ energy units.
We continue to schedule Γ in the same way till the end of the hyperperiod, where the energy reservoir has 4

energy units at $t = 24$.

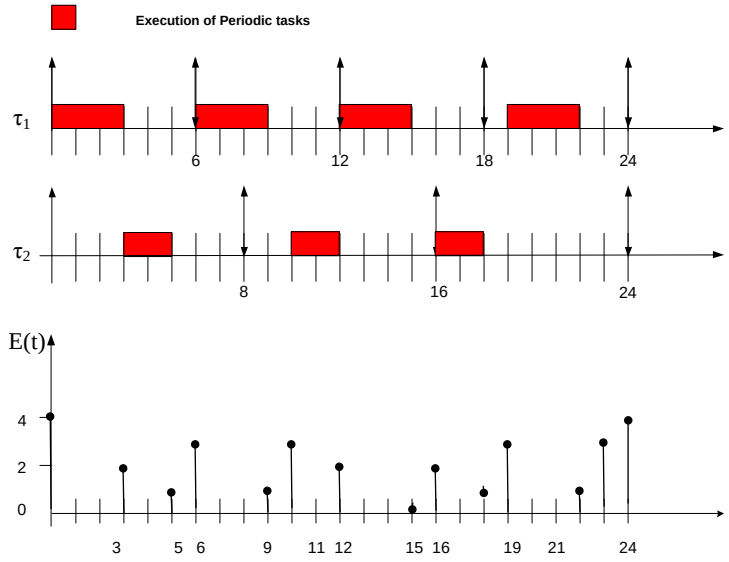


Figure 3.4: ED-H scheduling

3.6.3 Properties of ED-H

3.6.3.1 Optimality analysis

Theorem 6 *[8] The ED-H scheduling algorithm is optimal for the RTEH model.*

The optimality of ED-H means that ED-H can produce a valid sequence if and only if:
– there is no time interval with a duration smaller than the processor demand.
– and there is no time interval where energy demand exceeds the total energy available in that interval.

3.6.3.2 Clairvoyance analysis

The following theorem gives an important restriction on ED-H. It precisely gives the length of the time interval in future where prediction is required.

Theorem 7 *[8] The ED-H scheduling algorithm is online lookahead-D.*

It was proved in [108] that no online scheduling algorithm can be optimal without clairvoyance on at least D units of time. To make a decision at any time t_c, ED-H requires to know both the arrival process of the jobs and the energy production process on the following D units of time. Regarding clairvoyance, no scheduler can be better than ED-H which is precisely lookahead-D.

3.6.3.3 ED-H Schedulability test

The objective of any schedulability/feasibility test is to test whether time and energy will be sufficient to meet the time requirements of all executing jobs during the whole lifetime of the application. In the design of real-time systems composed of well-known periodic tasks, without energy constraints, we perform an off-line test and use an online algorithm to schedule and execute jobs such as EDF, FP, etc.
For the RTEH model, the test can be performed off-line only if 1) all the jobs are precisely known (that

is the case of periodic task instances), and 2) the profile of the incoming environmental energy is characterized for the entire lifetime of the application. In many applications, the source power varies with time. Consequently, there two possibilities. Either an off line schedulability test is implemented by taking into account the worst case situation in terms of energy production. This option has the drawback to be too pessimistic and leads to consider a system as non schedulable even if it is actually schedulable. The second option is to implement schedulability testing at run time periodically for example. The schedulability test is performed using specific prediction techniques which are adapted to the nature of the incoming energy. The on line schedulability test permits to verify that all jobs released on the next window of time will be properly scheduled. If the test gives a negative answer, a decision could be to commute on a degraded mode with a resulting lower quality of service. This can be achieved by discarding some jobs (those judged as the less important ones for the application). The approach called skip-over [109] is one of these approaches [110].

The schedulability test of the ED-H scheduler consists of a necessary and sufficient condition [8] as reported in the following theorem:

Theorem 8 *[8] A set of jobs τ compliant with the RTEH model is feasible if and only if*

$$SST_\tau \geq 0 \; and \; SSE_\tau \geq 0 \tag{3.6}$$

SST_τ is defined as the static slack time of a set of jobs τ (see Definition 10 in [8]). SSE_τ is the static slack energy of a job set τ (see Definition 17 in [8]).

This feasibility test is implemented in $O(n^2)$ since n^2 time intervals are the object of static slack time calculation and we assume to predict the ambient energy on each time interval by a finite number of values.

3.7 Conclusion

In this chapter, we have restricted our study to energy autonomous real-time systems with a mono-frequency uniprocessor platform. We have presented a brief state of the art related to scheduling tasks with deadlines with energy harvesting considerations. In majority of reasearch studies, the objective is to minimize the total energy consumed by software so as to maximize the lifetime of the application or to maximize duration between two recharges of batteries. In contrast, the objective of scheduling under energy harvesting setting is to guarantee energy neutrality that is to make sure that the system will never consume more energy than harvested while satisfying the real-time constraints expressed in deadline success.

This scheduling issue is much more difficult compared to real-time systems with no energy limitation. The complexity here comes from the variation of the energy produced by the source which does not necessarily fit to the timing characteristics of the real-time tasks. Consequently, any scheduling algorithm clearly includes a power management procedure which is able to decide on-line when to make the processor busy and for how long time, when to let the processor idle then permitting the energy storage unit recharging.

We have precisely described the system model under study which includes hard deadline tasks, may be periodic or not. The optimal dynamic scheduler ED-H proved to be optimal in 2014 for this model was introduced. As a variant of the EDF scheduler, we will assume to apply it to a set of periodic tasks, main part of the application software.

The next chapter will give the first contribution of our thesis. We will be concerned with the scheduling issue that arises when additional aperiodic tasks need to be executed with the hard deadline periodic ones. This problem has been extensively studied with no energy limitation. We will first introduce two new servers, namely BEP and BES servers that can be used in conjunction with the ED-H scheduler.

PART II

APERIODIC TASK SCHEDULING CONTRIBUTIONS

CHAPTER 4

BACKGROUND-BASED SERVERS: BES AND BEP

Summary

The purpose of this chapter is to study the scheduling issue related to an hybrid set of tasks composed of hard deadline periodic tasks and soft aperiodic tasks in the energy harvesting context. Our study is conducted with the system model that was described in the previous chapter. The objective is to provide a servicing procedure which is able to minimize the response time of occurring aperiodic tasks. We assume that the hard deadline periodic tasks are scheduled preemptively according to the optimal scheduler ED-H (see Chapter 3 Section 3.6). In addition, the FCFS (First Come First Serve) rule is applied to service the pending aperiodic tasks. In that chapter, we propose two aperiodic servers based on the background principle, respectively named BES and BEP. The effectiveness of these servers will be proved in Chapter 6 of this thesis in comparison with other servers.

Contents

4.1 Introduction

How to extend the classical Background servicing (BG) approach to energy harvesting hypotheses is the central issue of this chapter. The most important problem with this approach is that under high periodic processor utilization, aperiodic response times may be very long and too long for some applications. However, the BG server is the simplest method to handle aperiodic tasks in the presence of periodic ones. Any pending aperiodic task is executed only when there are no periodic task ready to be executed. In other terms, the background server can be assimilated to the lowest priority task of the system.
Firstly, we propose the "Background with Energy Surplus" (BES) server which benefits from energy surplus in the storage unit to execute the aperiodic tasks. Secondly, we propose the "Background with Energy Pre-

serving" (BEP) which executes the aperiodic tasks as long as no energy starvation is involved for periodic tasks.

4.2 System Model and Terminology

We consider the RTEH model described previously. In addition to periodic tasks with deadlines, aperiodic tasks arrive in the system irregularly. Each aperiodic task has worst case execution time and worst case energy requirement considered to be known at its arrival time i.e. the time at which the task is activated and becomes ready to execute. Any aperiodic task has no deadline and unpredictable arrival time.

We will use the following notation throughout the chapter: Ap is a stream of aperiodic occurrences defined as $Ap = Ap_i(a_i, c_i, e_i), i = 1..m$, where a_i is the arrival time, c_i is the worst case execution time, and e_i is the worst case energy requirement. The aperiodic tasks are processed in FIFO (FCFS) order.

From the characteristics of each aperiodic task, we are interested in the following variables:

f_i: Finish time of Ap_i,

s_i: Start time of Ap_i,

RT_i: Response time: $RT_i = f_i - a_i$,

JT_i: Jitter time: $JT_i = s_i - a_i$,

LT_i: Latency time: $LT_i = f_i - s_i$,

ART_i: Normalized response time:$ART_i = \frac{f_i - a_i}{c_i}$,

AJT_i: Normalized jitter time: $AJT_i = \frac{s_i - a_i}{f_i - a_i}$,

ALT_i: Normalized latency time: $ALT_i = \frac{f_i - s_i}{c_i}$.

4.3 The Background with Energy Surplus (BES) Server

Aperiodic tasks are executed when there is no ready periodic task and the energy reservoir is fully replenished. That means that any aperiodic task consumes the energy which would be wasted if there were no aperiodic task in the system. According to this approach, an aperiodic task is authorized to execute as long as its execution permits to guarantee that the storage unit be full when the next periodic task will release.

The algorithm behind the BES server is described below (Algorithm 1):

Example 9 *Consider a periodic task set Γ that is composed of two tasks, $\Gamma = \{\tau_i \mid 1 \leq i \leq 2 \ \ and\ \tau_i = (C_i, D_i, T_i, E_i)\}$. Let $\tau_1 = (4, 9, 9, 18)$ and $\tau_2 = (3, 12, 12, 18)$. Let us consider also $Ap = \{Ap_j \mid Ap_j = (a_j, c_j, e_j)\}$ the stream of 2 aperiodic tasks where $Ap_1 = (9, 1, 5)$ and $Ap_2 = (18, 3, 15)$. We suppose that the energy storage capacity is $E(0) = 10$. To simplify, the rechargeable power P_p is taking constant along time and equals 4.*

Figure 4.1 illustrates the actual schedule with the BES server during one hyperperiod $H = 36$.
At time $t = 0$, all periodic tasks are released. τ_1 is the highest priority one and executes until $t = 4$ where $E(4) = E(0) - E_1 + P_p * C_1 = 8$ energy units. At time $t = 4$, τ_2 is the highest priority task and executes until $t = 7$ where $E(7) = 2$ energy units.

From time $t = 7$ up to $t = 9$, the processor remains in the idle state since there are no pending tasks. During that time interval, the energy storage unit recharges. And the energy level at $t = 9$ is given by

Algorithm 2 BES server

Require:
t: current time
$L(t)$: list of ready periodic tasks at t
$J(t)$: list of ready aperiodic tasks at t
1: **while** TRUE **do**
2: **if** L(t) not empty **then**
3: schedule_ED-H(L(t))
4: **else if** J(t) is not empty AND energy reservoir is FULL **then**
5: schedule_FCFS (one time unit of J(t))
6: **else**
7: let processor idle
8: **end if**
9: $t := t + 1$
10: **end while**

$E(9) = 2 + 8 = 10$ *energy units.*

At $t = 9$, Ap_1 *is released. As* τ_1 *is now the highest priority task, it is executed until* $t = 13$ *where the energy storage level becomes* 8 *energy units.* τ_2 *is now executed until* $t = 16$ *where the energy storage level becomes* 2 *energy units. At* $t = 13$, Ap_1 *can't be processed as the storage unit is not fully charged. Accordingly, we continue to schedule* Γ. Ap_1 *is authorized to execute only from* $t = 33$, *where the ready periodic list is empty and the energy storage is completely charged (*$E(33) = 10$ *energy units).* Ap_1 *is executed till* $t = 34$ *where* $E(34) = 9$ *energy units. Its response time is* 25 *time units.* Ap_2 *is authorized to execute at time* $t = 35$ *until* $t = 38$ *where* $E(38) = 7$ *energy units and its response time is 20 units of time.*

Consequently, the BES aperiodic server (Figure 4.1) shows poor performance in terms of aperiodic responsiveness even if with a very simple implementation and consequently low run-time overhead.

4.4 The Background with Energy Preserving (BEP) Server

Despite its simplicity, BES shows poor performance theoretically because it performs totally background in terms of time and energy, i.e. it executes the aperiodic tasks in the remaining idle times and energy. Still we could achieve better aperiodic response times if we agree to pay something more. For example, looking at the example in Figure 4.1, we could argue that the two aperiodic jobs may be served sooner if we take benefit from the non execution of periodic tasks as well of the availability of energy in the system, without compromising the feasibility of the system. For that reason, system slack energy at time t, the minimum of the slack energy of all periodic jobs in the system, is the maximum amount of energy that can be consumed from t continuously while still satisfying all the timing constraints of the tasks.

Therefore, the main idea of BEP server is executing a ready aperiodic task if and only if a) the energy reservoir is not empty, b) no periodic task is pending for execution and c) the system slack energy is strictly positive; the system slack energy is calculated on all periodic jobs in order to avoid any energy starvation. If these three conditions are satisfied, the processor cannot be inactive and should run the occurring job. Indeed, the process of BEP implies a modification of the periodic tasks scheduling with respect to the schedule produced when there are no aperiodic tasks.

The procedure of generating the BEP schedule is summarized in the following pseudo-code (Algorithm 2).

Example 10 *As an example, we consider the same hypothesis which we studied in the previous section.*

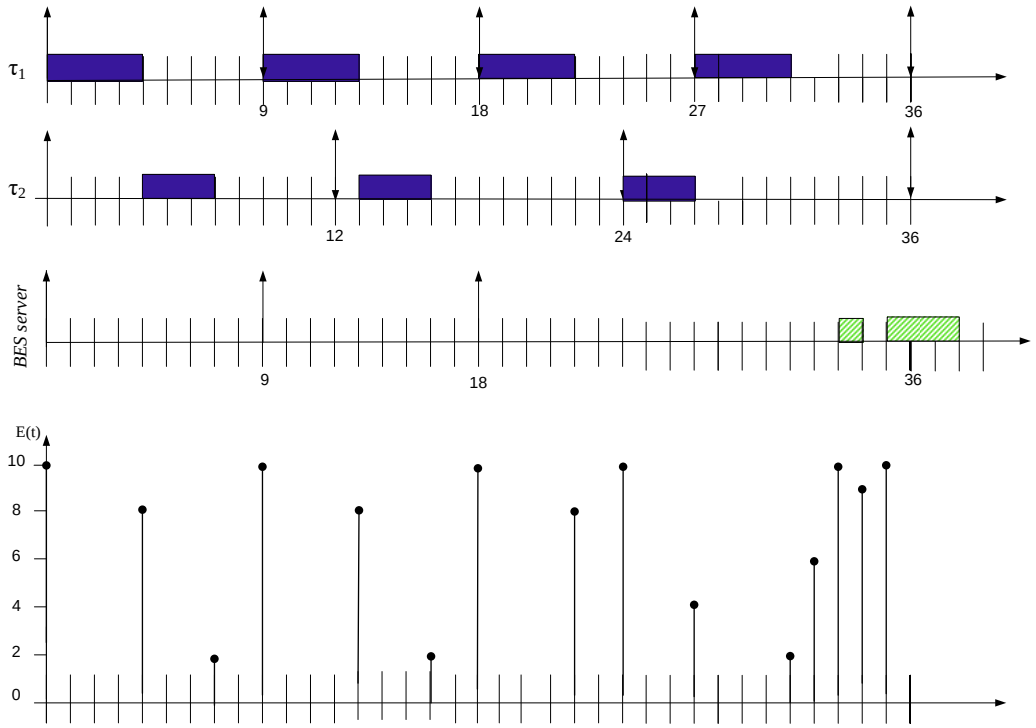


Figure 4.1: Aperiodic servicing with the BES server.

Figure 4.2 illustrates the schedule produced by BEP during the hyperperiod $H = 36$. *It shows the same behavior of BES until the arrival of the aperiodic task* Ap_1 *at time* $t = 9$. *But as its execution conditions are not all verified (i.e. the ready periodic list is not empty),* Ap_1 *cannot be executed. It is processed at time* $t = 16$ *where all conditions are applied (storage unit is not empty, no ready periodic task, and by applying equation 3.4 of Chapter 3, we have* $SE(16) = E(16) + \int_{16}^{18} P_p dt = 2 + 2 * 4 = 10 > 0$*).* Ap_1 *is finished at* $t = 17$, *where* $E(17) = 1$ *energy unit. Finally, we note that the response time of* Ap_1 *is* 8 *time units.*

At $t = 22$, Ap_2 *cannot be executed because only two among three conditions are verified (empty ready periodic list and not empty energy level). On the other hand,* $SE(22) = E(22) + \int_{22}^{24} P_p dt = 4 > 11$ *which is less than 15, the energy to be consumed by* Ap_2. *In this way,* Ap_2 *is executed at* $t = 32$ *until* $t = 35$, *where* $E(35) = 3$ *energy units. Its response time equals* 17 *units of time. The example shows that BEP server performs better than BES for each of* Ap_1 *and* Ap_2. *For example, the response time of* Ap_1 *is* 60% *shorter compared to that under BES.*

4.5 Implementation and overhead considerations

This chapter described two algorithms for scheduling aperiodic tasks among periodic ones in the energy harvesting context. For these two background approaches, we assume that the operating system maintains two queues. In the ready periodic tasks list, the jobs are placed and ordered according to deadlines (EDF rule used by the ED-H scheduler). Each newly arrived aperiodic task is placed in the aperiodic queue and is served in FCFS order. The two servers are simple to implement, nevertheless the complexity of BEP comes mainly from slack energy computations which involve more overhead. So, the complexity of BEP in the worst-case scenario is the same as the slack energy computation algorithm, which has a complexity of $O(m.n)$ where m is the number of iterations and n is the number of tasks [101]. We note that the number of iterations m depends on the periods and deadlines of the tasks and is bounded by $\frac{max_{1<i<n}(D_i)}{min_{1<i<n}(T_i)}$. Thus, the complexity of BEP is pseudo-polynomial.

Algorithm 3 BEP server

Require:
t: current time
$L(t)$: list of ready periodic tasks at t
$J(t)$: list of ready aperiodic tasks at t
1: **while** TRUE **do**
2: **if** L(t) is not empty **then**
3: schedule_ED-H(L(t))
4: **else if** J(t) is not empty AND reservoir is not empty AND $SlackEnergy(t) > 0$ **then**
5: schedule_FCFS (J(t))
6: **else**
7: let processor idle
8: **end if**
9: $t := t + 1$
10: **end while**

4.6 Conclusion

According to the classical Background aperiodic servicing approach, aperiodic tasks are scheduled and executed only at times when there is no periodic task ready for execution. This approach will clearly produce a correct schedule in that sense that presence of aperiodic tasks does not influence the scheduling of periodic tasks. Consequently, periodic tasks are guaranteed to respect their deadline requirements. However, the response times of aperiodic tasks is prolonged unnecessarily.
Two variants of the Background server were proposed. Under the BES server, aperiodic tasks should wait for the total replenishment of the storage unit. Under the BEP server, aperiodic tasks execute only if their energy consumption does not provoke possible energy starvation for any periodic task. This is guaranteed by computing the so-called slack energy of the system which gives at every instant, the maximum energy which could be consumed while still guaranteeing energy feasibility of periodic tasks. We may easily predict that the BEP server will outperform the BES server. Nevertheless, this is at the cost of the on line computing of the slack energy. Simulation results will be reported in Chapter 6 to evaluate the performance of the BEP and BES servers.
In the next chapter, we will describe an alternative technique in order to enhance the average aperiodic responsiveness over these two Background approaches.

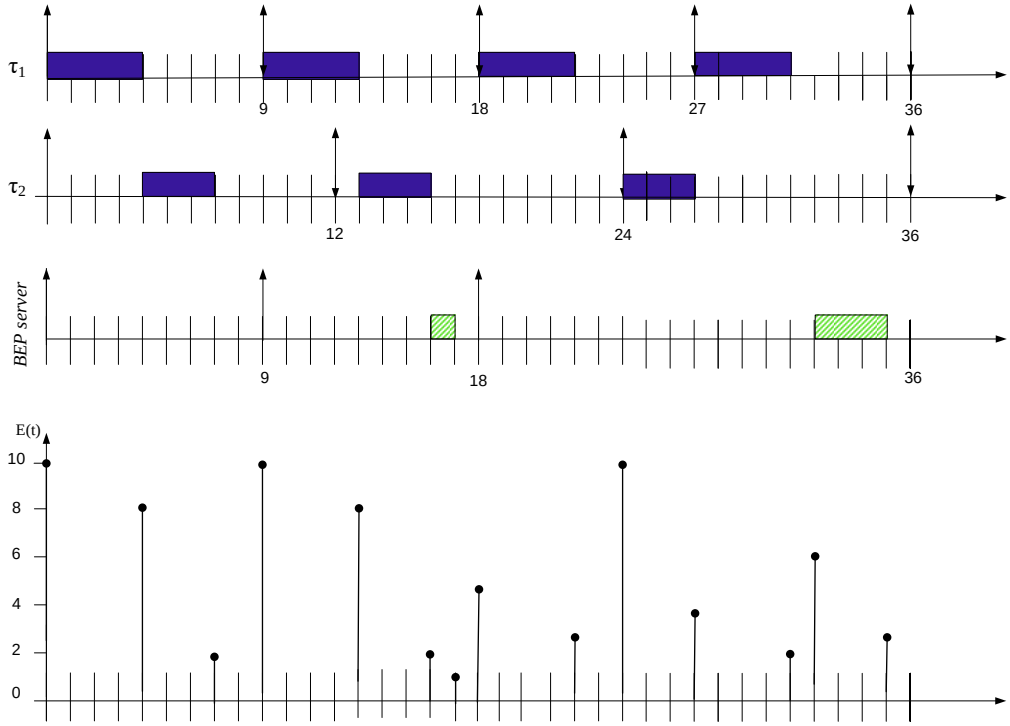


Figure 4.2: Aperiodic servicing with the BEP server.

CHAPTER 5

SLACK STEALING-BASED SERVER: SSP

Summary

This chapter presents a new algorithm for servicing aperiodic tasks called SSP (Slack Stealing with energy Preserving). SSP can be viewed as an extension of the EDL server which is optimal with no energy considerations. This Slack Stealing server works in the same way as the EDL server but taking into account variations on energy availability. We will see here that the concept of slack concerns both time and energy. The optimality of the SSP server is established in the sense that SSP provides the shortest aperiodic response time among all possible aperiodic servers for real-time energy harvesting systems.

Contents

5.1 Task model

The model associated to SSP is same as the one described in Section 4.2, Chapter 4.

5.2 Energy model

SSP is able to work with any energy source and storage models, theoretically and experimentally as we will see in Chapter 6. The energy produced by the source $P_p(t)$ is not controllable and not necessarily a constant value. To avoid short-term energy shortages, we have an accurate estimation of short-term energy within some prediction errors margin. Furthermore, an ideal energy reservoir (e.g. super-capacitor or rechargeable battery) is considered to continue operation even when there is no energy to harvest. The

energy reservoir receives power from the harvester and powers the processor. The stored energy at any time t is denoted $E(t)$. The energy reservoir does not leak any energy over time. If it is fully charged at time t and if we continue to charge it, energy is wasted. In contrast, if it is fully discharged at time t (energy depletion), no job can be executed.

5.3 Informal description

We present a new server, namely SSP (Slack Stealing with energy Preserving) which builds upon previous research into slack stealing algorithms including the EDL scheduler described in the state of the art. Let us recall that the EDL server determines the maximum processing time which may be stolen from hard deadline periodic tasks, without jeopardising their timing constraints. Periodic tasks are scheduled according to the EDF scheduler. The EDL server was extended to tasks with synchronization constraints. An approximate version of the EDL server was proposed to reduce the runtime overheads due to high computation costs. The original slack stealer, EDL is greedy since the available slack time is always consumed if there is at least one aperiodic task ready to run.

The main principle of the slack stealer SSP for aperiodic servicing with ED-H is to authorize aperiodic job executions as long as it does not involve a deadline violation for all the jobs generated by the periodic task set τ. Let us recall that a deadline violation occurs either because of processing time starvation (lack of time to complete a task before deadline) or energy starvation (lack of energy to complete a task before deadline).

This leads us to consider the system slack at current time t as a pair of values respectively called slack time and slack energy. The slack time of τ at time t is defined as the maximum processing time which is available at t after executing timely the tasks of τ. Slack time is a dynamic value that expresses variation of processing surplus. Its computation permits to determine whenever necessary for how long time the processor could be let either idle or busy executing additional tasks such as aperiodic ones.
The slack energy of τ at time t is defined as the maximum energy which is available at t after executing timely the tasks of τ. Slack energy is also a dynamic value that expresses variation in energy surplus. Its computation permits to determine whenever necessary how much energy could be either wasted or consumed executing additional tasks such as aperiodic ones.

In summary, the basic idea of the SSP server is to steal as much as possible both processing time and energy. It leads to execute the aperiodic tasks as soon as possible while avoiding energy starvation and deadline missing for periodic tasks. Whenever no aperiodic tasks are present, the periodic tasks are operated classically with the ED-H scheduler.

Whenever new aperiodic task arrives, it uses the collected values of slack time and slack energy to decide to service either an aperiodic task or a periodic one.

The slack stealer SSP can be viewed as a task which is ready for execution whenever the aperiodic queue is non-empty. This task is suspended when the queue is empty. The slack stealer receives the highest priority whenever there is slack i.e both slack time and slack energy. 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 can be described by the following pseudo-code:

Algorithm 4 SSP

Require:
t: current time
$L(t)$: list of ready periodic tasks at t
$Ap(t)$: list of ready aperiodic tasks at t
1: **while** TRUE **do**
2: **if** Ap(t) is not empty AND energy reservoir is not empty AND $SlackEnergy(t) > 0$ AND $SlackTime(t) > 0$ **then**
3: schedule_FCFS (Ap(t))
4: **else**
5: schedule_ED-H(L(t))
6: **end if**
7: $t := t + 1$
8: **end while**

5.4 Slack computations

5.4.1 Computing the current slack time

The system slack time, denoted $ST(t)$ in this dissertation, is the same as EDL in Section 1.4.2.1, Chapter 1. It is used to delay periodic tasks executions as long as possible by anticipating all the available slack time. The tasks are scheduled according to EDF in Section 1.3.1, Chapter 1. EDL is used in the energy-harvesting systems context to maximize replenishment periods. The slack time in classical scheduling concepts shows the maximum length of idle periods within a time interval. When the system is energy constrained, it uses the slack time notion to replenishment as in the ED-H model [8].

When an aperiodic job is ready to be executed at time t, the SSP computes the available system slack time. Then, the ready job is executed only if the system slack time is positive, the system slack energy is positive, and the energy is sufficient to execute. Otherwise, if a periodic job is ready to execute, the SSP will give it permission. If the energy reservoir is empty, the ready job is delayed until the slack time is fully consumed. The computation of slack time with EDF scheduling is described in Section 1.4.2.1, where we recall the static offline approach as well as the dynamic online one. In order to compute the system slack time, we recall some definitions about the dynamic approach notions presented in [8]. Formally:

The slack time of a hard deadline job J_i at current time t is

$$ST_{J_i}(t) = d_i - t - h(t, d_i) \tag{5.1}$$

where $h(t, d_i)$ is the total processing demand of uncompleted jobs at t with deadline at or before d_i. $ST_{J_i}(t)$ gives the available processor time after executing uncompleted jobs with deadlines at or before d_i. The equation of the processor demand $h(t, d_i)$ is given by the following equation:

$$h(t, d_i) = \Sigma_{d_k \leq d_i} C_k \tag{5.2}$$

We may then define the slack time of job τ at current time t as follows:

$$ST_\tau(t) = \min_{d_i > t} ST_{J_i}(t) \tag{5.3}$$

The slack time as computed with (5.3) gives the maximum continuous processor time that could be made available from time t while still guaranteeing the deadlines of all the jobs generated by τ.

The computation of slack time is widely described and illustrated in Section 1.4.2.1, Chapter 1..

5.4.2 Computing the current slack energy

As mentioned in previous section, the Slack Stealer executes an aperiodic task whenever there is slack time and slack energy in the system, i.e. $ST(t) > 0$ and $SE(t) > 0$. Moreover, the intuition behind ED-H is to run jobs according to EDF rules, but this decision is constrained by the use of the notion of slack energy to predict eventual future energy failures. If a job has the potential to miss its deadline in the future due to energy insufficiency, the current jobs are delayed as long as possible by expanding the available slack time to replenish a maximum of energy. In our algorithm, we use the EDL server [18] to compute the slack time and determine the busy periods of periodic tasks. Furthermore, when the energy reservoir is fully replenished during an idle period, the algorithm resumes executions in order to avoid energy waste. By definition, the slack energy of a job J_i at time t represents the maximum energy that can be consumed to execute jobs from t until the deadline of J_i, while still guaranteeing energy requirements and deadlines [8].In other words, it means the maximum amount of idle time that can be used to delay executions without violating deadlines. Two cases are examined:

1. If this quantity is positive, the ready aperiodic job is authorized to execute and consume at most $SE(t)$.
2. If there is no slack, it is delayed as much as available slack time without wasting energy.

Thus, we have to compute the slack energy of every job with deadline less than or equal to d_i. The minimum of the values of all periodic jobs in the system will gives us the system slack energy; i.e. the maximum energy surplus that the system can consume instantaneously at t to execute an aperiodic job ready at t. However, the minimum of the slack energy of all periodic jobs at time t with a higher priority than an active, say J_i gives us the preemption slack energy; i.e. the maximum energy that can be consumed by J_i whilst guaranteeing absence of energy starvation for jobs that may preempt it. The computation of the slack energy of J_i at current time t is performed as follows:

$$SE_{J_i}(t) = E(t) + E_p(t, d_i) - g(t, d_i) \tag{5.4}$$

Where $E(t)$ is the energy storage capacity at time t. $g(t, d_i)$ represents the total energy required by jobs on the time interval $[t, d_i)$. It concerns both jobs which are ready at t but not completed at d_i and future jobs, with deadline less than or equal to d_i. The total energy produced by the source within $[t, d_i)$ is $E_p(t, d_i) = \int_t^{d_i} P_p(t)dt$ where $P_p(t)$ gives the source power that varies with time t.

So, the slack energy of J_i is the difference between the energy available within the interval $[t, d_i)$, i.e. $E(t) + \int_t^{d_i} P_p(t)dt$, and the energy demand of J_i and higher priority jobs that are released at or after t and have a deadline earlier than d_i. The equation of the energy demand $g(t, d_i)$ is given by the following equation:

$$g(t, d_i) = \Sigma_{d_k \leq d_i} E_k \tag{5.5}$$

Before running the job that is ready to be executed at time t, the system slack energy denoted $SE(t)$ is computed. Its computation is summarized as follows:

$$\begin{aligned} g(t, d_i) &= \Sigma_{d_k \leq d_i} E_k \\ SE_{J_i}(t) &= E(t) + \int_t^{d_i} P_p(t)dt - g(t, d_i) \\ SE(t) &= \min_{t < d_i} SE_{J_i}(t) \end{aligned}$$

Illustrations of slack energy

We will illustrate an expanded example to show the meaning of this concept. However, let us first recall the definition we gave in Chapter 3 for each of the *Preemption slack energy* notion and *System slack energy* notion.

– *Preemption slack energy* gives the maximum energy that could be consumed by the active job whilst guaranteeing absence of energy starvation for jobs that may preempt it.
– *System slack energy* represents the maximum energy surplus that the system could consume instantaneously at t.

An example illustrating the use of the preemption slack energy and its computation, was presented in Example 7, Chapter 3, by considering a periodic job set.

Now, let us consider scheduling both periodic and aperiodic task sets. The periodic task set is scheduled according to ED-H till the release of an aperiodic task. System Slack energy must then be computed to check whether it can be executed or not. In details:

Example 11 *We will take another example to handle the joint scheduling. Let us illustrate the case where the slack energy of the system reveals to be positive at a given time t. We consider a periodic task set* Γ*, composed of two periodic tasks* τ_i *with* $\tau_i = (C_i, D_i, T_i, E_i)$*. Let* $\tau_1 = (1, 5, 6, 12)$ *and* $\tau_2 = (4, 8, 10, 15)$*. We assume that the energy level at time 0 is* $E(0) = 25$ *and that the capacity of the battery is* $C = 35$*. For simplicity, we assume that the rechargeable power is constant along time with* $(P_p = 5)$*.*

Before beginning to schedule the task set Γ*, we verify the energy feasibility condition.* $U_e = \sum_{i=0}^{n} \frac{E_i}{C_i} = \frac{149}{30} \leq 5$*. Consequently,* $U_e \leq P_p$*. That means that the average instantaneous power consumption of* Γ *is no more than the average power drained from the environmental source, here constant along time.*

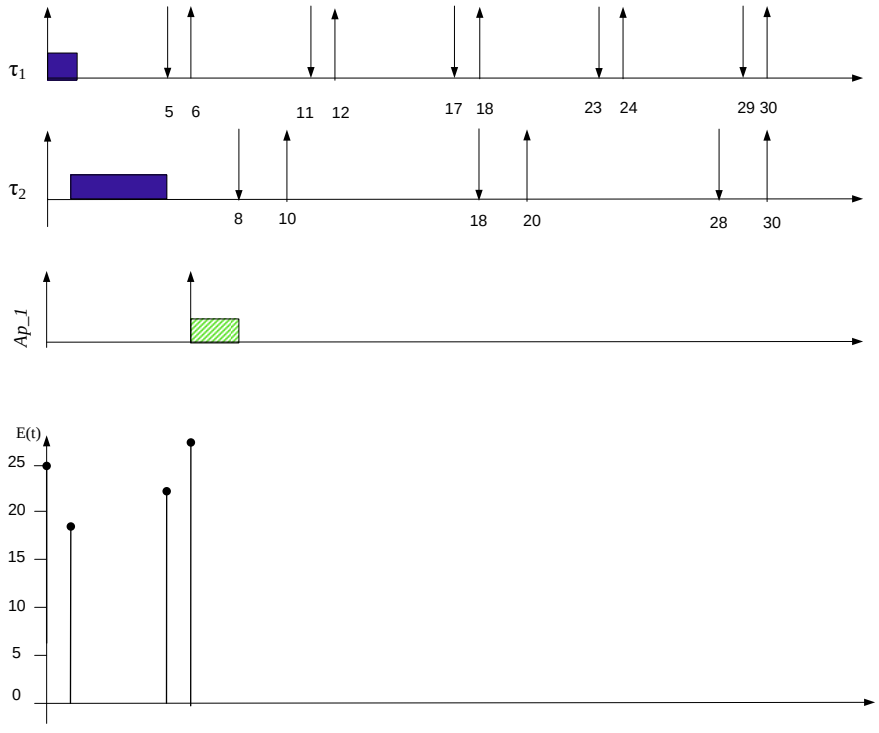


Figure 5.1: Illustration of Slack Energy.

At t= 0, τ_1 is the highest priority task. It is executed until time $t = 1$ where E(1)= 18. At t= 1, τ_2 is the highest priority task. It is executed until time $t = 5$ where E(5)= 23. At t=6, the aperiodic task Ap_1 arrives with computation time $c_1 = 2$ and energy consumption $e_1 = 8$ energy units. System slack energy is then computed at time t=6, it is given by the minimum of the slack energy of all periodic instances in the system. The processor remains inactive until $t = 6$, where E(6)= 28. The following steps summarize the

computation of system slack energy.

$$\begin{cases} SE(\tau_1, 6) = E(0) + \int_6^{11} P_p dt - (E_1 + E_2) = 31 \\ SE(\tau_2, 6) = E(0) + \int_6^{8} P_p dt - E_1 = 23 \\ SE_\tau(6) = min(SE(\tau_1, 6), SE(\tau_2, 6)) = 23 > 0 \end{cases}$$

We abide by two conditions of three to execute Ap_1, since system slack energy and the energy reservoir level are positive. However, slack time must be computed in the way we explain with the dynamic EDL (see Chapter 1). If slack time is also positive, Ap_1 is formally authorized to execute (Figure 5.1).

5.5 Illustration of SSP

Example 12 *As an example, we consider a set of 2 periodic tasks that we studied in the previous chapter. Suppose that the first aperiodic job Ap_1 has computation time 1, energy consumption of 5 energy units, and is released at $t = 9$. Another aperiodic task with computation time 3 and energy consumption 15 energy units, is released at $t = 18$.*

At time 0, the residual capacity of the storage unit is maximum since the storage is full. τ_1 is the highest priority task which finishes at time 4 and consumes 18 energy units. At time 4, the residual capacity is given by $E_{max} - E_1 + P_p * C_1 = 8$. Now, τ_2 has the highest priority. It executes completely until time 7 and consumes 18 energy units. The residual capacity equals 2 energy units.

From $t = 7$ until $t = 9$, the processor remains idle and the energy level at $t = 9$ is $E(9) = 10$ energy units.

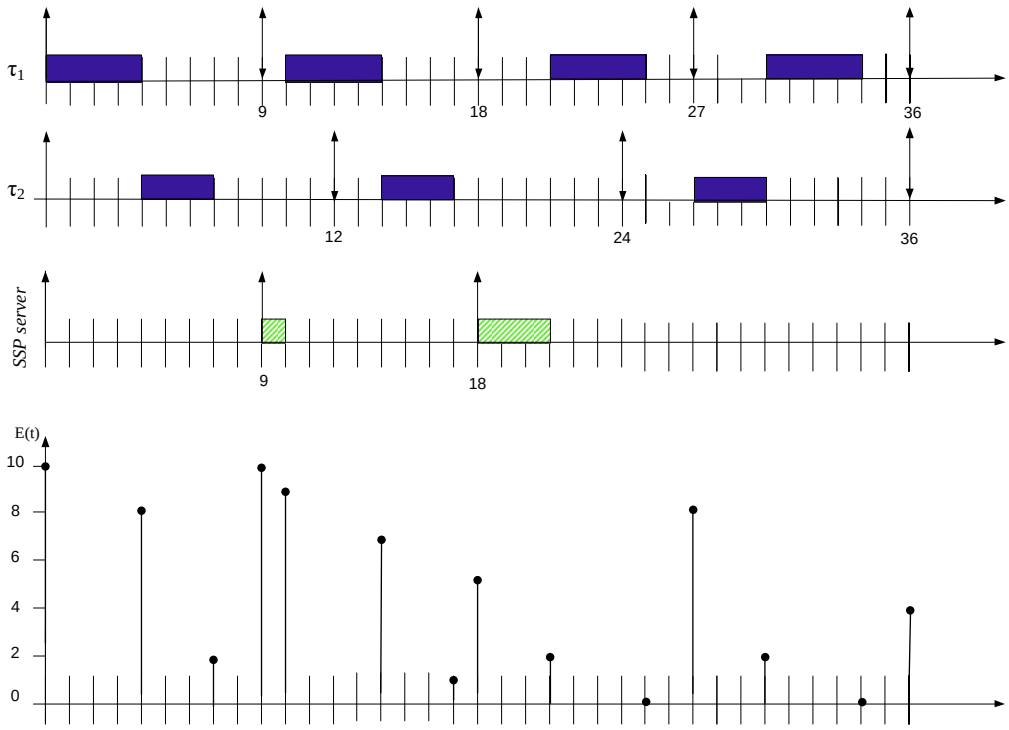


Figure 5.2: Aperiodic servicing with the slack stealer SSP.

At time 9, Ap_1 is released. As the storage unit is not empty, the slack time is positive (equal to 5) and the slack energy is positive (equal to 4), Ap_1 is authorized to execute and to consume a maximum of 5 energy

units. After its execution, the residual capacity falls at 9. At time 10, according to the ED-H scheduler, we continue to schedule the tasks till time 18 where the aperiodic task Ap_2 is released. Here, we have to check again if we abide by the three conditions: 1) the reservoir is not empty (5 energy units), 2) the slack time is strictly positive, equal to 5 and 3) the slack energy is strictly positive and equal to 23. Consequently, Ap_2 is authorized to execute immediately for executing during 3 units of time and for consuming 15 units of energy. Periodic tasks execute according to ED-H till the end of the hyperperiod where the energy reservoir contains 4 energy units, as illustrated by Figure 5.2.

We notice that the aperiodic tasks Ap_1 and Ap_2 are executed at the earliest time instant regarding processor and energy availabilities whilst the periodic tasks are deferred as much as possible. We observe that the response times of the aperiodic tasks Ap_1 and Ap_2 are 1 and 3 units of time, respectively, which is a clear evidence of minimizing the aperiodic responsiveness.

Moreover, in comparison with the background servers described in the previous chapter, we notice the following: the response time of Ap_1 is 95% and 87% shorter in comparison to BES and BEP, respectively. This important gain confirms the optimality of SSP.

5.6 Optimality Analysis

Optimality of SSP is stated in the following theorems.

Theorem 9 *All periodic tasks meet their deadlines when scheduled according to ED-H with the slack stealer SSP for aperiodic servicing.*

Proof: We prove the theorem by contradiction. Suppose that a job, say J_1, issued from a periodic task misses its deadline at d_1. And d_1 is the first deadline that is missed in the schedule. This violation is due to one of the two following reasons: The time starvation case is when deadline d_1 is missed with the energy reservoir that is not exhausted at d_1. The energy starvation case is when the reservoir is exhausted at d_1 and J_1 is not completed. As the periodic task set is feasible, the deadline violation necessarily comes from the execution of aperiodic tasks. Let t_0 be the latest time instant before d_1 where an aperiodic task, say Ap_0 executes. By definition of the slack stealer, Ap_0 was authorized to execute within $[t_0 - 1, t_0)$ because $ST(t_0 - 1) > 0$ and $SE(t_0 - 1) > 0$. And Ap_0 stops execution at t_0, because either the system has no more slack time i.e. $ST(t_0) = 0$ or the system has no more slack energy i.e. $SE(t_0) = 0$. Let us examine the two cases.

Case 1: $ST(t_0) = 0$

The slack time, $ST(t_0)$ as computed at t_0 with (3.2), gives the maximum processing time that could be made available from time t_0, while still guaranteeing the deadlines of all the jobs issued from the periodic tasks ready at or from time t_0.

The condition $ST(t_0) = 0$ guarantees that if the jobs are executed from time t_0 according to the earliest deadline first rule, all periodic jobs can be completed by deadlines even if one of these jobs is completed exactly at deadline. This contradicts that d_1 is violated.

Case 2: $SE(t_0) = 0$

The slack energy, $SE(t_0)$ as computed at t_0 with (3.4), gives the maximum energy surplus that the system could consume instantaneously at t_0, while preventing an energy starvation for all the jobs issued from the periodic tasks ready at or after time t_0. From t_0 to d_1, no energy is wasted (definition of ED-H) and all the jobs that execute within $[t_0, d_1)$ are periodic ones. Consequently there is no energy starvation, which contradicts the deadline violation at d_1 with $E(d_1) = 0$.□

Theorem 10 *For any periodic task set scheduled according to ED-H and a stream of aperiodic tasks processed in FCFS order, the slack stealing algorithm SSP minimizes the response time of every aperiodic task, amongst all algorithms which are guaranteed to meet all deadlines.*

Proof: We prove the theorem by showing that any alternative algorithm, A, which results in a shorter response time for any aperiodic task cannot guarantee that the deadlines of all the periodic tasks will be met. Let Ap_0 be the first aperiodic task which has a shorter response time when scheduled by algorithm A. As Ap_0 is the first such task, the response times of all previously serviced soft tasks must be the same as, or longer than when scheduled by the slack stealer. Once at the head of the queue, Ap_0 is serviced by the dynamic slack stealer so long as $SE(t) > 0$ and $ST(t) > 0$. For algorithm A to result in a lower response time, it must process Ap_0 for at least one clock tick when the slack stealer is unable to do so. We denote the time at which this occurs by t_0. The slack stealer computes two data at time t_0. The first one is the slack time i.e. the spare processing time which may be stolen. The second one is the slack energy i.e. the spare energy which may be stolen. One of these two data is zero at time t_0.
First case: $ST(t_0) = 0$.
Hence, for at least one job of periodic task, say J_1, we have $ST_{J_1}(t_0) = 0$. In servicing aperiodic task Ap_0 from t_0 to $t_0 + 1$, algorithm A has therefore lead to $ST_{J_1}(t_0 + 1) = -1$ culminating in the impossibility to complete job J_1 by its deadline. Algorithm A cannot, therefore, guarantee that the deadline of job J_1 will be met.

Second case: $SE(t_0) = 0$.
Hence, for at least one job of periodic task, say J_1, we have $SE_{J_1}(t_0) = 0$. It means that any additional energy consumption between t_0 and $t_0 + 1$ leads to $SE_{J_1}(t_0 + 1) < 0$ culminating in insufficient energy to execute job J_1 entirely by its deadline. Algorithm A cannot therefore guarantee that the deadline of job J_1 will be met due to energy starvation.□

5.7 Implementation and overhead considerations

The complexity of SSP is $O(m.n)$, where m is the number of iterations and n is the number of periodic tasks. We note that the number of iterations, m, depends on the periods and deadlines of the hard deadline tasks, thus the complexity of algorithm is pseudo-polynomial. Compared to the background-based servers, SSP suffers from high time overheads resulting from repeated slack time and slack energy calculations, as well as from high preemptions number. These dynamic computations are duplicated as long as there are aperiodic tasks in the list. However, additional processing time and energy are exploited in the optimal slack stealing approach, whenever possible, for the aperiodic activities, by delaying the execution of periodic activities. This process, called procrastination, shows a trade-off between efficiency and complexity of the SSP.

5.8 Conclusion

SSP, a new optimal server for enhancing average response time for aperiodic tasks is the central contribution of this thesis. SSP is particularly adapted to a dynamic real time energy harvesting platform where periodic tasks are scheduled according to the optimal ED-H algorithm.
The SSP server profits from energy surplus as well as from processing time surplus to execute aperiodic tasks with optimal responsiveness as it was proved theoretically. Nevertheless, let us notice that this theoretical performance evaluation does not consider the runtime overheads which are incurred by the online computations of the so-called slack time and slack energy values. And necessarily, such overheads will influence the actual performance of SSP.
In the next chapter, experimentations will point out the effective performance of this new slack stealing server in comparison to background servers.

CHAPTER 6

PERFORMANCE EVALUATION OF THE APERIODIC TASK SERVERS

Summary

This chapter is devoted to compare and evaluate by simulation the performance of the aperiodic servers that we described in chapters 4 and 5. We want to show to what extent SSP constitutes a contribution to the problem of minimizing aperiodic responsiveness. In order to show its advantages and limits, we will systematically compare the SSP server against the two background base servers. In that objective, we will consider different application profiles in terms for example of processing loads and energy limitations. The experimentation aims in addition to exhibit the influence of different parameters (i.e. capacity of the energy storage unit, power produced by the environmental source) on the response time of soft aperiodic tasks. That is why, several simulations have been performed by changing these parameters.

Contents

6.1 Description of the performance analysis

Let us recall that the ED-H scheduler is chosen to schedule the periodic tasks. Aperiodic tasks are served according to the FCFS policy. In that section, we present the simulation environment. We introduce the metrics applied to evaluate the performance of the SSP, BEP and BES algorithms.
We made several assumptions:

- the total processing load U_p incorporates 50% of the periodic processor utilization U_{pp} and 50% of the aperiodic utilization U_{ps}.
- Identically, the total energy load U_e includes 50% of the periodic energy utilization U_{ep} and 50% of the aperiodic energy utilization U_{es}.

The results of simulations are carried out as a function of the total energy/processing utilization and/or aperiodic utilization.
We will show the behaviour of each aperiodic task server under different perspectives including average response time of aperiodic tasks, normalized response time jitter of aperiodic tasks, normalized input-output latency of aperiodic tasks, number of preemptions, and overhead.

It is worth noting that the objective is to improve mean response time, jitter, and latency of the soft aperiodic tasks without jeopardizing schedulability of the periodic tasks with the lowest possible implementation costs i.e. minimum number of preemptions and minimum number of computing operations.

In this perspective, we made two different sets of experiments. In the first one, the incoming source power is assumed constant while it is variable in the second set of experiments.

6.1.1 Simulation Environment

We developed a simulator in Matlab. The resulting two-dimensional and three-dimensional graphs are plotted with high resolution. Our code is able to produce any simulation with given parameters that are specified by the user.

To design the *task generator of periodic tasks*, the simulator receives the number *n* of desired periodic tasks, the hyper-period H, the assigned periodic processing utilization U_{pp}, and periodic energy utilization U_{ep}. The simulator infrastructure automatically generates a periodic task set τ of quadruples $(C_i, E_i, D_i, T_i) \mid 1 \leq i \leq n\}$. Periods and computation times are distributed uniformly in discrete time steps, depending on $U_{pp} = \sum_{i=1}^{n} \frac{C_i}{T_i}$. 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 remain feasible in terms of processing time and energy i.e. $U_{pp} \leq 1$ and $U_{ep} \leq P_p$ where P_p is the recharging power.

A *task generator of aperiodic tasks* is also designed. Be given the number of desired tasks *m*, the aperiodic processing utilization factor U_{ps} and the aperiodic energy utilization U_{es}, a stream of aperiodic tasks is generated according to a uniform distribution by simulating a poisson aperiodic arrival.

A simulation run consists of one task set composed of n = 20 periodic tasks. To reduce the bias effect of random generation procedure, simulations are performed throughout 10 hyperperiods. Each point in the curves corresponds to 100 runs. The mean value is computed from the results obtained at each run. The energy reservoir is assumed to be initially full which is the hypothesis always retained in the literature. That is the amount of energy available at time zero is the capacity of the storage unit.
The value of this capacity has been chosen so as to equal E_{min}, defined as the minimum size of the reservoir that permits to guarantee the system feasibility.

The energy produced by the source is not controllable and not necessarily a constant value. Thus, the recharging power P_p is considered constant in the experiments in Section 6.2. Thereafter, it is considered as variable in the experiments in 6.3. To avoid short-term energy shortages, we assume to have an accurate

estimation of short-term energy within some prediction errors margin. In summary, four different energy profiles are 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 6.1). The different waveforms produced are periodic. It is worth mentioning that E^p_{min} of each profile p should not be less than the area A_p.

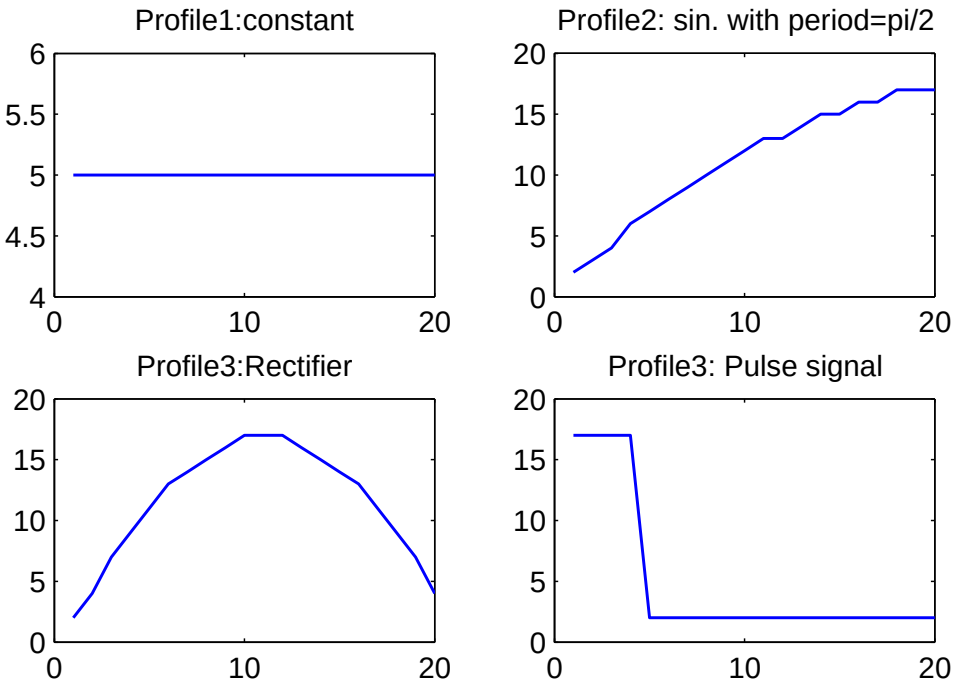


Figure 6.1: Energy source profiles under study.

6.1.2 Evaluation metrics

In our study, we address the following metrics:

1. **Average response time of aperiodic tasks**: It is the average duration taken by an aperiodic task from its arrival time until its finishing time ($R_i = f_i - a_i$), normalized with respect to the average aperiodic computation time. It is defined as:

$$ART_i = \frac{f_i - a_i}{c_i} \tag{6.1}$$

   A value of 10 on the y-axis means an average response time 10 times longer than the task computation time.

2. **Average response time jitter of aperiodic tasks**: Real-time systems, especially software control systems, are developed to meet the requirements of real-time automation systems. One such crucial requirement is reducing the delay and jitter of tasks in such systems. Jitter represents the induced offset between the release time of the aperiodic task and its start of execution ($JT_i = s_i - a_i$). Therefore, we evaluate the average jitter which is normalized with respect to its response time.

$$AJT_i = \frac{s_i - a_i}{f_i - a_i} \tag{6.2}$$

   Hence a value of 1 on the y-axis corresponds to a jitter equal to its response time; a value of 0 corresponds to the minimum achievable jitter time and means that the task was immediately serviced without being blocked.

3. **Average input-output latency of aperiodic tasks**: There are many factors that influence the latency of a task. A task can be usually interrupted either by a leakage of energy or by a task of highest priority (preemption). These interruptions could invoke a time dilation and therefore a delay of tasks execution and/or deadline violation. Latency allows us to confine the total time that elapses between the start time of execution of a task and the time of its effective end ($LT_i = f_i - s_i$) to guarantee the relevance of the final results. Its average normalized value is given by:

$$ALT_i = \frac{f_i - s_i}{c_i} \tag{6.3}$$

   The normalized latency of a job is 1 if the job is not preempted at all.
4. **Preemption task rate**: It represents the runtime overhead due to context switches. It is defined by the ratio of the number of preemptions per the number of processed jobs. Preemption occurs when the execution of a task is interrupted in favor of a higher priority task. The execution of the preempted task is then resumed later in time.

$$PE = \frac{number\,of\,preemptions}{total\,number\,of\,jobs} \tag{6.4}$$

5. **Overhead**: Overhead is the time taken by the kernel in performing a service on behalf of a specific task, such as invoking, resuming, or terminating it. In case of our scheduling algorithms, overhead occurs for example when the operating system has to compute on-line either the slack time or the slack energy. Thus, overhead depends on the complexity for computing slack time and slack energy and frequency of these computations along time in the SSP aperiodic server. In contrast such variables do not need to be computed in the BES server.

6.2 First set of experiments: constant energy profile

6.2.1 Motivations

The most important item in energy harvesting systems is surely the harvester, and the most common one is a solar cell. The electricity generated by the harvester needs to be converted into a useful voltage or current to power the system. Taking full advantage of the solar power may be the most convenient in wireless sensors usage. Surely, outdoor solar power is time and season dependent. However, the power which is produce is stable in that sense it does not change every milli-second. Consequently, whatever indoor or outdoor, the dynamics of variation of solar energy is very low in comparison to the periods of the sensing tasks in most of applications. The consequence is that we may assume that for large periods of time, the source power is constant.

Another example of constant source power is thermal energy. Energy harvesting is also a key technology to enable self-sustained wearable devices in medical applications. Thermoelectric generators (TEG) for scavenging of human body heat are today a promising option because of their independence of light conditions and the activity of the wearer. These harvesters can power a lot of different wearables such as a multi-sensor bracelet that measures activity, acquires images and displays results. The human body offers a constant heat source because typically a constant temperature difference exists between the body core and the environment. Even when the wearer is not in movement and situated in a dark room during sleep for example, energy can be produced. It is clear that lower ambient temperatures or increased activity of the wearer will drastically increase the amount of accumulated energy. But here too, we may assume that these change in power production does not appear with high frequency.

In summary, there are many energy harvesting devices which may profit from stable power generation characteristics of the environmental source. That is why, we may consider two types of profiles for energy production: constant and variable.

6.2.2 Experiment 1: Average response time of aperiodic tasks

6.2.2.1 Varying U_e/P_p

Aperiodic responsiveness is measured for three processing load profiles: 1) weakly constrained with $U_p = 20\%$, 2) fairly constrained with $U_p = 40\%$ and 3) highly constrained with $U_p = 80\%$. U_e/P_p varies from 5% to 100% in order to show the impact of energy availability on aperiodic responsiveness. Results are reported in Figures 6.2 and 6.3, respectively.

As expected, the SSP server outperforms the two background policies BES and BEP for all configuration settings.

It is worth mentioning that the higher the energy limitation, the wider the performance of SSP over the background techniques. BES shows inferior performance for high energy requirements since aperiodic tasks may execute only when the reservoir is fully replenished. BES and BEP behave similarly when renewable incoming energy is greatly available in comparison to energy requirements.

For the first experiment, (Figure 6.2, $U_p = 20\%$), we examine a system which is softly constrained by processing utilization. In other terms, the system is often idle. We can see that the Slack Stealer SSP has aperiodic response time which is at least 25% lower compared to background servers for all energy conditions.

For U_p = 80% (Figure 6.3), the SSP server benefits from time slack stealing to optimize the processor utilization and performs much better than background servers. They both behave poorly even when there is no energy limitation. When the system is highly constrained both in terms of time and energy, the performance of the slack stealing based server approaches that of the background servers.

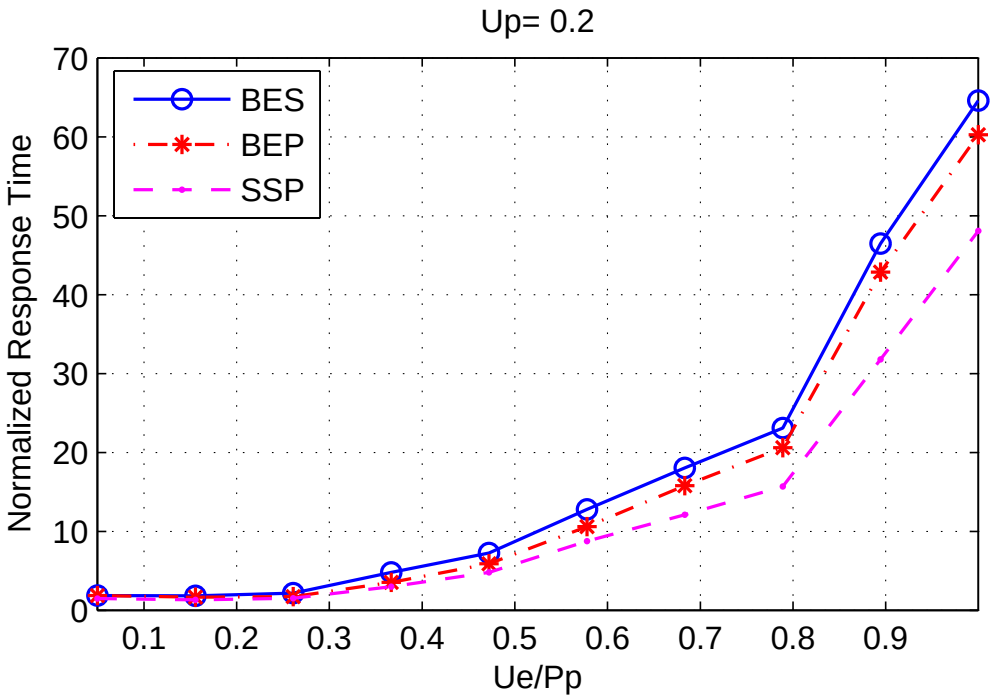


Figure 6.2: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.2.

6.2.2.2 Varying U_p

This set of experiments includes two simulations which show the performance of the algorithms as a function of the total processing utilization U_p for two energy utilization ($U_e/P_p = 0.2$ and $U_e/P_p = 0.8$). Figures 6.4 and 6.5 show that the background service strategies offer higher response times than SSP. When the system has low energy constraints (i.e. $U_e/P_p = 0.2$), the response times achieved by BES and BEP are close to one for a lower total load, meaning that aperiodic tasks execute immediately. When the total

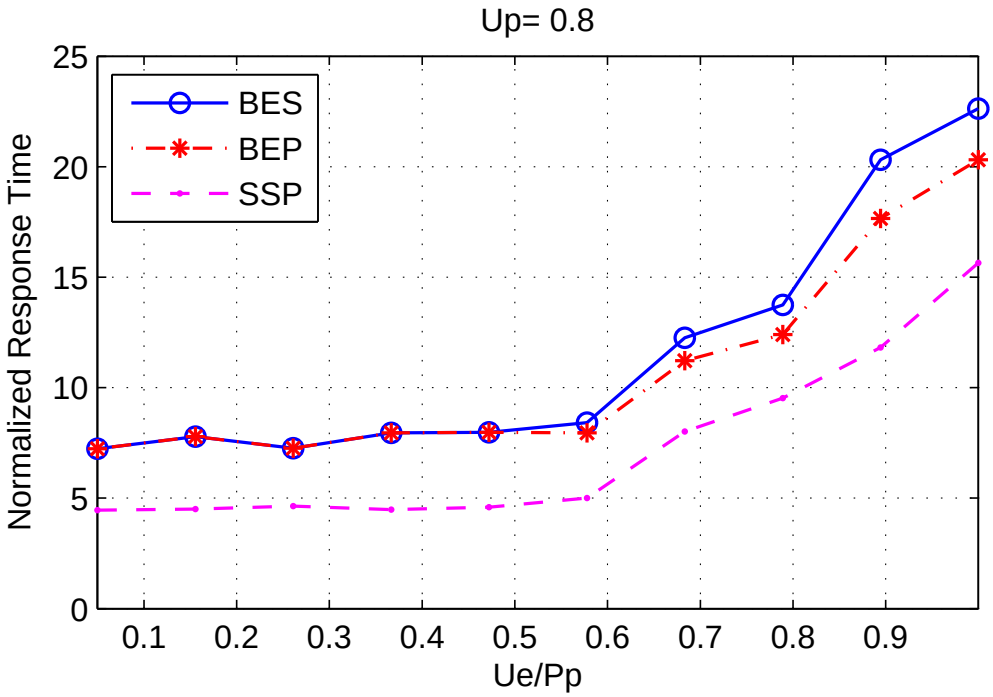


Figure 6.3: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.8.

load increases, the two background servers perform poorly and have a similar performance. From U_p = 0.5 until U_p = 1, the Slack Stealing server outperforms the other algorithms. It has a normalized response time which is reduced by more than 15 % with respect to the Background curves.

Figure 6.5 shows the case in which $U_e/P_p = 0.8$. The normalized response times of BES and BEP increase with the increase of U_p. BEP services the aperiodic tasks 1.5 times faster than BES and converges gracefully to BES as U_p increases. The execution of aperiodic tasks is done as long as there is always sufficient energy for future periodic tasks. The SSP can still provide a significant reduction in aperiodic responsiveness compared to the background services regardless of the increase of the energy constraint. With the variation of U_p, the response time of SSP is at least 30% less than BES and BEP.

The previous experiment proves that the SSP server exhibits the best performance in reducing the aperiodic responsiveness by taking advantage of the energy slack stealing.

6.2.3 Experiment 2: Average jitter of aperiodic tasks

In this section, the experiment aims to show how much SSP reduces delays and jitters of aperiodic tasks in energy constrained real-time systems, which are enforced by the operating system, control tasks, kernel mechanisms, etc.

Many experiments were performed. To limit the number of graphs, we only present two experiments where SSP is evaluated as a function of the total energy utilization U_e/P_p and is compared with the background policies in terms of jitter for a fixed total processing utilization. The results of both simulations are shown in Figures 6.6 and 6.7 for $U_p = 0.2$ and $U_p = 0.8$ respectively.

We observe that the SSP server reduces the delay and jitter of the aperiodic tasks in comparison with the background servers. The results in Figure 6.6 show that the jitter of SSP is at least 14% lower than BEP and BES.

In case of $U_p = 0.8$ (Figure 6.7), the results show that when the total energy utilization is low, BES and BEP have similar performance and the difference between them increases for a large range of energy

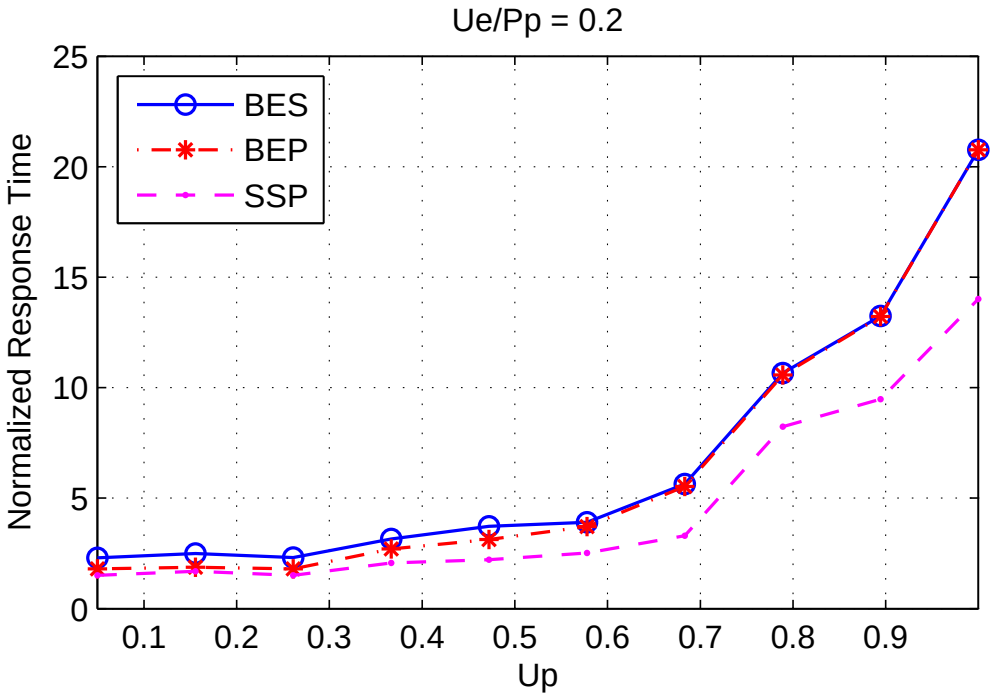


Figure 6.4: Normalized aperiodic response time with respect to U_p, for U_e/P_p=0.2.

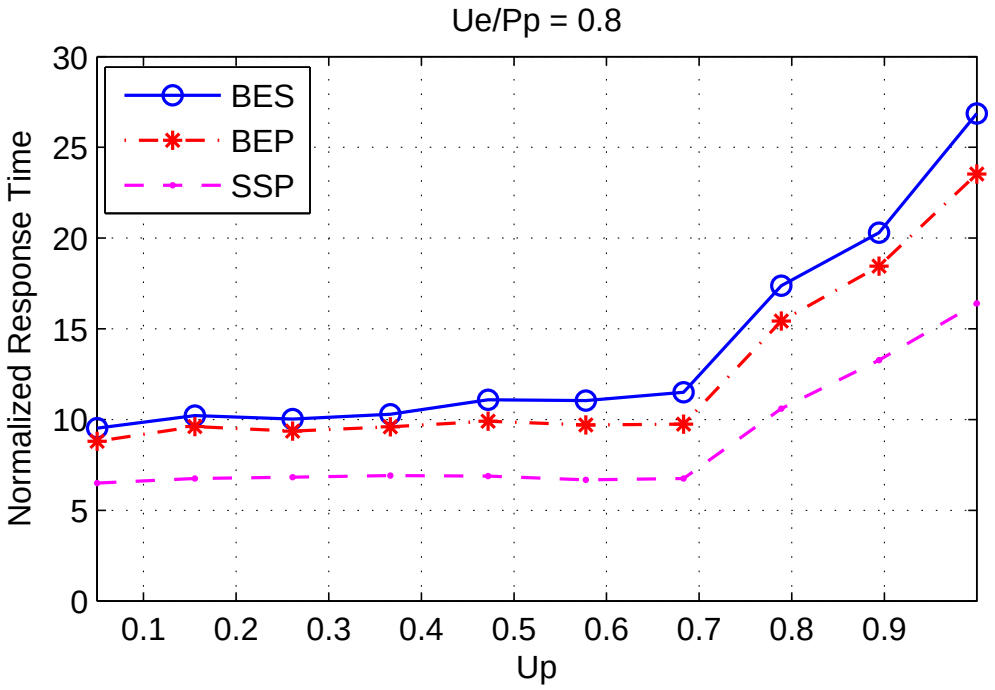


Figure 6.5: Normalized aperiodic response time with respect to Up, for U_e/P_p=0.8.

utilization. The large range of energy utilization confirms that SSP is more effective (13% lower jitter time) and can guarantee the prompt service of aperiodic tasks when the system is adopted to high processing and energy constraints due to its stealing concepts.

6.2.4 Experiment 3: Average latency of aperiodic tasks

In this experiment, we compare the latency of the optimal SSP algorithm versus the two background mechanisms (BEP and BES) as a function of the total energy utilization. The two graphs shown in Figures 6.8 and 6.9 correspond to two different total processing utilizations, low and high, as addressed above.

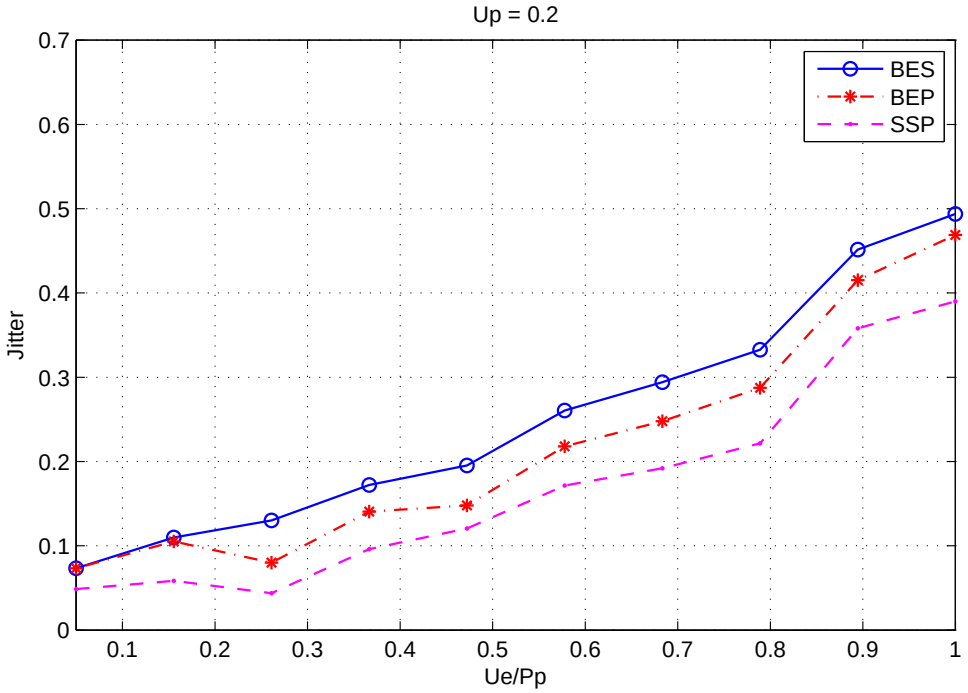


Figure 6.6: Normalized jitter time with respect to U_e/P_p, for U_p=0.2.

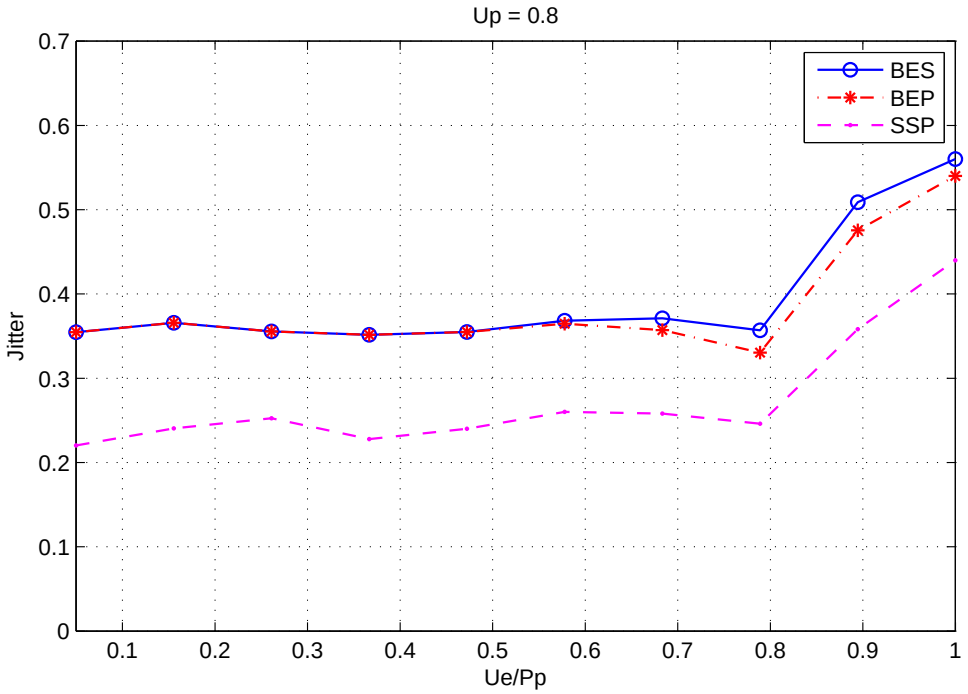


Figure 6.7: Normalized jitter time with respect to U_e/P_p, for U_p=0.8.

We observe from each graph, that the SSP algorithm can provide a significant reduction in latency time compared to BES or BEP service (lower at least 6%). The performance of BES and BEP depends on the energy utilization. For low energy utilization, BES performs as well as BEP, but as the total energy utilization increases, their performance tends to be different from each other showing that BEP outperforms BES by a 11% deviation.

This implies that SSP has the optimal deviation time relative to the arrival time of the aperiodic task, and the lowest number of interruptions that may occur from the start to the end of its execution due to energy

shortage or highest priority periodic task preemption.

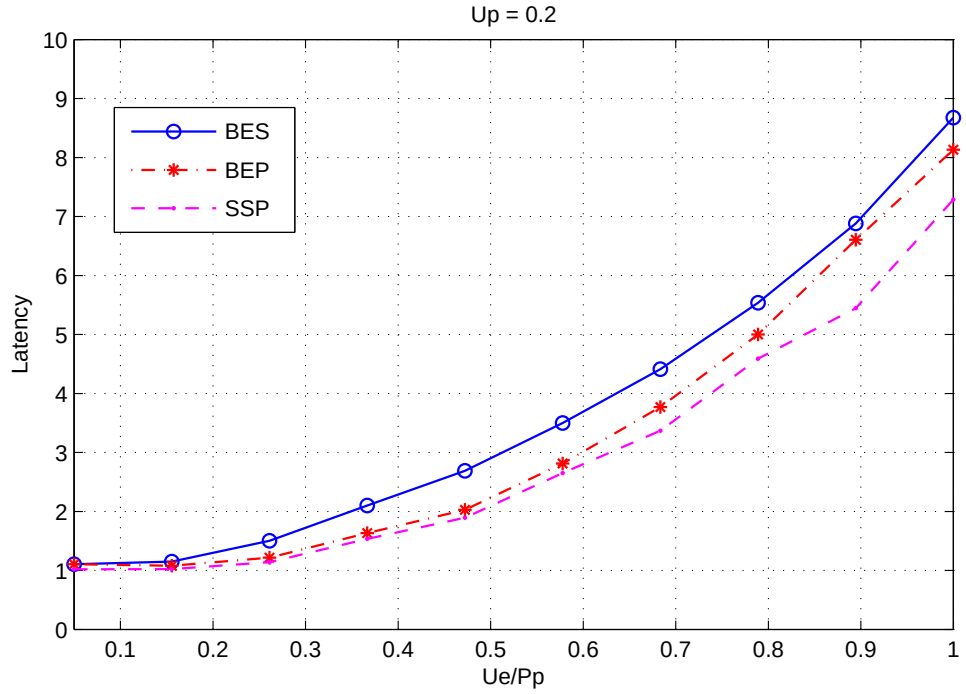


Figure 6.8: Normalized latency time with respect to U_e/P_p, for U_p=0.2.

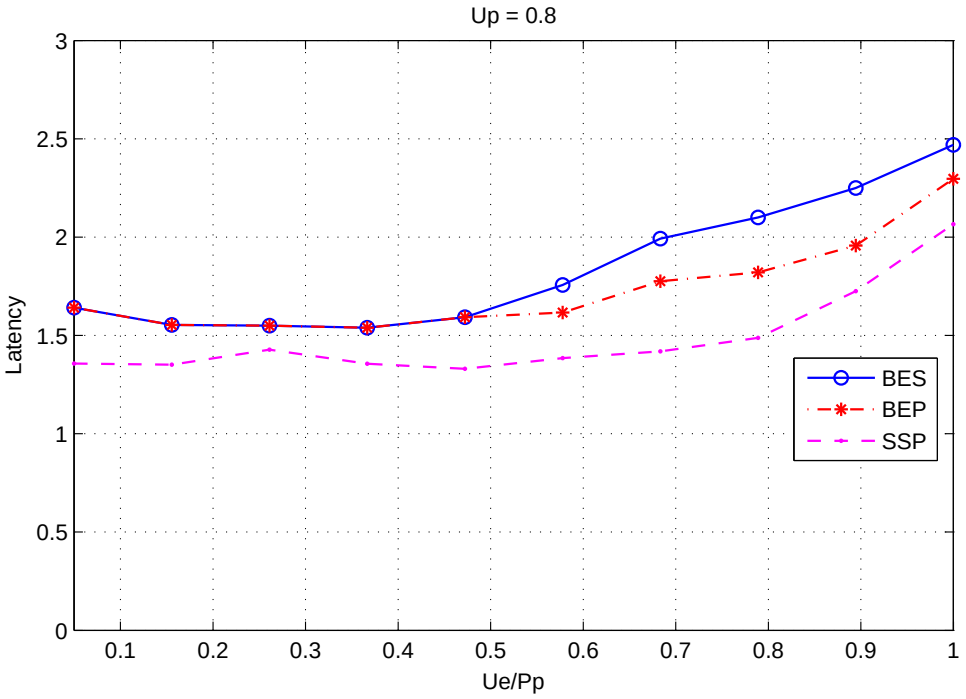


Figure 6.9: Normalized latency time with respect to U_e/P_p, for U_p=0.8.

6.2.5 Experiment 4: Relative performance with different reservoir sizes

In this set of experiments, we evaluate the performance of the servers by varying the reservoir size with E_{min}, 5*E_{min}, and 9*E_{min}. E_{min} is the minimum size of the reservoir that guarantees time and energy

feasibility, for given U_p, U_e and P_p. Here, we report the results for systems which are not time-constrained i.e. $U_p = 0.2$. In the Table 6.1, the 3rd, 4th and 5th columns give the aperiodic responsiveness of BEP, BES and SSP servers, respectively, for two profiles in terms of energy constraints.

Table 6.1 shows that the SSP server achieves significant reduction in aperiodic responsiveness, comparing with BEP and BES servers under all parameter settings. BEP achieves low aperiodic response time compared to BES. For example, when the system uses 20% of available energy with minimum reservoir size, the response time under SSP is 19% and 29% lower compared to BEP and BES respectively. If the energy requirement is set to 80%, all servers record relatively high response times. However, the optimal slack stealer still beats the background servers by a large difference due to optimal exploitation of slack energy.

For each of the three servers, higher is the size of the reservoir, lower is the normalized aperiodic response time for a given energy setting. When the reservoir size is set to E_{min} and the system uses 80% of available energy, the BES, BEP, and SSP servers have aperiodic response time respectively equal to 37.4, 35.2 and 26.2. When increasing the reservoir size to $9 * E_{min}$, the response time of BES, BEP and SSP will be respectively reduced by 64%, 75% and 76%. Such a significant improvement in aperiodic responsiveness comes from possible immediate service through extra energy which is available in the reservoir. We can see that the BES algorithm achieves the lowest reduction in response time over all the servers. It is because under BES, aperiodic job executions have to wait for the energy reservoir be fully replenished.

Table 6.1: Relative performance with different reservoir sizes

Reservoir Capacity	U_e/P_p	BES	BEP	SSP
E_{min}	0.2	2.4	2.1	1.7
	0.8	37.4	35.2	26.2
$5 * E_{min}$	0.2	2.0	1.7	1.4
	0.8	23.0	15.8	14.7
$9 * E_{min}$	0.2	1.5	1.3	1.1
	0.8	13.4	8.7	6.3

6.2.6 Experiment 5: Impact of the harvested power and the reservoir capacity on the responsiveness

The system can be rich in energy or deficient, due to harvesting unit status and the nature of energy source.

In this set of experiments, the impact of the harvested power P_p and the capacity of storage E_{min} on the aperiodic responsiveness of SSP is studied. For that purpose, we evaluate the proposed algorithm with four different harvesting power settings extracted from Profile 2 in Figure 6.1: $P_p, 2 * P_p, 4 * P_p$ and $8 * P_p$; P_p is considered the lowest value. Also five different storage capacities sweeping from E_{min} to 9*E_{min} are considered. E_{min} being the minimum size of the reservoir that guarantees time and energy feasibility depends on the configuration parameters of each experiment.

Results recorded in Figures 6.10 and 6.11 show the plots of sweeping both harvest power and storage capacity for different processing and energy constraint settings. They are classified into four processing/energy constrained systems:
1)weakly processing constrained system ($U_p = 0.4$) and weakly energy constrained system ($U_e/P_p = 0.2$),
2)weakly processing constrained system ($U_p = 0.4$) and highly energy constrained system ($U_e/P_p = 0.8$),
3)highly processing constrained system ($U_p = 0.8$) and weakly energy constrained system ($U_e/P_p = 0.2$),
and 4)highly processing constrained system ($U_p = 0.8$) and highly energy constrained system ($U_e/P_p =$

0.8).

Before analysing the results, it is necessary to notice that, for presentation issue, the aperiodic response times on the y-axis are normalized as following:

$$RT = \frac{\sum_{i=1}^{m} \frac{f_i - a_i}{c_i}}{m} \tag{6.5}$$

Thus, a value of 1 on the y-axis corresponds to the shortest response time, and a value of zero to the worst response time.

From the graphs,

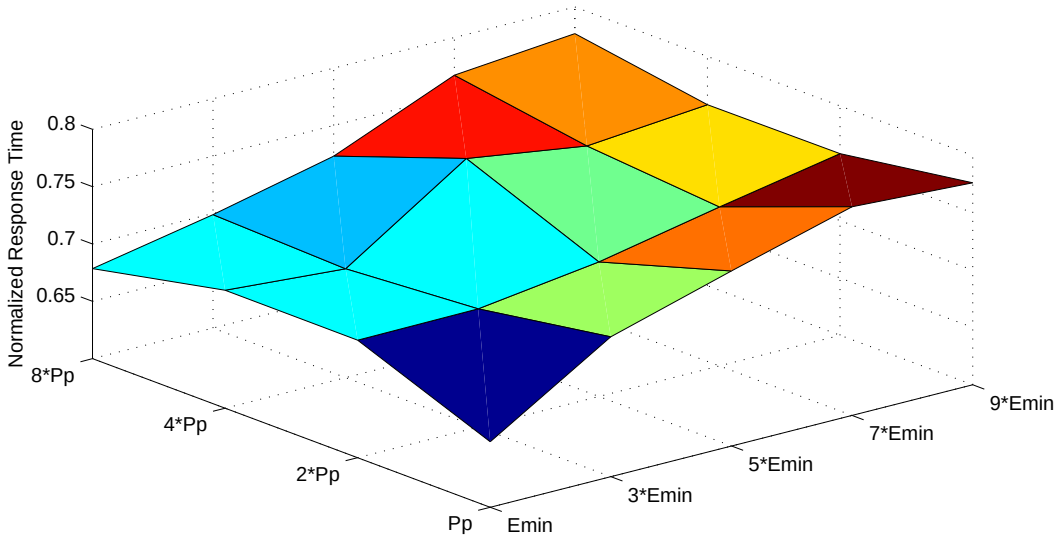
1. An important reduction of aperiodic responsiveness is noticed when the capacity and/or the power harvested increase. When the couple of capacity and/or power is much more significant, the response times become higher (close to one) for all configuration settings. And the most important increase occurs under higher power P_p and capacity E_{min} parameters.
2. The normalized response time increases in a significant and progressive way with the increase of the processing utilization U_p. For example, see Figures 6.10a and 6.11a, when U_p increases, the normalized response time declines by a factor of 32%.
3. The normalized response time increases significantly with the increase of the energy utilization. For example, see Figures 6.10a and 6.10b, when U_e increases, the normalized response time declines by a factor of 23%.
4. As the results show, all aperiodic tasks can be executed as soon as possible within their deadlines, without involving energy starvation for future periodic tasks due to the extra energy coming from the power source or storage unit.

6.2.7 Experiment 6: Task preemption rate

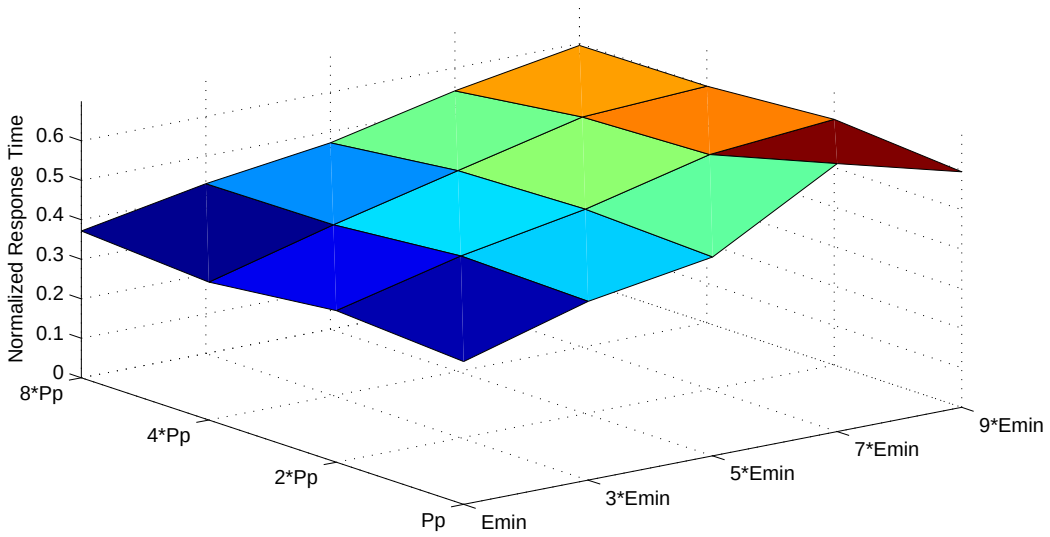
It is interesting because of practical consequences to compare the number of preemptions which are generated by the various algorithms. This is in order to properly evaluate the run time overheads due to context switching. The simulations presented below evaluate the number of preemptions which are generated by the three strategies BES, BEP, and SSP algorithms as a function of the aperiodic load U_{ps} for two total energy load settings: 1) low energy utilization ($U_{ep}/P_p = 0.1$, $U_{es}/P_p = 0.1$) in Figure 6.12 and high energy utilization ($U_{ep}/P_p = 0.4$, $U_{es}/P_p = 0.4$) in Figure 6.13.

In view of the results, we can say that globally, the SSP servicing algorithm induces a great number of preemptions with the variation of the aperiodic load U_{ps}. This is because an aperiodic task can preempt a periodic one when an aperiodic task occurs and the slack time is positive. And an aperiodic task can be preempted when the slack time is zero or the slack energy is zero. By way of illustration, for U_{pp} = 0.3 and U_e = 0.2, the average preemption rate generated by the SSP algorithm reaches a maximum value of 1.47% (Figure 6.12).

On the other hand, under the two background strategies, we observe that the preemption rate is independent from the aperiodic load applied to the system, i.e. the aperiodic tasks never preempt the periodic ones. Nevertheless, any running aperiodic task may be preempted whenever a periodic task releases. As shown in Figure 6.13, under SSP, the number of preemptions is higher than before because slack energy is lower and falls to 0 more often. For example, for U_{pp} = 0.3 and U_e = 0.8, the average preemption rate generated by the SSP algorithm reaches a maximum value of 1.6%, whereas the curves of BES and BEP are identical to the previous ones because background scheduling does not depend on energy.



(a) weakly energy constrained system



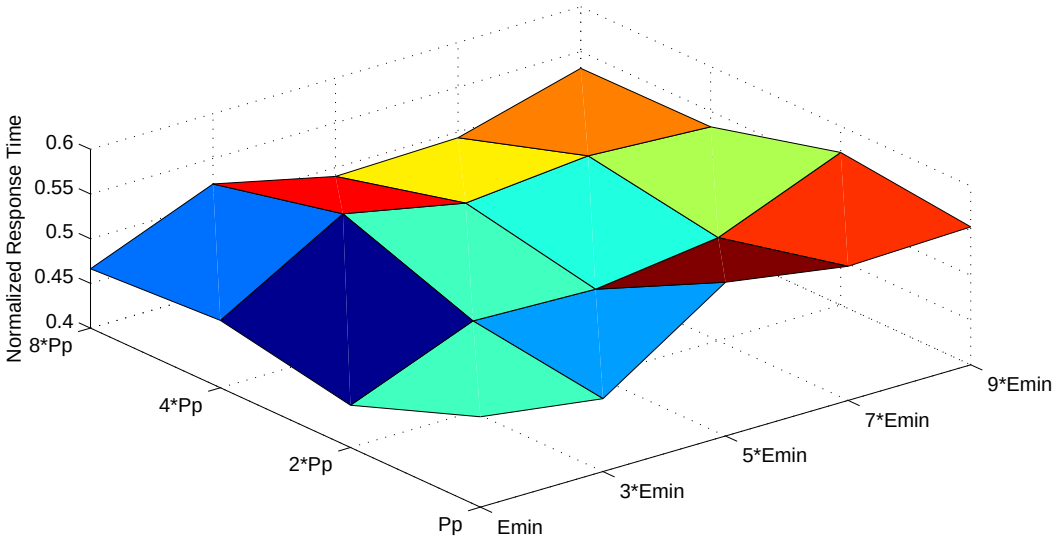
(b) highly energy constrained system

Figure 6.10: Impact of storage capacity and harvested energy on responsiveness of SSP for weakly processing constrained system.

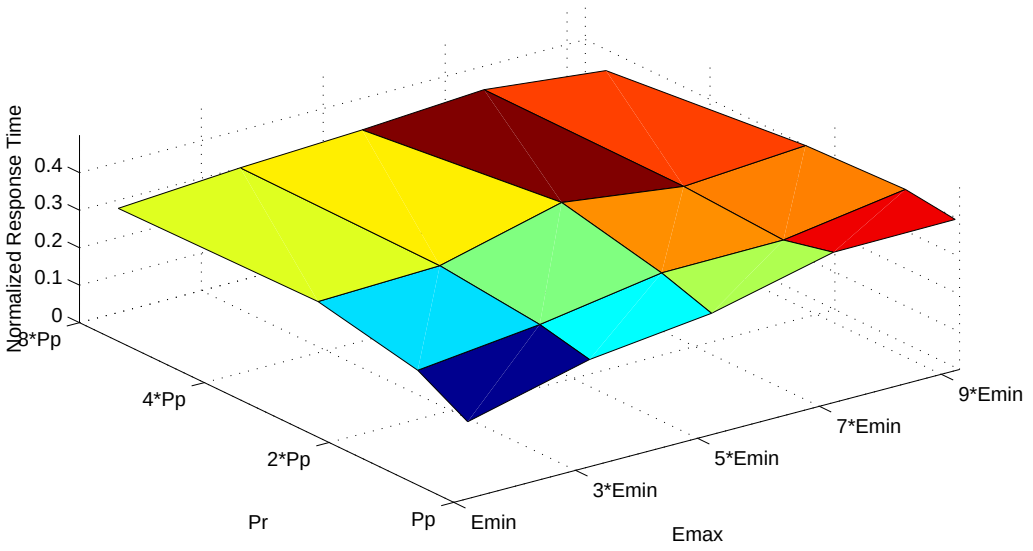
The previous set of experiments illustrates how the theoretical highest performance of the SSP server may be affected due to preemptions. The background-based servers are the most competitive in terms of preemption rate as long as the periodic load is low. The variation in the periodic load and the energy utilization affect the efficiency of the SSP algorithm: higher are these two parameters, more important is the performance deviation of SSP with respect to the two background servers.

6.2.8 Experiment 7: Overhead

We consider here one kernel cost: overhead. Overhead is the time taken by the kernel in performing a service on behalf of a specific task such as computing dynamic variables each time the scheduler is invoked.



(a) weakly energy constrained system



(b) highly energy constrained system

Figure 6.11: Impact of storage capacity and harvested energy on responsiveness of SSP for highly processing constrained system.

In some applications, system services are very often performed. And, they can take a considerable time in comparison with the execution times of the tasks. For that reason, overhead wields a particular influence upon the actual timing behaviour of tasks. And some scheduling mechanism may be more susceptible to a given parameter than another one. Overhead in the SSP server is incurred by on line computing/updating slack time and slack energy is a must. Since slack time under ED-H is computed when we recharge the battery, its overall impact increases as tasks increases their energy consumption. On the other hand, slack energy of the system under ED-H is computed every time we have to start the execution of a job within $[t, d_i]$ while higher priority jobs need to be processed in the future. It is worth mentioning that higher priority jobs concern both jobs which are ready at t but not completed at d_i and future jobs, with deadline less than or equal to d_i. Consequently, the global impact of slack energy increases as the number of jobs increases.

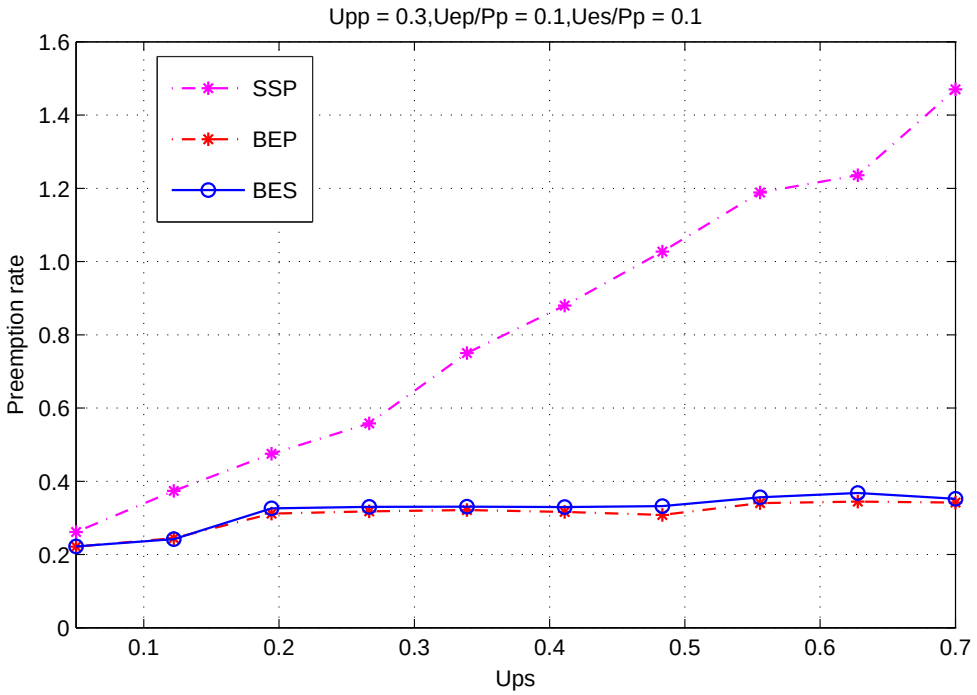


Figure 6.12: Preemption rate with respect to U_{ps}, for low energy utilization.

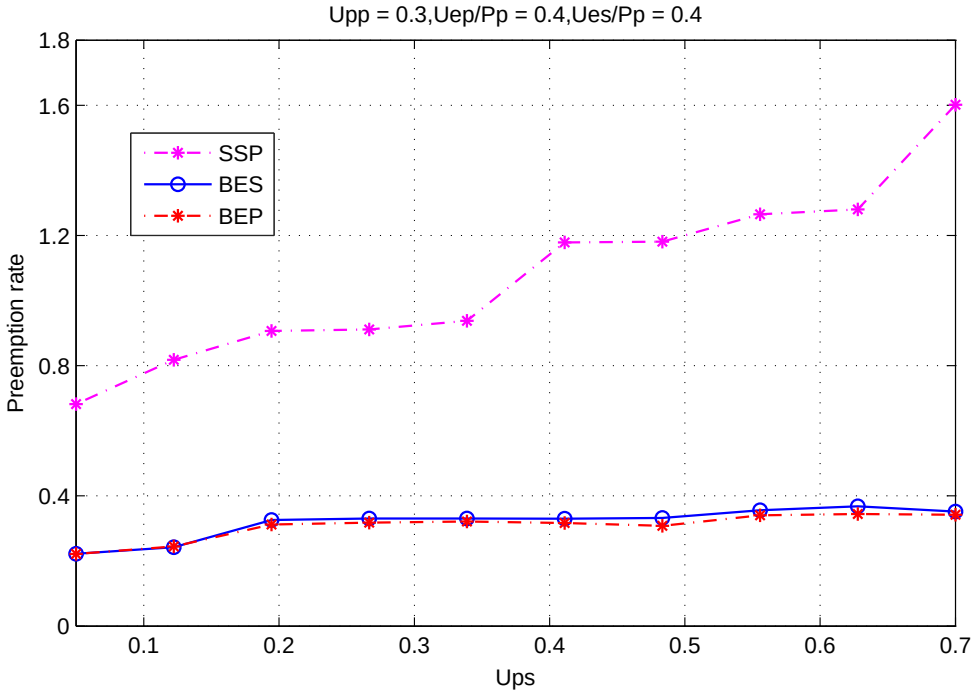


Figure 6.13: Preemption rate with respect to U_{ps}, for high energy utilization.

The overall overhead for the SSP server is explained by the total overhead due to on-line computations of slack time plus the total overhead due to on-line computations of slack energy.

6.2.8.1 Varying U_{ps}

The present section discusses this subject and testifies its practical importance by measuring the average overhead time which is normalized, for performance evaluation, with respect to the periodic and aperiodic processing loads. Such overhead is measured for the three algorithms BES, BEP, and SSP as a function of

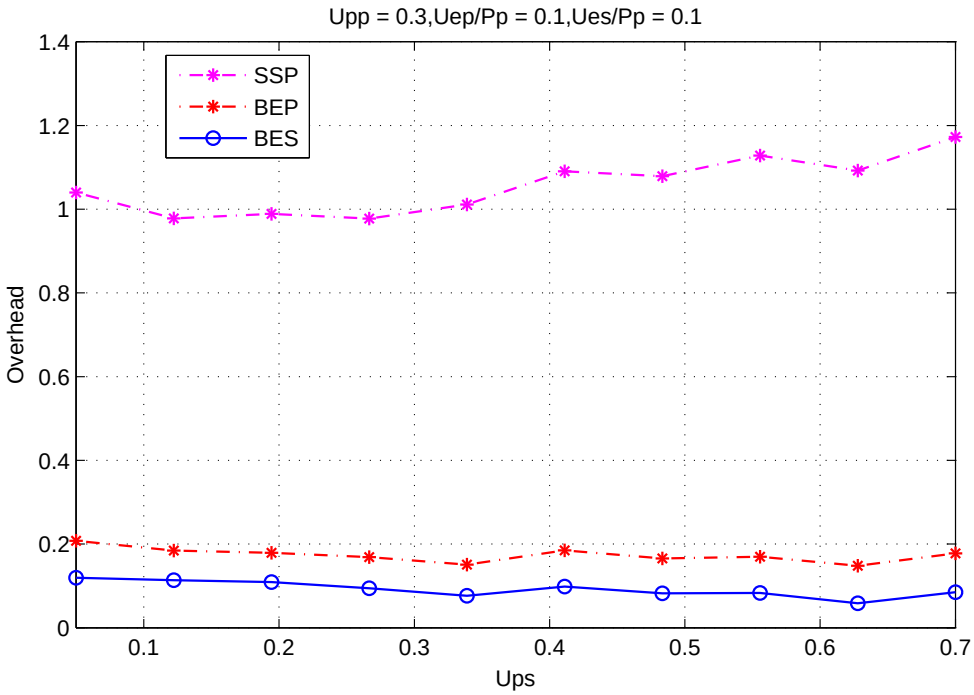


Figure 6.14: Time Overhead with respect to U_{ps}, for low energy utilization.

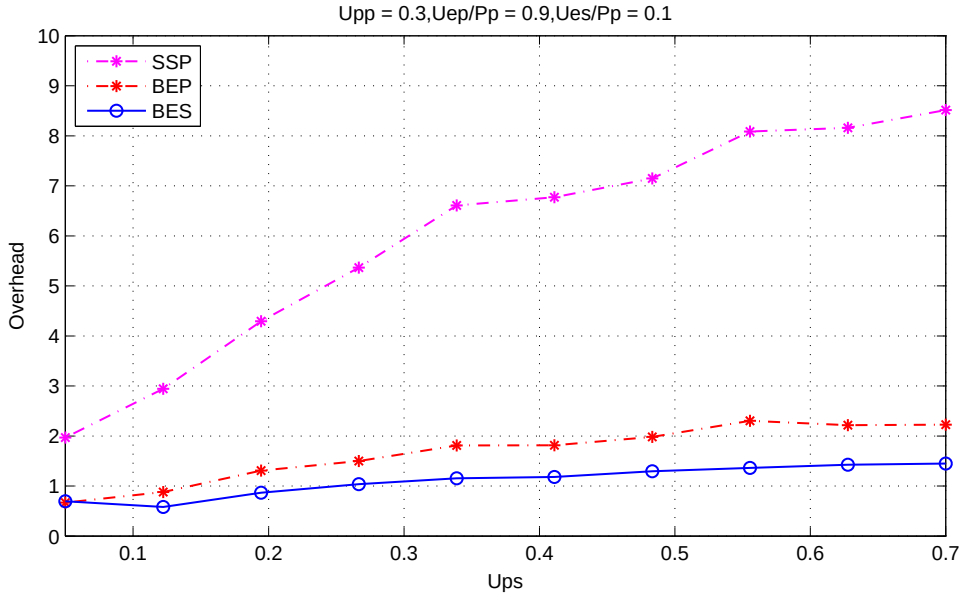


Figure 6.15: Time Overhead with respect to U_{ps}, for high energy utilization.

the aperiodic load, for a constant periodic load $U_{pp} = 0.3$ and two energy load settings: 1) weakly constrained with U_{ep}/P_p = 0.1 and U_{es}/P_p = 0.1 and highly constrained with U_{ep}/P_p = 0.9 and U_{es}/P_p = 0.1. Overhead results are depicted in Figures 6.14 and 6.15 respectively.

For all these tests, the power P_p received from the environment is assumed constant and equal to 5.

From Figure 6.14, it can be easy concluded that, under BEP and BES, as U_{ps} increases, slack time is approximately constant since we computed it each time the battery is deplenished. So, it depends on energy variation and not on the processing variation. Also, under the two background servicing approaches, slack energy is nearly constant because the higher priority task utilization, which is supposed the periodic load U_{pp}, is constant. Since the slack time is computed under BEP each time an aperiodic task occurs, its curve

is higher than that of BES.
Under SSP, as U_{ps} varies, the overhead due to slack energy and slack time happens each time a periodic/aperiodic task is ready to be executed. But since the periodic load is constant, the total overhead is considered at the variation of the aperiodic computation time. Thus, it slightly increases with the increase of U_{ps} and reaches a moderate average value of 1.16.

We observe in Figure 6.15 that when the periodic energy utilization U_{ep} increases, the performance of the background servers do not change and remain approximately constant for the same reasons mentioned above. But under SSP, with the increase of energy utilization, the total overhead of SSP exhibits a significant degradation (the overhead increases quickly to reach a high value, i.e. the time overhead of one worst case execution time worth 8.3 when $U_{ps} = 0.7$. Such deterioration can be explained as follows, comparing to the results in Figure 6.14: When the periodic energy utilization increases, the energy consumed by the periodic tasks increases. Therefore, the requirement of inserting an idle time becomes mandatory which increases significantly the overhead due to slack time. On the other hand, slack energy depends on U_e/P_p and is computed each time a job starts its run-time. This will increase the overhead due to slack energy. As a consequence, under a high energy factor, there is a higher impact on SSP behaviour than the Background servers, since the average time overhead increases by about 86% from low to high energy utilization.

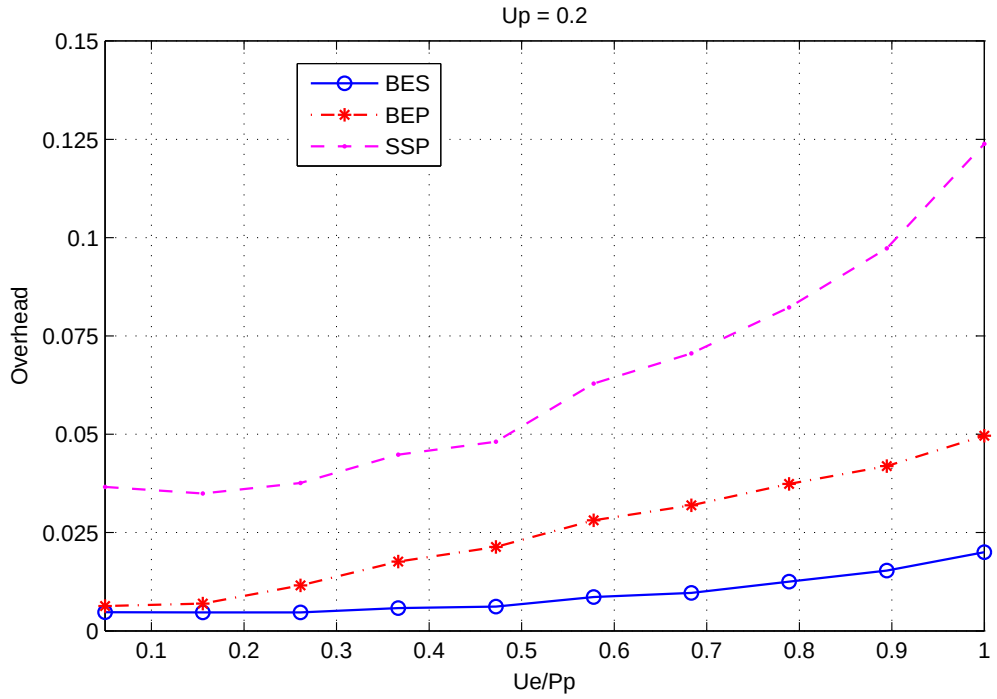


Figure 6.16: Time Overhead with respect to U_e/P_p, for low processing utilization.

6.2.8.2 Varying U_e/P_p

With varying the total energy utilization U_e/P_p, Figures 6.16 and 6.17 show the evaluation of time overhead normalized for the three servers for two total processing settings $U_p = 0.2$ and $U_p = 0.8$. In particular, the overhead is normalized with respect to the total number of periodic and aperiodic jobs. It is evident for the reader to conclude that the overhead increases with the increase of U_e/P_p from 0.5 to 1. When the energy utilization varies, the energy consumed by tasks increases. In that case, the need of inserting idle times and of computing the maximum energy to consume by future jobs increases too. Thus, the computations of slack time and slack energy variables become more frequent.

The overhead is notorious in Figure 6.17 compared to that in Figure 6.16; 67% higher. That signifies that tasks with high processing load are more prone to significant overhead.

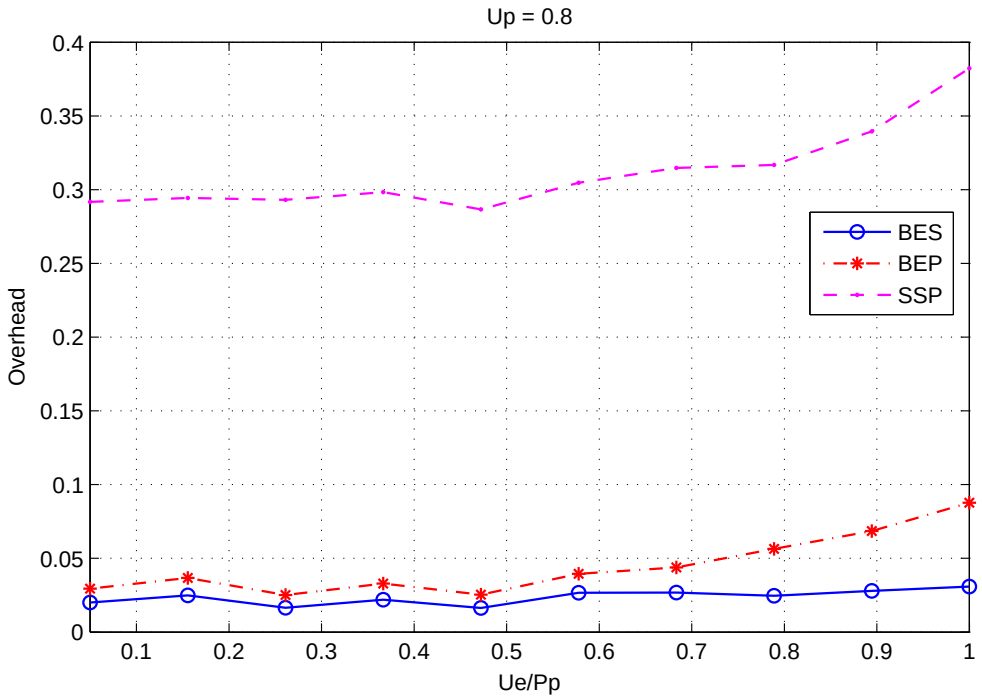


Figure 6.17: Time Overhead with respect to U_e/P_p, for high processing utilization.

6.3 Second set of experiments: variable energy profile

In previous experiments, the three servicing algorithms have been compared under constant energy profile. In this set of experiments, we have evaluated their performance with respect to various energy aspects.

6.3.1 Experiment 1: Average response time of aperiodic tasks

As previously, we have varied the incoming environmental energy in order to study the impact of the four different profiles on the normalized response time of each of SSP, BES, and BEP servers. The power for the first profile was supposed constant and equal to 5. The output power of the three profiles (Figure 6.1) is supposed variable between 2 and 17. The minimum size of the reservoir (E_{min}) for the different profiles should not be less than the area of each profile.

The evaluation are performed for a total energy utilization applied to the system, which varies from 5% to 100%, while the total processing utilization load remains constant ($U_p = 0.6$). The four simulation experiments depicted in Figures 6.18, 6.19, 6.20 and 6.21 refer to the results obtained for constant profile, Sine wave with period $\pi = 2$, Rectifier signal and Pulse signal of 20% duty-cycle respectively. They illustrate the mean aperiodic response time which is normalized with respect to the aperiodic computation time.

The graphs plainly show that SSP outperforms the other algorithms under all energy profiles. This confirms our theoretical analysis whas been performed without any restrictive assumption on the production of energy along time. BEP performs better than BES with a small deviation and exhibits a significant degradation with respect to the BEP algorithm (Figure 6.21) under the Pulse signal profile. It is expected that the results of the Pulse model show that the performance obtained for the three algorithms and in particular for BES is slightly less than ones obtained under the three other models. For example, the responsiveness by BES under the Pulse signal profile is at least 12.5% less than the other profiles. The reason is that the power is only harvested on 20% duty-cycle of the overall signal and BES permits aperiodic tasks to be only

executed when the energy reservoir is full, which leads to an increase in aperiodic responsiveness.

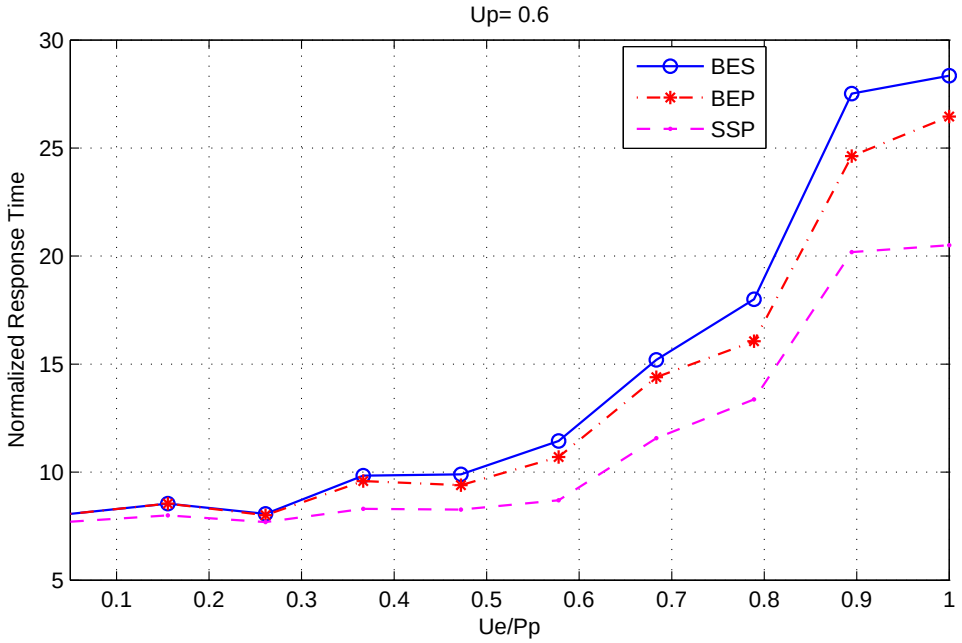


Figure 6.18: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.6 under constant profile.

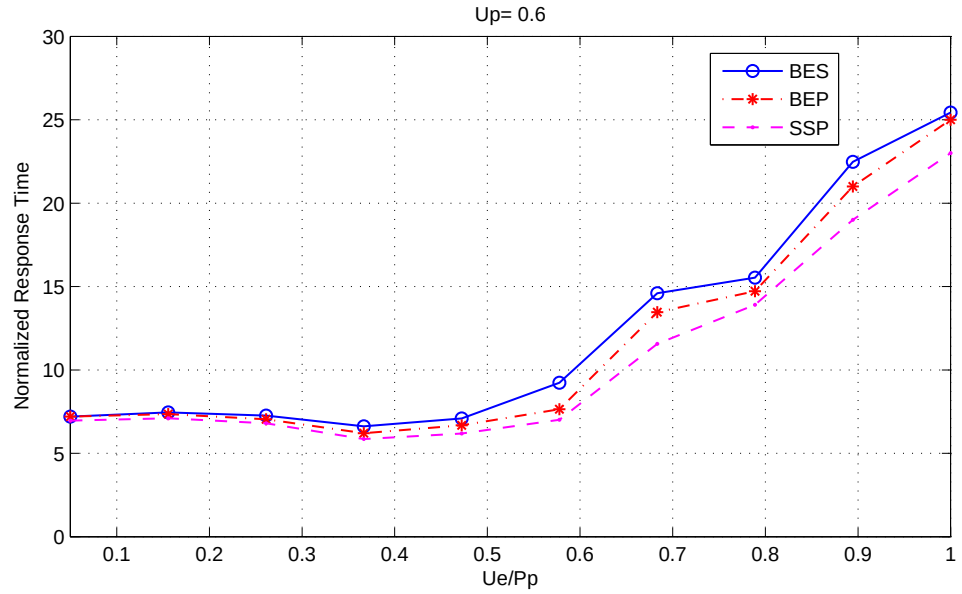


Figure 6.19: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.6 under sinusoidal signal of period $\pi/2$.

6.3.2 Experiment 2: Number of preemptions for various energy profiles

6.3.2.1 Varying U_e/P_p

In this section, we compare the cost incurred by SSP over BEP and BES due to preemptions with varying the total energy utilization U_e/P_p. The Table here below illustrates this metric with four different power profiles in Figure 6.1 assuming that P_p can be accurately predicted. The $3rd$ and $4th$ columns report the

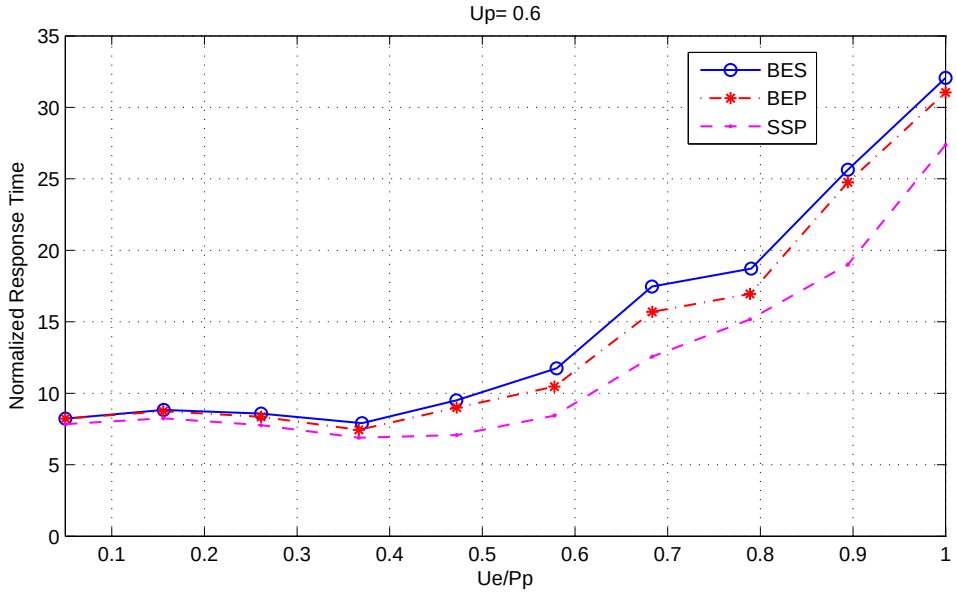


Figure 6.20: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.6 under rectifier signal.

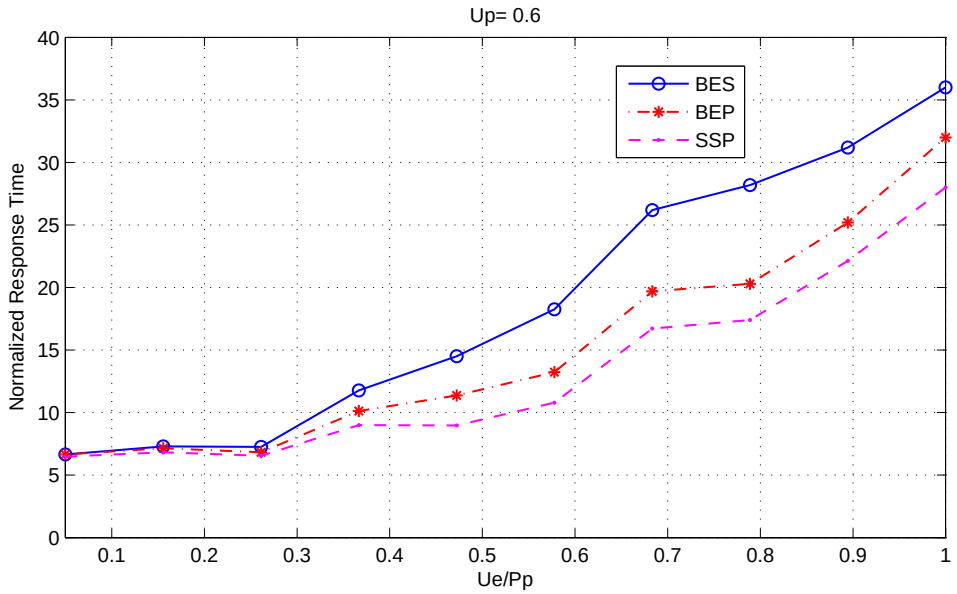


Figure 6.21: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.6 under pulse signal.

overhead results for SSP vs BEP and SSP vs BES respectively for the processing utilization ratio U_p sweeping from 0.2 to 0.8, stepped by 0.2 under constant profile, while for a fairly constrained system ($U_p = 0.4$) under other profiles.

It is shown in Table 6.2 that the percentual overhead of each of SSP vs BES and SSP vs BEP slightly decreases when total utilization is higher because the preemptions number of BES and BEP increases with long execution times at the release of periodic tasks. For example, it decreases by only 10% from low processing utilization to high processing utilization which shows a good impact on the SSP performance. However, we can conclude that the issued overhead ratio doesn't exceed 31.5%. This number is still acceptable and does not affect the SSP performance.

We can also report that the different ratio values show minor differences in all profile when $U_p = 0.4$.

There is no correlation between overhead and the variable profiles Sine wave, Pulse and Rectifier, since they offer results in the same margin than the constant profile. However, the Pulse signal allows a low overhead ratio of SSP over BES. This is because, according to the energy harvested on bounded duty-cycle

of the overall Pulse signal, BES permits aperiodic tasks to be only executed on this bound. This leads to an important number of preemptions in BES.

Table 6.2: Overhead of SSP over BEP and BES with respect to preemption rate with varying total energy utilization.

Profile	U_p	Overhead of SSP (%)	
		SSP vs BEP	SSP vs BES
Constant	0.2	31.5	28.3
	0.4	30.9	27.8
	0.6	29.9	27.0
	0.8	28.3	23.9
Sine wave signal	0.4	29.4	23.5
Pulse signal	0.4	29.2	14.2
Rectifier	0.4	30.2	28.4

6.3.2.2 Varying U_p

Table 6.3 reports a performance comparison between SSP and background policies in terms of number of preemptions with four different power profiles in Figure 6.1 assuming that the incoming power P_p can be accurately predicted in the future. The results are recorded in the $3rd$ and $4th$ columns for four different total energy utilization settings varying from 0.2 to 0.8 under constant profile, and for $U_e/P_p = 0.4$ under other profiles.

From the Table here below, under constant profile, we notice that as the total energy load becomes reletively high, a little increase of the number of preemptions is shown. Such deterioration can be explained as follows: When the normalized energy utilization factor increases, the energy consumed by the periodic and aperiodic tasks increases and therefore the slack energy decreases and is declared with a null value, very often. Despite this behavior, a high energy factor (i.e. 0.8) has an acceptable impact on the percentage overhead since it increases by 5.8% from low to high energy utilization. And the highest resulting overhead does not exceed 29.5%.

On the other hand, we report for each variable ambient energy profile an overhead ratio close to that provided in constant profile for $U_e/P_p = 0.4$. The exception is with SSP vs BES for Pulse signal for the same reason as mentioned in previous section.

In summary, we have used a normalized metric to evaluate the sensitivity of the different servers to preemptions. Our study permits to certify that the SSP server is no more influenced by preemptivity than the background servers under the various energy profiles which were selected in our thesis.

Table 6.3: Overhead of SSP over BEP and BES with respect to preemption rate with varying total processor utilization U_p.

Profile	U_e/P_p	Overhead of SSP (%)	
		SSP vs BEP	SSP vs BES
Constant	0.2	27.8	26.1
	0.4	27.9	26.3
	0.6	29.4	27.4
	0.8	29.5	27.8
Sine wave signal	0.4	28.1	20.4
Pulse signal	0.4	27.9	14.3
Rectifier	0.4	28.5	24.2

6.4 Performance summary

The simulation experiments reported above allow us to give useful insights on the different parameters that can affect aperiodic responsiveness. Here below is a summary of our simulation results and the conclusions we may draw from them:

- BEP and BES show poor performance in terms of response time, jitter time, and latency regardless of time or energy utilization.
- BES is not optimal, especially when it is exposed to high timing and energy constraints. This is because it executes aperiodic task which consumes only processor time and energy which should be wasted if there was no aperiodic task.
- The advantage of the background servers lies in their simplicity.
- BEP invokes the calculation of the slack energy less often than SSP. For that, it shows a less time overhead.
- The SSP algorithm can provide a significant reduction in aperiodic responsiveness, jitter time, and latency, compared to the background services and regardless of the processing/energy constraints variation. It is due to the use of the available idle times and energy surplus in the system.
- the normalized response time of SSP increases by about 20% in fairly processing constrainted system relative to the weakly one, by about 39% in quite highly processing constrainted system relative to the fairly system , and about 60% in highly processing constrainted system relative to the latter one.
- Compared to SSP and BEP, BES is not affected by the process of increasing the reservoir size because it reclaims a complete reservoir replenishment to execute the aperiodic task.
- The Background servers illustrate the lower preemptions rate than SSP because, in general, an aperiodic task never preempts a periodic task. This means that they have low overhead.
- SSP has the higher overhead due to the higher preemptions rate and to the slack time and slack energy computations.

6.5 Conclusion

In that chapter, we reported an extensive simulation that was carried out to measure the relative performance of the optimal (theoretical performance) server SSP, compared with the two background strategies BEP and BES. We formally reported their properties by showing the behaviour of each one under various perspectives like aperiodic responsiveness, jitter, latency, preemptions rate, etc.

The servers work in a dynamic real-time energy constrained environment, where the periodic tasks are scheduled according to the optimal ED-H algorithm and respecting their timing constraints is a commitment. Globally, the simulations confirm that the SSP policy allows a much better responsiveness for the aperiodic activities, taking profit from energy surplus as well as processing time surplus to execute the aperiodic tasks at the earliest time instant. Other properties typically claimed for SSP, such as better jitter control, better latency, in all time and energy conditions, and under various harvested energy profiles.

The real advantage of BEP and BES over SSP is their simple implementation because the latter reveals a high computational complexity and high run time overheads due to the frequent dynamic computation of the slack energy and the slack time. Consequently, the computation requirements of the slack stealing server may be too high in some cases and not compatible with implementation specifications. Let us recall that such scheduler will be dedicated to small devices with very severe limitations in terms of processing power and memory space. So the idea could be to look for a compromise between performance and implementation efficiency. The last chapter of the thesis aims to propose a new server that could be such compromise.

CHAPTER 7

BANDWIDTH-PRESERVING BASED SERVERS: TB-H AND TB*-H

Summary

We propose, in this chapter, a new method that deals with the mixed scheduling of periodic tasks and soft aperiodic tasks with energy harvesting considerations. In the first part of the chapter, we successively describe the so-called TB-H and TB$*$-H task servers. We prove that TB$*$-H is optimal in terms of aperiodic responsiveness. Secondly, our additional contribution lies in extensive simulations that were carried out to show the effectiveness of this Bandwidth based server with respect to background and the slack stealing server, SSP. In this chapter, we only focus on constant renewable energy profiles.

Contents

7.1 The TB-H Server

7.1.1 Background materials with no energy constraints

We have presented in Chapter 1 the foundation of the so-called TB server. We gave the formula proposed by Spuri and Butazzo to compute the fictive deadline of any occurring aperiodic task so that the fraction of processing time demanded by any aperiodic job never exceeds the server utilization U_{ps}. The proposed formula is as follows [13]:

$$d_k = max(r_k, d_{k-1}) + \frac{c_k}{U_{ps}} \tag{7.1}$$

And the condition for schedulability of the periodic task set with the TB aperiodic task server (TBS) is given as follows:

Lemma 2 *In each interval of time $[t_1, t_2]$, if C_{ape} is the total execution time demanded by aperiodic tasks arrived at t_1 or later and served with deadlines less than or equal to t_2, then*

$$C_{ape} \leq (t_2 - t_1)U_{ps} \tag{7.2}$$

Proof: $C_{ape} = \sum_{k=k_1}^{k_2} c_k$
$= U_{ps} \sum_{k=k_1}^{k_2} [d_k - max(r_k, d_{k-1})]$
$\leq U_{ps}[d_{k_2} - max(r_{k_1}, d_{k_1})]$
$\leq U_{ps}(t_2 - t_1)$

Theorem 11 *Given a set of n periodic tasks with processor utilization U_{pp} and a set of aperiodic tasks served by TBS with processor utilization U_{ps}, the whole task set is schedulable if and only if*

$$U_{pp} + U_{ps} \leq 1. \tag{7.3}$$

Proof: "If". Suppose there is an overflow at time t. The overflow is preceded by a period of continuous utilization of the processor . Furthermore, from a certain point $t^{'}$ on, only jobs (periodic or aperiodic) ready at $t^{'}$ or later and having deadlines less than or equal to t are running. Let C be the total execution time demanded by these jobs. Since there is an overflow at time t, we must have $t - t^{'} < C$.
We also know that
$C \leq \sum_{i=1}^{n} \lfloor \frac{t - t^{'}}{T_i} \rfloor C_i + C_{ape}$
$\leq \sum_{i=1}^{n} \lfloor \frac{t - t^{'}}{T_i} \rfloor C_i + (t - t^{'})U_{ps}$
$\leq (t - t^{'})(U_{pp} + U_{ps})$.
It follows that $U_{pp} + U_{ps} \leq 1$, a contradiction.□

"Only If". If an aperiodic request enters the system periodically, say each $T_s > 0$ units of time, and has execution time $C_s = T_s U_{ps}$, the server behaves exactly as a periodic task with period T_s and execution time C_s. Being the processor utilization $U = U_{pp} + U_{ps}$, again from Theorem 7 of [22] we can conclude that $U_{pp} + U_{ps} \leq 1$. □

7.1.2 Total Bandwidth for energy harvesting settings

Let us now demonstrate how to assign the deadlines to the aperiodic tasks when the energy availability is limited in each interval of time. We have to consider that the fraction of energy consumed by the aperiodic tasks should not exceed the aperiodic energy utilization of the server, say U_{es}. This is proved by the following lemma.

Lemma 3 *Given a periodic task set n with an average energy consumption of τ per time unit U_{ep} and a stream of aperiodic tasks processed in FCFS order by a Bandwidth based server with an aperiodic energy utilization U_{es}. In each interval of time $[t_1, t_2]$, consider that E_{ape} is the total energy demanded for aperiodic tasks arrived at t_1 or later and serviced with deadlines less than or equal to t_2, then*

$$E_{ape} \leq (E(t_1) + (t_2 - t_1)P_p)\frac{U_{es}}{P_p} \tag{7.4}$$

Proof: Suppose that, within the interval $[t1, t2]$, there exist m aperiodic tasks having a release time greater than or equal to t_1 and lower than t_2. Also, the total energy produced by the source within this interval is equal to $(E(t_1) + (t_2 - t_1)P_p)$, where P_p is the constant source power. As $\frac{U_{es}}{P_p}$ is the proportion of energy available for aperiodic tasks, it follows intuitively that the total energy demanded by aperiodic tasks after t_1 and before t_2 is less than or equal to $(E(t_1) + (t_2 - t_1)P_p)\frac{U_{es}}{P_p}$. □

Now we have to prove that the aperiodic energy utilization does not exceed $\frac{U_{es}}{P_p}$.

Theorem 12 *Given a set of n periodic tasks with periodic energy utilization U_{ep} and a stream of m aperiodic tasks served by a TB-H server with aperiodic energy utilization U_{es}, the whole set is schedulable only if:*

$$U_{ep} + U_{es} \leq P_p. \tag{7.5}$$

Proof: We prove the theorem by contradiction. Suppose there is a deadline violation due to an energy starvation at time t. The deadline violation comes from the execution of periodic and/or aperiodic tasks. Let t_0 be the time instant before t where only periodic or aperiodic jobs ready at t_0 or later and having deadlines less than or equal to t are executed. Let E be the total energy demanded by these jobs within $[t_0, t]$. Since there is an energy starvation at time t caused by the fact that the energy demand exceeds the energy produced between t_0 and t, we must have $E > E(t_0) + (t - t_0)P_p$.
We also know that

$$E \leq (\tfrac{1}{P_p} * \sum_{i=1}^{n} \lfloor \frac{t - t_0}{T_i} \rfloor E_i) + E_{ape}$$
$$\leq (\frac{1}{P_p} * \sum_{i=1}^{n} \lfloor \frac{t - t_0}{T_i} \rfloor E_i) + (E(t_0) + (t - t_0)P_p) * \frac{U_{es}}{P_p}$$
$$\leq (E(t_0) + (t - t_0)P_p)(U_{ep} + U_{es}) * \frac{1}{P_p}$$

It follows that $U_{ep} + U_{es} \leq P_p$, a contradiction.□

Since the energy required by the aperiodic tasks never exceeds the energy available in a certain interval of time, we will prove that the new deadline assigned can be computed as described by theorem 13.

Lemma 2 states that there exists some interval $[t_1, t_2]$ where the aperiodic energy demand E_{ape} is lower than the energy equal to $(E(t_1) + (t_2 - t_1)P_p)U_{es}$ that could be available in $[t_1, t_2]$. We may draw Theorem 13, the new deadline assignment method for an aperiodic task under energy harvesting constraints.

Theorem 13 *Let U_{es} be the aperiodic energy utilization such that $U_{ep} + U_{es} \leq P_p$. The fictive deadline of a k-th aperiodic task arriving at time $t = r_k$ must be computed as follows:*

$$\tilde{d_k} = max(r_k, \tilde{d_{k-1}}) + \lceil \frac{E_k}{U_{es}} - E(r_k) \rceil \tag{7.6}$$

Proof: When aperiodic task Ap_k arrives at time $t = r_k$, the assignment of the deadline is done. Suppose that Ap_k has an energy demand E_k. Therefore, the fictive deadline must be computed in such way that the occurring task may benefit from this amount of energy. We know that E_k cannot exceed the energy available in the interval $[r_k, \tilde{d_k}]$. It follows that $E_k \leq (E(r_k) + (\tilde{d_k} - r_k)U_{es})$, hence the following inequality $\tilde{d_k} \geq \frac{E_k - E(r_k)}{U_{es}} + r_k$.
We recall that an aperiodic task cannot be preempted by another aperiodic one and thus $\tilde{d_{k-1}} < \tilde{d_k}$. Finally, in the worst-case, Ap_{k-1} may finish its execution at $\tilde{d_{k-1}}$. Then the next aperiodic job Ap_k starts to run at $t = max(r_k, \tilde{d_{k-1}})$ and we obtain: $\tilde{d_k} \geq \frac{E_k - E(r_k)}{U_{es}} + max(r_k, \tilde{d_{k-1}})$. □

Hereafter, we present the deadline assignment with both real time and energy harvesting constraints with TB-H (Total Bandwidth for energy Harvesting systems).

Theorem 14 *Given a set of n periodic tasks and a stream of m aperiodic tasks served by the TB-H server. A suitable deadline for an aperiodic task Ap_k is computed as follows:*

$$d_k^f = max(d_k, \tilde{d_k}) \tag{7.7}$$

Proof: From formula 7.1 and formula 7.6 in Theorem 13, it is the case that formula 7.7 is satisfied. □

The pseudocode of the TB-H aperiodic task server is illustrated as follows:

Algorithm 5 The Total Bandwidth server TB-H with ED-H

Require:
t: current time
Ap_k: Aperiodic task that occurs at t
while True **do**
  if $Ap(t)$ is not empty **then**
    $d_k = max(t, d_{k-1}) + \lceil c_k/U_s \rceil$
    $\tilde{d_k} = max(t, d_{k-1}) + \lceil e_k/U_{es} \rceil$
    $d_k^f \leftarrow max(d_k, \tilde{d_k})$
    assign d_k^f to Ap_k
    insert Ap_k in the ready queue
  end if
  execute the ED-H scheduler
end while

Example 13 *The following example illustrates the TB-H deadline assignment procedure when any aperiodic task occurs. We consider the same sets of periodic and aperiodic tasks that was presented in the previous chapters. A task set Γ of three periodic tasks is considered and represented in Table 7.1.*

Table 7.1: Parameters of periodic tasks with energy considerations

Task	C_i	D_i	T_i	E_i
τ_1	4	9	9	18
τ_2	3	12	12	18

We assume that the energy storage capacity is $C = 10$ energy units at $t = 0$. The rechargeable power is constant along the hyperperiod and equal to 4 ($P_p = 4$).

We suppose that the first aperiodic job Ap_1 has computation time 1, energy consumption 5 energy units, and is released at $t = 9$. Another aperiodic task with computation time 3 and energy consumption 15 energy units is released at $t = 18$.

The periodic processor utilization is $U_{pp} = 0.7$ and the periodic energy utilization is $U_{ep} = 3.5$. This leads to get a bandwidth of processing time $U_{ps} = 0.3$ and bandwidth of energy $U_{es} = 0.5$ dedicated to the aperiodic tasks.

At time 0, the residual capacity is the greatest one since the storage unit is full. τ_1, the highest priority periodic task, runs and finishes at time 4 and consumes 18 energy units. At time 4, the residual capacity is given by $E_{max} - E_1 + P_p * C_1 = 8$. Now, τ_2 has the highest priority. It executes completely until time 7 and consumes 18 energy units. The residual capacity equals 2 energy units. At time 7, the storage unit is recharging since the processor is let idle. At time 9, Ap_1 arrives. It receives a fictive deadline $d_1 = 13$ by equation 7.1 (due to processing time bandwidth of the server) and a second deadline $\tilde{d}_1 = 19$ by equation 7.6 (due to energy bandwidth of the server). Finally, the deadline $d_1^f = max(d_1, \tilde{d}_1) = 17$ is assigned to Ap_1.
That means that Ap_1 will not be serviced immediately, as τ_1 has an earliest deadline. Ap_1 will be jointly scheduled with the periodic tasks under the ED-H scheduler using the deadline newly assigned at $t = 13$ and consumes a maximum of 5 energy units.At time 14, the highest priority task τ_2 executes completely according to ED-H where the residual capacity equals 7. Periodic tasks run till time 18 where a new aperiodic task Ap_2 is released. Ap_2 is assigned a deadline $d_2^f = 49$. But it is not executed immediately, since at time $t = 18$, there is an active periodic task with a shorter deadline equal to 27. Tasks of the system execute timely according to ED-H till the end of the hyperperiod where the energy reservoir has 8 energy units.

Let us notice that the response times of aperiodic tasks Ap_1 and Ap_2 are respectively 5 units of time and 16 units of time.

7.1.3 Implementation considerations

As the classical TB server, TB-H is very simple to implement. For assigning a suitable deadline to a new arriving aperiodic task, we have to keep track of the deadline assigned to the last occurring aperiodic task. After computation of its deadline, the aperiodic task is inserted in the list of ready tasks which gathers all the tasks, periodic ones and aperiodic ones. Consequently, the scheduler has to manage only one list of ready tasks.
Simulations hereafter will show that TB-H exhibits better performance than background servers even if they have both the same low implementation costs.

7.2 The TB*-H Server

In this section, we propose a new deadline assignment method that improves the performance of the previous TB-H server. Our idea is to try to find a deadline which is shorter than that given by TB-H while still guaranteeing the feasibility of the system.

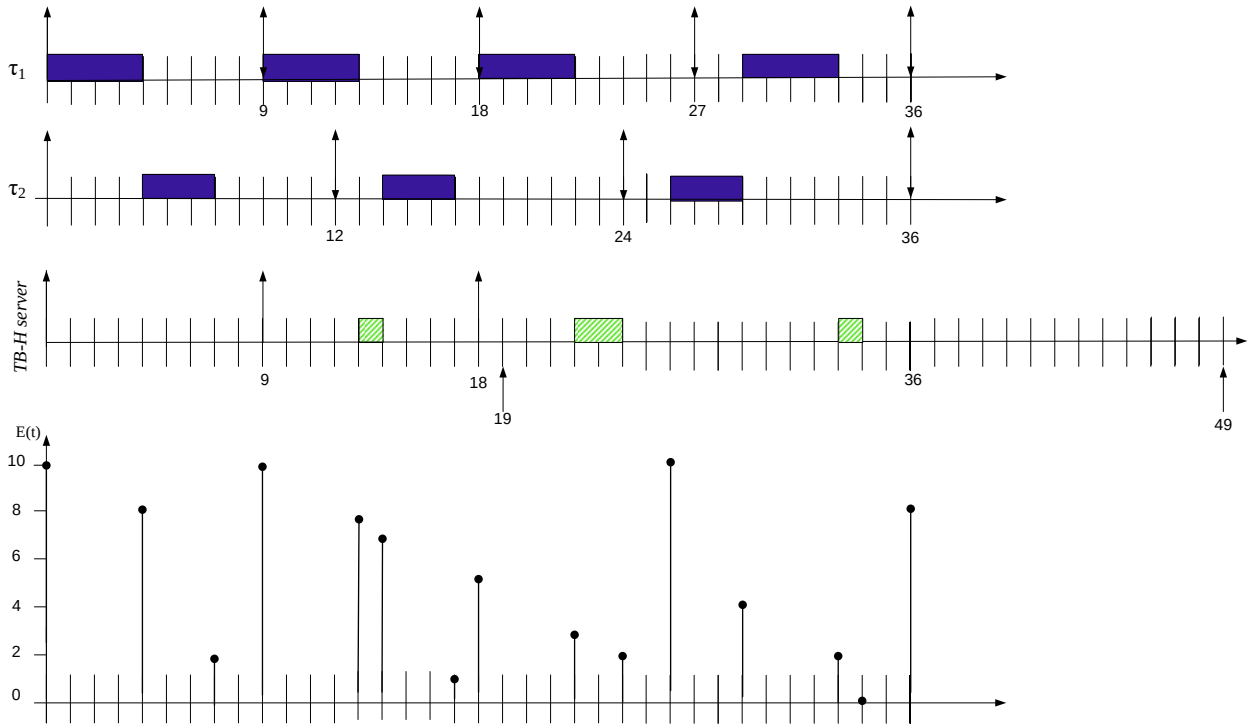


Figure 7.1: Illustration of the TB-H server.

The deadline assignment of the new TB*-H works as follows: the formula given by TB-H is used to determine the deadline of any aperiodic task which arrives while there is no more aperiodic task present in the system. Then, a recurring formula, is used to shorten as much as possible this deadline in order to enhance aperiodic responsiveness. The new deadline is computed so as to be the worst case finishing time of the aperiodic task which takes into account the interferences with the periodic tasks. This procedure was described in Section 1.4.2.3 and is used here in the same way in the case where the periodic tasks are scheduled by ED-H.
Let us recall that the formula is:

$$d_k^{s+1} = t + c_k + I_p(t + d_k^s) \tag{7.8}$$

Here t denotes the current time, c_k is the worst case execution time of the aperiodic task. $I_p(t + d_k^s)$ represents the interference on the aperiodic task due to the jobs of the periodic tasks between t and the deadline. Here the tasks are scheduled by ED-H instead of EDF. But these two schedulers are the same in terms of decision rule for selecting the future active task. Both selects the task with the closest deadline. The only difference these two schedulers is the placement of idle time intervals. Although we did not prove its optimality, we may intuitively expect that TB*-H is an optimal server as the SSP server.

Consequently, the final deadline assigned by TB*-H to an aperiodic task Ap_k is computed as follows:

$$d_k^f = max(d_k^{s+1}, \tilde{d}_k) \tag{7.9}$$

The pseudo-code of TB*-H is similar to that of TB-H with a difference in the computation of deadline. Today, the optimality of this new server has not been stated yet.
In summary, the TB*-H server can be considered as the optimal version of the TB-H server which contains only one iteration for computing the deadline of any aperiodic task. We may imagine different versions of TB*-H according to the number of iterations which are performed to determine the exact optimal deadline.

But higher is the number of iterations, higher will be the performance of the server. But higher will be the overhead due to the computational complexity for determining the deadline.

Example 14 *We use the same example as one which illustrated the TB-H server. The periodic processor utilization is* $U_{pp} = 0.7$ *and the periodic energy utilization is* $U_{ep} = 3.5$*. This leads to get a bandwidth of processing time* $U_{ps} = 0.3$ *and bandwidth of energy* $U_{es} = 0.5$ *dedicated to the aperiodic tasks. By applying equation 7.8, we obtain* $d_1 = 13$ *for* Ap_1 *and* $d_2^* = d_2^1 = 25$ *for* Ap_2*. Then, by applying equation 7.6, we got* $\tilde{d}_1 = 13$ *and* $\tilde{d}_2 = 25$ *for* Ap_1 *and* Ap_2*, respectively. Finally, 7.9 gives us the final deadlines* $d_1^f = max(d_1, \tilde{d}_1) = 19$ *and* $d_2^f = max(d_2^*, \tilde{d}_2) = 49$*. Hence, the example results lead to the same schedule illustrated under the TB-H server in Figure 7.1.*

7.3 Performance Evaluation

TB-H, TB$*$-H, SSP, BEP, and BES are simulated to compare their performance in terms of aperiodic responsiveness, jitter, latency, preemptions task rate, overhead, and impact of power and/or energy reservoir. In all experiments, we adopted the identical simulation environment presented in Chapter 6 to generate and schedule the periodic tasks set and the soft aperiodic tasks stream.
The total processing load U_p incorporates 50% of the periodic processor utilization U_{pp} and 50% of the aperiodic utilization U_{ps}. Identically, the total energy load U_e includes 50% of the periodic energy utilization U_{ep} and 50% of the aperiodic energy utilization U_{es}.

In this work, we consider that the storage capacity is initially full and the recharging power P_p is constant.

7.3.1 Experiment 1: Average response time of aperiodic tasks

In this first set of experiments, SSP, $TB^* - H$, TB-H, BEP, and BES algorithms have been simulated to compare the average response times of soft aperiodic tasks with respect to the total energy load.

Simulation results reported in Figures 7.2 and 7.3 are carried out for a processing load equal to 0.2, and 0.8 respectively, varying the energy load ($5\% \leq U_e \leq 100$).

From the graphs, we can say that the SSP server and the TB servers offer better performance compared to the two BG servers. Moreover, this advantage is more significant as the energy load U_e is higher. The major difference in the performance between the optimal servers (SSP and $TB^* - H$) and the naive servers (BEP and BES) appears for heavy energy loads. Note that, TB-H and $TB^* - H$ have about the same responsiveness when the energy load is low, and they show a slightly different behavior for high energy conditions.

In all graphs, the maximum difference between the performance of SSP and the optimal $TB^* - H$ is no more than 7%, even for high energy load.

For the first experiment (Figure 7.2, $U_p = 0.2$), the SSP and the TB servers provide a significant decrease of at least 16% of response times compared to the background servers for $U_e \geq 70\%$.

In the third experiment (Figure 7.3), the SSP and the TB servers always provide the best response time performance still with a highly processing constrained system (U_p = 0.8). For example, if we consider the performance of SSP and $TB^* - H$ when the total energy load equal to 90%, the aperiodic response time offered is reduced by at least 28% in comparison with the background strategies. The optimality of $TB^* - H$ server has to be paid with the increasing number of shortening steps where SSP takes profit of the energy slack stealing to outperform the other servers.

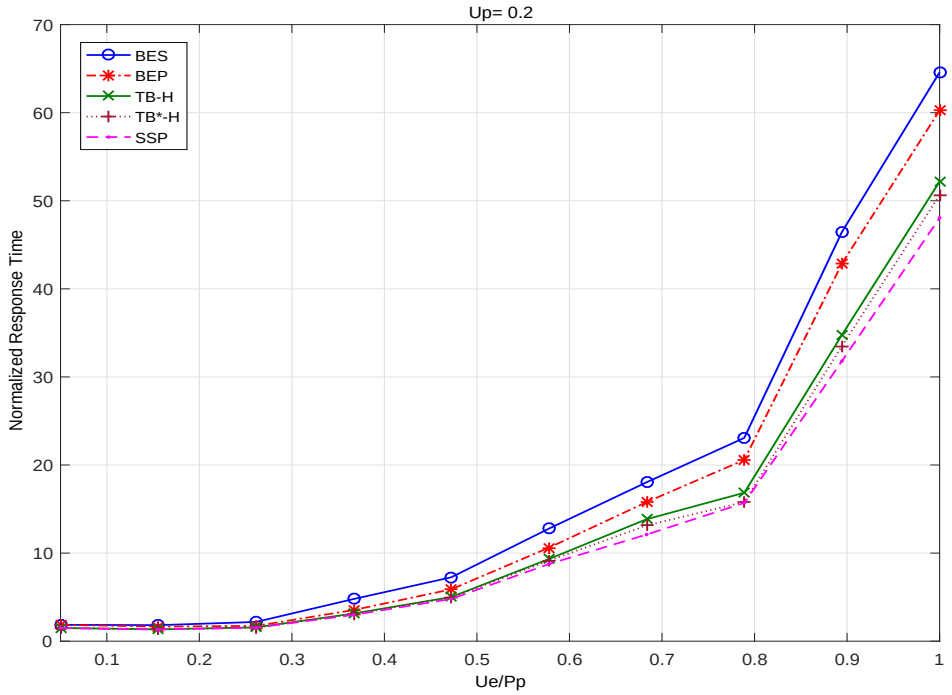


Figure 7.2: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.2.

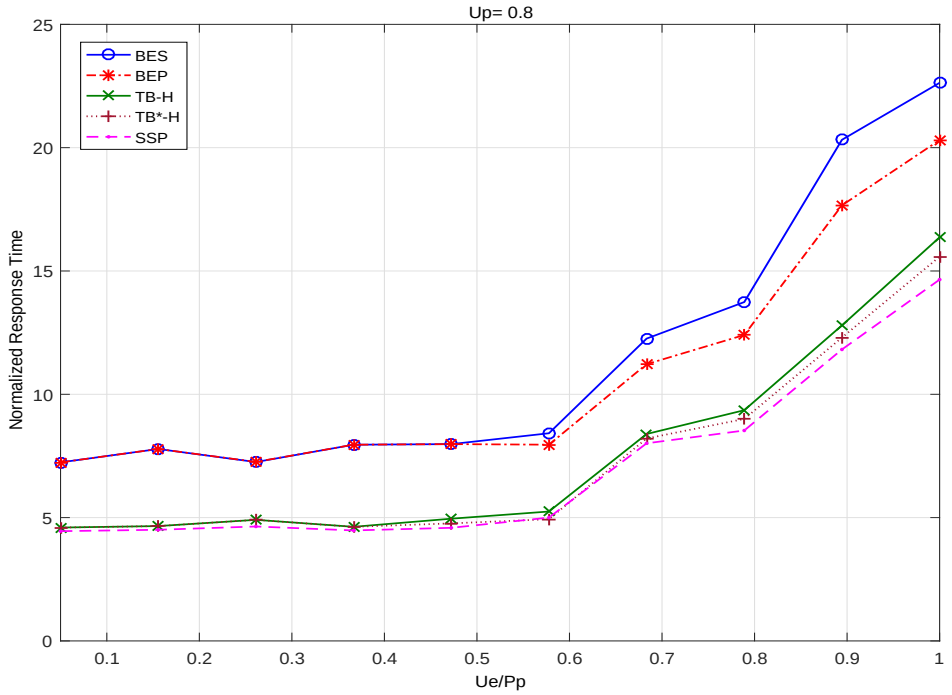


Figure 7.3: Normalized aperiodic response time with respect to U_e/P_p, for U_p=0.8.

7.3.2 Experiment 2: Average jitter of aperiodic tasks

The behaviour of SSP, BES, BEP, and TB servers is studied as a function of the total energy utilization for two processing utilization settings ($U_p = 0.2$ and $U_p = 0.8$). The normalized jitter time measured is illustrated in Figures 7.4 and 7.5, respectively. Globally, the Total Bandwidth and Slack Stealing servers can provide a significant reduction in jitter time compared to background servers; their maximum reached

normalized jitter time is less than the half of their corresponding response time, which means that they do not take a lot of time to service the aperiodic tasks.

The results ahow that the TB-H and $TB^* - H$ approaches record the same jitter time for all the energy utilization steps and under all workload settings.

In Figure 7.4, all algorithms linearly increase with the increase of tasks energy consumption.

For higher load (Figure 7.5), the jitter time is increasing with the increase of the energy utilization. For heavy energy consumption tasks (i.e. U_e/P_p>0.8), the jitter is reduced at the expense of tasks with lower energy consumption, because a small jitter time increase of tasks is detected.

Then, as expected, the SSP and TB servers still beat BEP and BES algorithms by a large margin. For example, TB-H reduces the jitter time by 16%, compared to BES. That means that the aperiodic task in TB-H doesn't take so much time to be serviced upon its release.

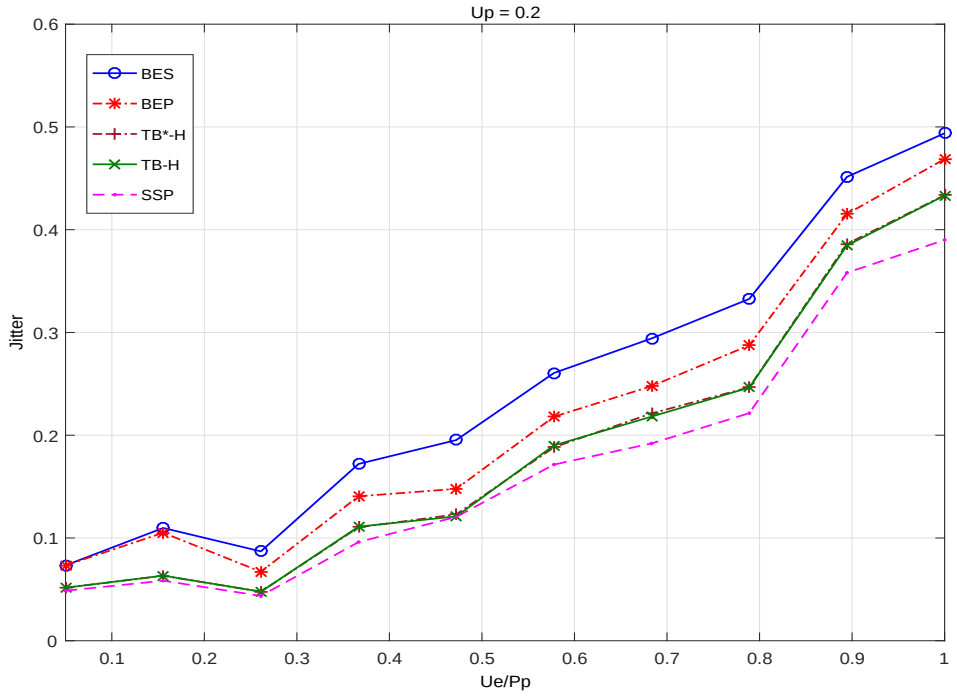


Figure 7.4: Normalized jitter time with respect to U_e/P_p, for U_p=0.2.

7.3.3 Experiment 3: Average latency of aperiodic tasks

To test the latency of the algorithms with respect to the total processing load, two simulations were performed, using $U_p = 0.2$ and $U_p = 0.8$. The results achieved by the TB, SSP, and background servers depending on the total energy utilization are shown in Figures 7.6 and 7.7.

We can observe, that TB-H and TB*-H decrease the maximum latency time compared to the other algorithms. For low processing utilization (Figure 7.6), the two proposed preserving bandwidth algorithms and the SSP do not have significant differences in their performance. Their effectiveness over BEP and BES is achieved for heavy energy utilization. For high processing utilization (Figure 7.7), the TB servers outperform the SSP server by a maximum difference of 21%. For high energy utilization, the aperiodic jobs are executed at the expense of the presence of periodic tasks because a reduced latency indicates that aperiodic jobs are not so much interrupted and because of the minimum deadline assigned by the TB servers.

The result of this set of experiments suggests that it may not be worth using sophisticated algorithms as SSP server, because the latency time achievable in TB servers are not so much affected by high processing and/or energy utilization.

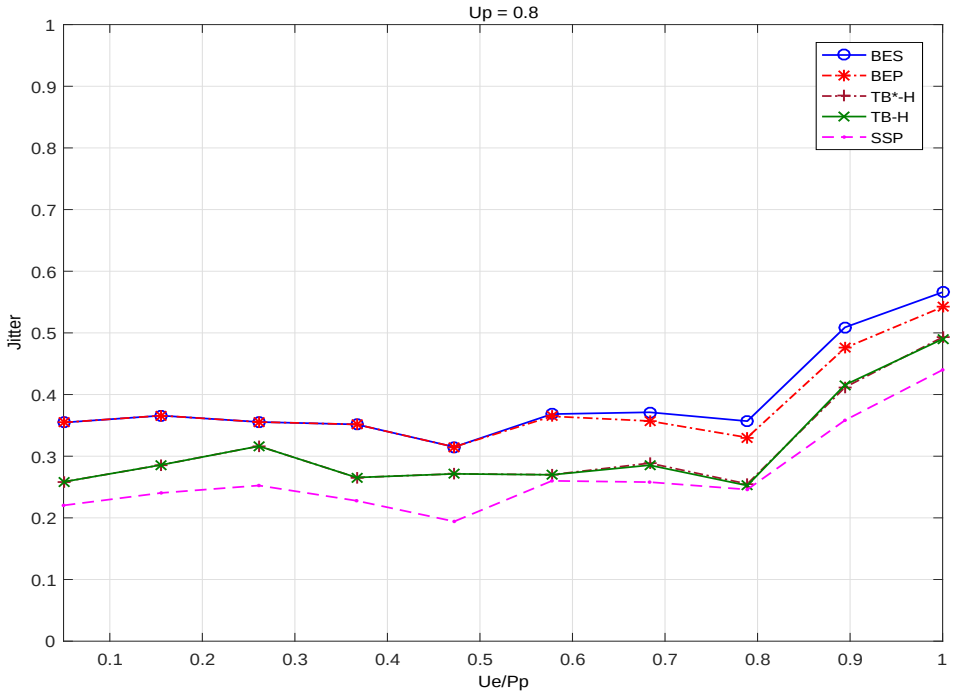


Figure 7.5: Normalized jitter time with respect to U_e/P_p, for U_p=0.8.

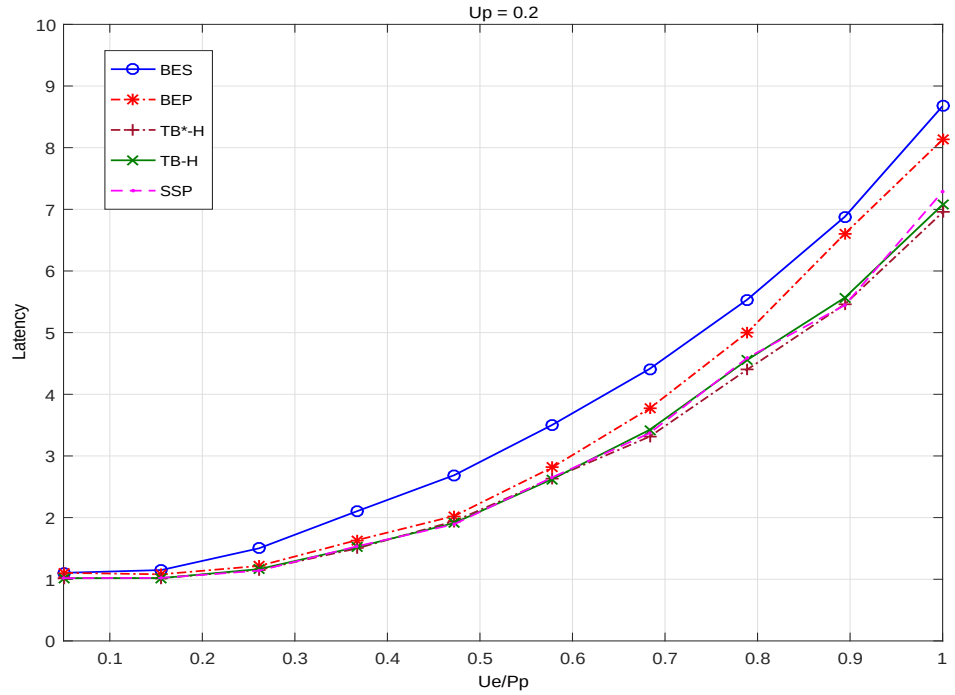


Figure 7.6: Normalized latency time with respect to U_e/P_p, for U_p=0.2.

7.3.4 Experiment 4: Relative performance with different reservoir sizes

In this set of experiments, we evaluate the performance of the servers by varying the reservoir size with E_{min}, 5*E_{min}, and 9*E_{min}. E_{min} is the minimum size of the reservoir that guarantees time and energy feasibility, for given U_p, U_e, and P_p. Here, we report the results for systems which are not time-constrained i.e. $U_p = 0.2$. In Table 6.1, the 3rd, 4th, 5th, 6th and 7th columns give the aperiodic responsiveness of BEP,

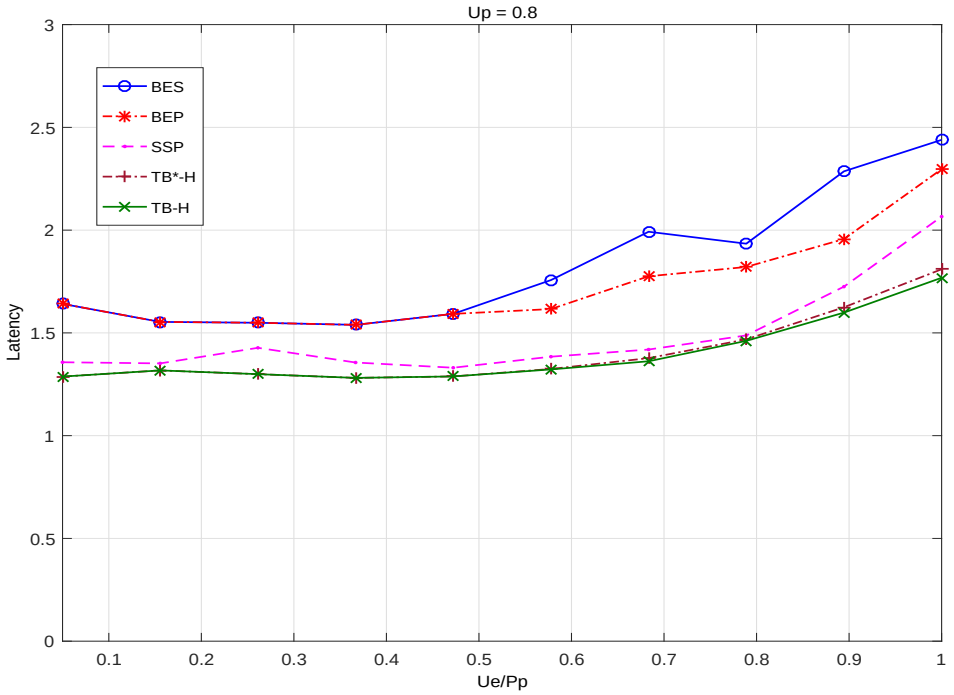


Figure 7.7: Normalized latency time with respect to U_e/P_p, for U_p=0.8.

BES, SSP, TB-H, and $TB^* - H$ servers, respectively, for two profiles in terms of energy constraints.

Table 7.2 shows that the Total bandwidth servers achieve significant reduction in aperiodic responsiveness, comparing with the Background servers under all parameter settings, and a very close performance (sometimes similar under some energy conditions) to SSP.

For example, when the system uses 20% of available energy with minimum reservoir size, the response time under TB-H is 14% and 25% lower compared to BEP and BES respectively. If the energy requirement is set to 80%, all servers record relatively high response times. However, the optimal Total Bandwidth server $TB^* - H$ tends to approach the SSP server and still outperforms the background servers by a large difference due to optimal deadline assignment and the smart utilization of the extra energy.

For each of the five strategies, the higher is the size of the reservoir, the lower is the normalized aperiodic response time for a given energy setting. For example, if the reservoir size is set to E_{min} and the system uses 80% of available energy, the TB-H and the $TB^* - H$ servers have aperiodic response time equal to 29.0 and 28.1 respectively. When increasing the reservoir size to $9 * E_{min}$, their response time is respectively reduced by 75%.

Such a significant improvement in aperiodic responsiveness comes from possible immediate service through extra energy which is available in the reservoir.

Table 7.2: Relative performance of TB-H and TB^*-H with different reservoir sizes

Capacity	U_e/P_p	BES	BEP	SSP	TB-H	TB^*-H
E_{min}	0.2	2.4	2.1	1.7	1.8	1.7
	0.8	37.4	35.2	26.2	29.0	28.1
$5 * E_{min}$	0.2	2.0	1.7	1.4	1.5	1.4
	0.8	23.0	15.8	14.7	14.9	14.7
$9 * E_{min}$	0.2	1.5	1.3	1.1	1.2	1.1
	0.8	13.4	8.7	6.3	7.3	7.0

7.3.5 Experiment 5: Impact of harvested power and reservoir capacity on responsiveness

In this experiment, we tested the impact of the harvested power and the reservoir capacity on the aperiodic responsiveness of TB-H. For presentation issue, the aperiodic response times on the y-axis are normalized as following:

$$RT = \frac{\sum_{i=1}^{m} \frac{f_i - a_i}{c_i}}{m}. \tag{7.10}$$

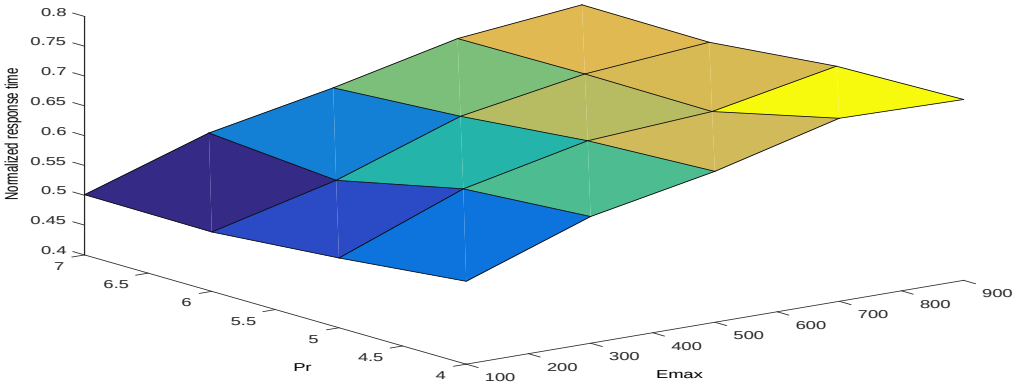
Thus, a value of 1 on the y-axis corresponds to the shortest response time, and a value of zero to the worst response time. Figure 7.8 illustrates the three-dimensional plots of normalized response time of the TB-H server by sweeping both harvest power and reservoir capacity for three different total energy utilization settings: $U_e/P_p = 0.2$, 0.4 and 0.8. It is worth recalling that the values of variable power P_p is extracted from Profile 2 in (Figure 6.1, Chapter 6); $P_p, 2 * P_p, 4 * P_p$, and $8 * P_p$ where P_p is considered the lowest value. Also five different storage capacities sweeping from E_{min} to 9*E_{min} are considered; E_{min}, being the minimum size of the reservoir that guarantees time and energy feasibility depends on the configuration parameters of each experiment.

The simulation results in Figure 7.8 show that, in addition to the total energy utilization, the harvested power P_p and the reservoir capacity E_{min} also affect the performance of TB-H. For weakly and fairly energy constrained systems, TB-H exhibits good aperiodic responsiveness. For highly energy constrained system, its performance degrades by at least 20%. In all experiments, the higher the harvested power and/or the reservoir capacity, the better the response time achieved by TB-H. This appears because of the excess of energy introduced by the power or the reservoir. As shown in the plots, the reservoir capacity has the most impact on responsiveness since the improvement in reducing the normalized response time is significant with the increase of the reservoir capacity, e.g. the response time is reduced by 49% when rising from E_{min} to 9*E_{min} and by only 6% when sweeping from P_p to $8 * P_p$ in Figure 7.8c.

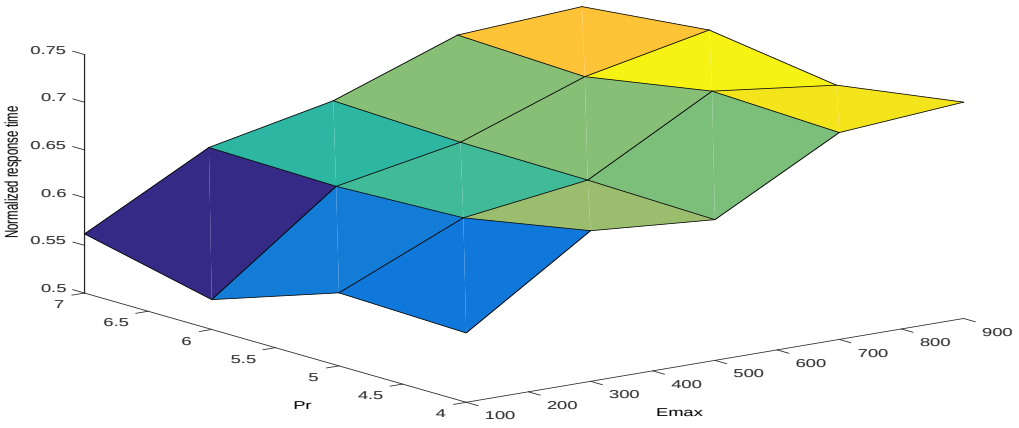
7.3.6 Experiment 6: Tasks preemption rate

As previously mentioned, the task preemption rate serves to evaluate the overhead involved by context switching. The fewer are the preemptions, the fewer are the context changes, and the more efficient is the scheduler. The number of preemptions is compared as a function of the aperiodic load U_{ps}, between the BEP, the BES, the SSP, and the TB servers, for low and high total energy utilization settings in Figure 7.9 and 7.10, respectively.

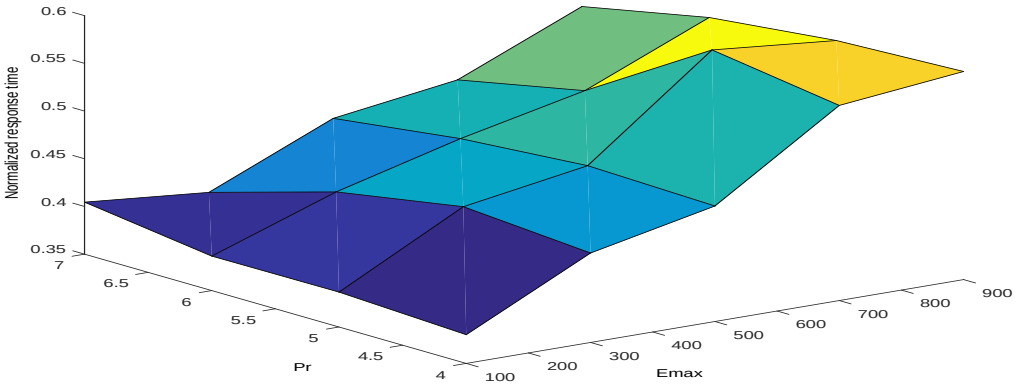
For $U_{ep}/P_p = 0.1$ and $U_{es}/P_p = 0.1$ (Figure 7.9), BES and BEP exhibit similar performance in terms of preemption rate which is constant with the variation of U_{ps}. This phenomenon appears because of the constant periodic processing utilization. The preemption rate of SSP increases and differs from the background servers by at least 76%. As expected, TB-H and $TB^* - H$ outperform SSP by 41.3% when the processor utilization is close to one. For $U_{ep}/P_p = 0.4$ and $U_{es}/P_p = 0.4$ (Figure 7.9), the preemption rate obtained with BES and BEP are also similar. The curve of SSP behaves higher than in previous



(a) weakly energy constrained system



(b) fairly energy constrained system



(c) highly energy constrained system

Figure 7.8: Impact of storage capacity and harvested energy on Responsiveness of TB-H for different energy utilization settings.

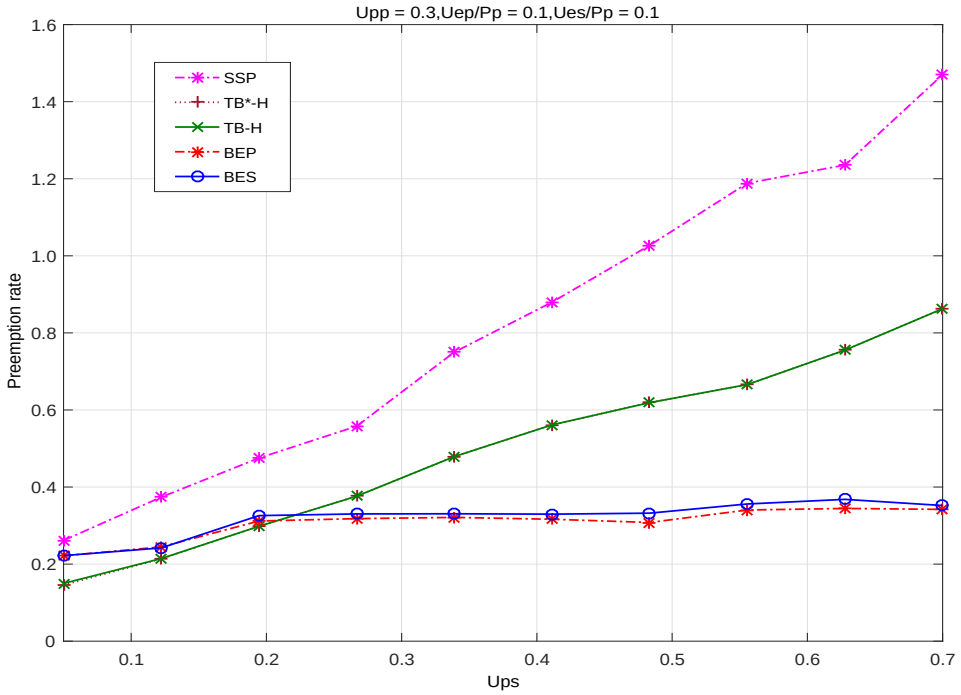


Figure 7.9: Preemption rate with respect to U_{ap}, for low energy utilization.

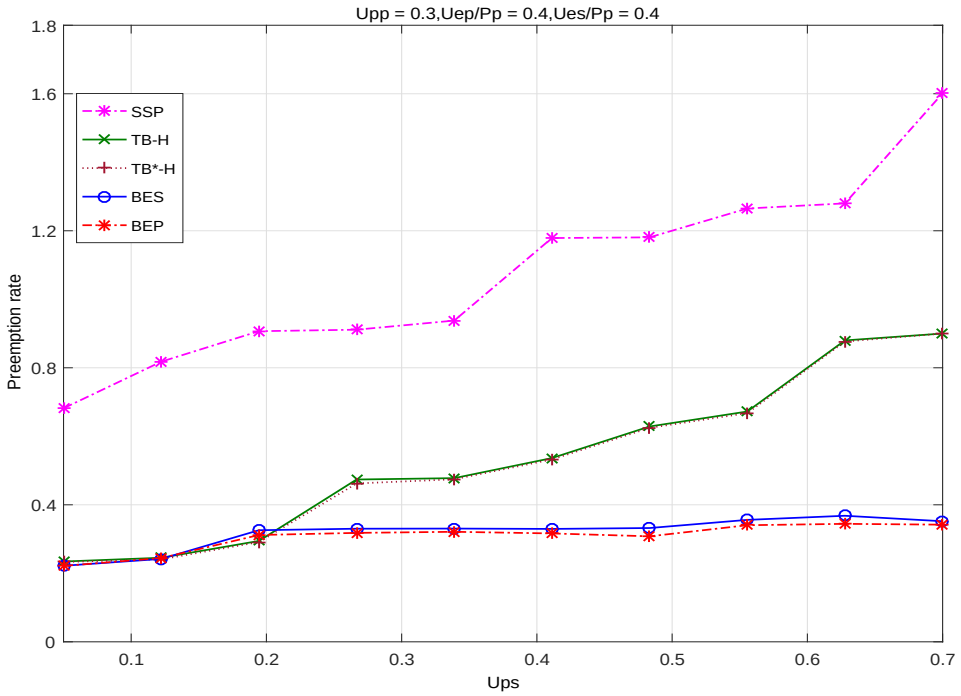


Figure 7.10: Preemption rate with respect to U_{ap}, for high energy utilization.

case. The TB servers outperform also the SSP server by 43.8% when $U_{ps} = 0.7$. The low number of preemptions provided by the TB servers reveals that they incur a low context switching overhead, and that their implementation is more simpler than SSP.

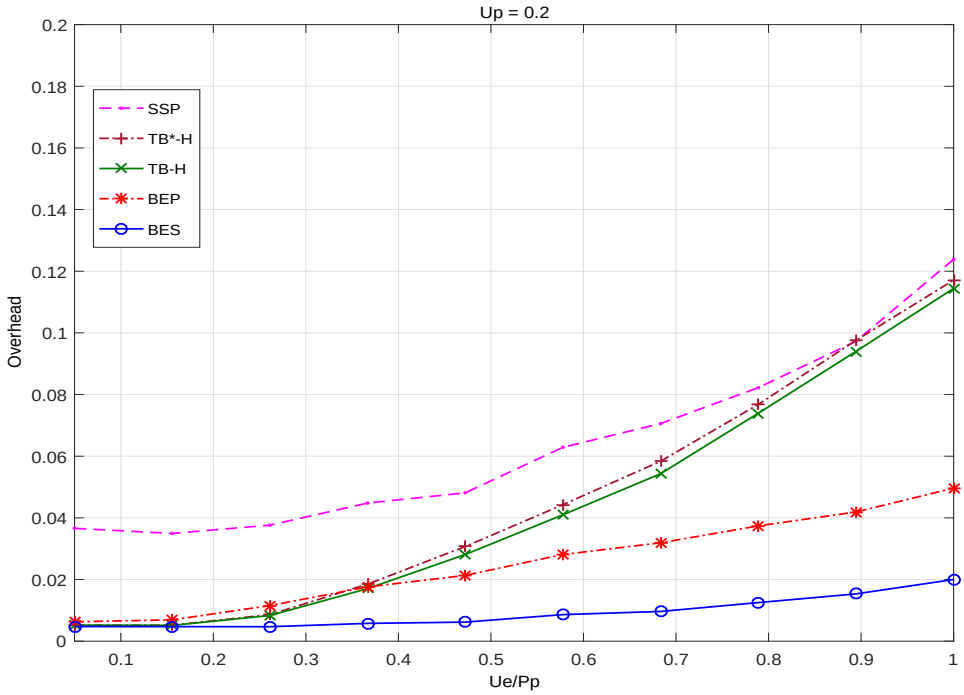


Figure 7.11: Time Overhead with respect to U_e/P_p, for low processing utilization.

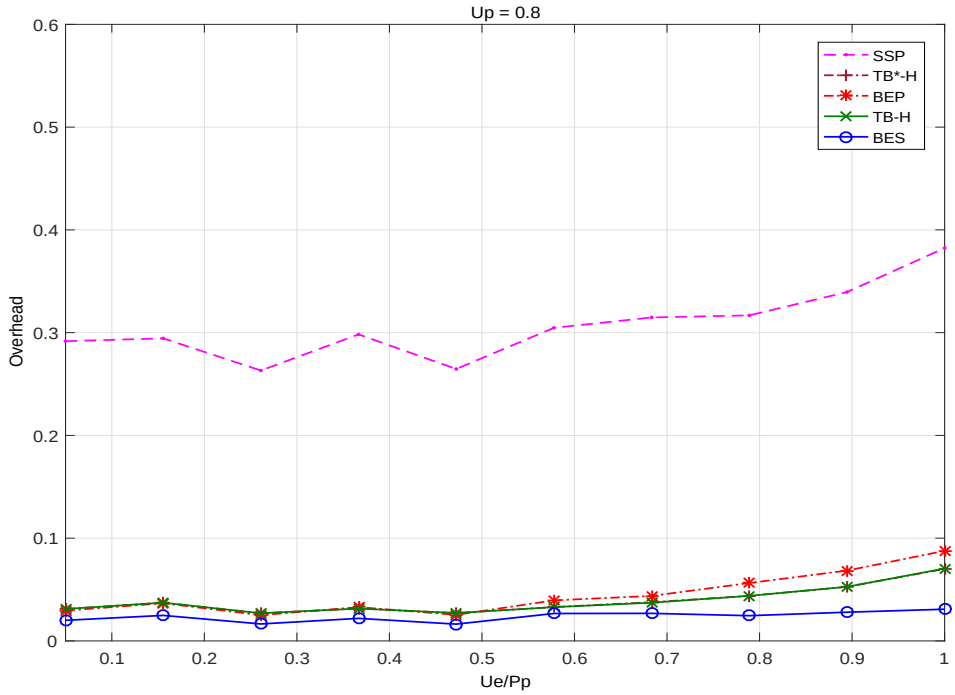


Figure 7.12: Time Overhead with respect to U_e/P_p, for high processing utilization.

7.3.7 Experiment 7: Overhead

In this section, we evaluated the effect of total processing utilization U_e/P_p on the computing overheads. Figures 7.11 and 7.12 show the normalized overhead introduced by the SSP, BEP, BES, TB-H, and TB^*-H algorithms with two different processing utilization settings (light and heavy): 1) U_p = 0.2 and 2) U_p = 0.8. The normalized time overhead is the number of times where we compute either the slack time or the slack

energy relative to the total number of jobs in SSP.

The graphs show that the harvesting-aware (TB-H and $TB^* - H$) servers consistently outperform the slack stealing approach, in terms of overhead, for a light energy utilization (Figure 7.11). For a heavy energy utilization (Figure 7.12), they outperform SSP by 81.69%, and BEP by 19.5%, with exception of the BES server that incurs less overhead than the TB servers. The difference between BES and TB-H is considered negligible in practical applications.

7.4 Synthesis

The aim of the simulations is to evaluate the performance of the Total Bandwidth algorithms (TB-H and TB*-H) proposed in this chapter, in comparison with the Slack Stealer and Background algorithms introduced in the previous chapters. Table 7.3 summarizes the properties of the evaluated algorithms.

In this table, the ✓ represents the good performance of the scheduling algorithms at a given criterion. These results highlight the existence of a performance-complexity trade-off. In fact, this classification shows the poor performance of the background-based algorithms (BES and BEP), which is characterized by simple implementation. Moreover, this table highlights the performance prevalence of the SSP algorithm, which is acquired at the expense of an increase of complexity computation and implementation. The TB-H and TB*-H provide good results by balancing performance against implementation complexity.

Table 7.3: Algorithms comparison summary.

	BG-based		SS-based	BP-based	
	BES	BEP	SSP	TB-H	TB*-H
Optimality	not optimal	not optimal	optimal	not optimal	optimal
Complexity	$O(n)$	pseudo-polynomial	pseudo-polynomial	$O(n)$	$O(n)$
Worst-case scenario	$O(n.m)$	pseudo-polynomial	pseudo-polynomial	$O(n.m)$	$O(Nn.m)$
Average Response Time	✓	✓	✓ ✓✓	✓✓	✓✓
Average Jitter Time	✓	✓	✓ ✓✓	✓✓	✓✓
Average Latency Time	✓	✓	✓✓✓	✓✓	✓✓
Preemptions Rate	✓	✓	✓✓✓	✓✓	✓✓
Time Overhead	✓	✓ ✓	✓✓✓	✓✓	✓✓

7.5 Conclusion

In this chapter, we presented a new family of aperiodic task servers for energy harvesting systems. The server TB-H is an extension of the Total Bandwidth server introduced by Spuri and Buttazzo in 1994. It consists in: 1) computing a first suitable deadline to each ready aperiodic job while still ensuring that the timing constraints are met, 2) computing a second suitable deadline while still guaranteeing that the energy constraints are met, 3) assigning a final fictive deadline which is the maximum of the two deadlines computed in step 1 and step 2 and finally 4) inserting the aperiodic task into the system ready queue together with the periodic tasks. A refinement of TB-H called $TB^* - H$ has been proposed. Its optimality is intuitive as the server TB^* was proved optimal by Buttazzo in 1999 under no energy consideration.

Compared with the slack stealing and background approaches, TB-H was proved to provide identical performance in terms of response time, jitter time, and latency. TB-H is very simple to implement, and no additional overhead is required since aperiodic tasks are processed with the periodic ones in one queue.

However, its main drawback is that the energy model assumes a constant source power. To the contrary, the slack stealing server SSP has no restriction on the energy source model.

GENERAL CONCLUSION

Motivations

The issue we dealt with in this thesis is fundamental because of the following reasons:

- Energy-harvesting technologies eliminate the need for batteries and consequently remove the obstacle to the success of the Internet of Things. Indeed, energy harvesting presents a straightforward way to easily power remote devices in the IoT using clean energy (solar, thermal, mechanical, etc.).
- Most of devices connected to the IoT are things with real-time constraints. These wireless terminals are equipped with sensors. They collect information about the environment that surrounds the sensor terminal, regularly and continuously along time.
- The application software in a wireless sensor node can be modelled by a set of periodic tasks with hard deadlines in conformance to the classical model of Liu and Layland.
- Nevertheless, in the so called real time energy harvesting system, each task is characterized not only by its timing parameters but in addition by its energy requirements. And the system is characterized by two additional elements: the storage unit with a given capacity for transient storage of energy and the energy harvester with a given energy production profile.
- Although most of tasks are periodic, some additional tasks said to be aperiodic may be activated on the occurrence of certain events. Theses ones have consequently irregular and unpredictable arrival times. They require to be processed as soon as possible still guaranteeing the feasible execution of the periodic tasks.

Issue addressed in the thesis

This dissertation addressed the problem of scheduling mixed sets of tasks with time and energy constraints: hard periodic tasks and soft aperiodic tasks. Based on real-time energy harvesting system concepts, this work tackled the following issue: how to optimize the response times of the aperiodic tasks without injuring the schedulability of periodic tasks on a dynamic-priority basis. Considerable research has been done in this area but not for real-time systems with energy harvesting capabilities.

This problem concerns a uniprocessor hardware architecture that is powered by one or several energy harvesters which allow it to achieve perpetual energy autonomy known as energy neutrality.

The problem of scheduling a mixed set of hard periodic tasks and soft aperiodic tasks has been widely considered when there are no energy restrictions. Under that hypothesis, famous aperiodic task servers were proposed. Among them, there are three important families: Background, Slack Stealing and Total Bandwidth. The objective of that thesis was to investigate how to extend these servers to the energy harvesting constrained systems.

All the methods we proposed assume that periodic tasks are scheduled by the ED-H algorithm. ED-H is an optimal algorithm with no restriction on energy production profile. It is dynamic and may achieve maximum processor utilization and maximum energy utilization.

Proposals of the thesis

The key contribution of this work was to provide the real-time system designer with new on-line algorithms for servicing aperiodic tasks in real time energy harvesting systems. Our proposal includes:

– *two Background based servers respectively called Background with Energy Surplus (BES) and Background with Energy Preserving (BEP) (chapter 4).*
BES executes an aperiodic task only if no periodic task is pending for execution and the energy reservoir is fully replenished. BEP executes an aperiodic task if there is no awaiting periodic task and the system slack energy is positive so as to avoid energy starvation. Theoretically and experimentally, BEP significantly enhances the performance of BES with no much additional overhead. Besides the simplicity of the two background-based servers, the performances they provide in terms of aperiodic responsiveness are so poor and crave strongly to be optimized.

– *a new slack stealing based server called SSP (Slack Stealing with energy Preserving) (chapter 5).*
SSP contains a method for determining the maximum processing time (slack time) and the maximum energy (slack energy) which may be stolen from hard deadline periodic tasks, without jeopardizing their timing constraints and without provoking energy starvation.
We have presented the theoretical foundations for the SSP server. Our main contribution is that we proved SSP to be optimal. Optimality is in terms of providing the shortest aperiodic response time among all aperiodic task servers. We compared the performance of the optimal SSP server to the previous Background servers with extensive simulations (chapter 6). The results of these tests confirm the theoretical analysis. The SSP server outperforms the two background servers for all harvested power models and any processing and/or energy utilizations.
In contrast to Background based servers, the SSP server is not so simple to implement. This is because we need to keep track of the two key dynamic data: slack time and slack energy. And their computation is pseudo-polynomial in time. This makes SSP requiring a relatively large overhead to be practical for some real-world applications.

– *two new Bandwidth based servers called TB-H (Total Bandwidth for Energy Harvesting systems) and TB***-H (chapter 7).*
Both servers consists in assigning a suitable deadline to any new aperiodic task that enters the system. Firstly, the TB-H server is proposed as an extension of the TB server that does not consider energy issues. Secondly, we further extend the TB-H server to provide a better method for assigning deadlines with an iterative manner to the soft aperiodic tasks. Basically, we reduce the deadline assigned by TB-H to enhance responsiveness but still guaranteeing the deadlines of periodic tasks.
A valuable feature of this approach is to be flexible: we may improve the performance of the TB-H server up the best one at the cost of additional computational complexity and consequently at the cost of additional run-time overhead.

Extensions of the thesis

All the perspectives that we consider affordable can either bring improvements or innovations to the contributions made by our work. In a short future, we propose to relax some of the assumptions introduced in our real time energy harvesting (RTEH) model.

– Demonstrating the optimality of the $TB^* - H$ server,
– Sharing mutually exclusive resources,
– Adding aperiodic tasks with deadlines,
– Considering a more realistic model for the energy storage unit,
– Considering a processor with DVFS capabilities.

PART III

SUMMARY IN FRENCH

CHAPTER 8

CONTRIBUTIONS À L'ORDONNANCEMENT EN TEMPS RÉEL POUR LES SYSTÈMES AUTONOMES EN ENERGIE

Sommaire

Ce chapitre présente succinctement les différents points abordés dans la thèse. Dans un premier temps, nous y introduisons brièvement l'ordonnancement des systèmes temps réel et les plus importants algorithmes à priorité dynamique existants dans la litérature. Puis, le modèle de la récupération d'énergie et la technique d'ordonnncement que nous avons adaptée sous contraintes temporelles et énergétiques sont donnés. Dans un second temps, nous y résumons les contributions principales de cette thèse. Nous les classifions selon trois familles d'ordonnancement: arrière-plan (en anglais, Background), vol de temps (en anglais, Slack Stealer) et préservation de bande totale (en anglais, Total bandwidth). Dans un dernier temps, nous rapportons les résultats d'une étude comparative des performances des différents algorithmes proposés.

8.1 Introduction

L'objectif d'un système autonome est d'assurer un fonctionnement perpétuel sans l'intervention humaine et ceci grâce à des batteries (ou tout autre type de réservoirs d'énergie), qui se rechargent en continu au cours de temps à partir d'une source d'énergie renouvelable. Des sources d'énergie alternatives existantes dans notre environnemnt peuvent être exploitées pour assurer cet objectif: c'est la récupération d'énergie. Elle consiste à convertir l'énergie de l'environnement et remplir un réservoir d'énergie formé d'une batterie ou d'un super-condensateur. Un tel réservoir d'énergie est requis parce que le système embarqué a besoin de fonctionner continuellement sans jamais manquer l'énergie disponible (dans le réservoir).

Donc utiliser une énergie renouvelable (solaire, piezo-électrique,..) pour alimenter des systèmes embarqués demande de concilier performances et consommations énergétiques. Et cette contrainte s'ajoute aux contraintes temporelles dans le cadre des systèmes temps réel d'où la violation de l'une d'elles conduira à la défaillance du système.

Dans la suite, nous aborderons le problème de l'ordonnancement des tâches périodiques conjointement avec des tâches apériodiques non critiques avec des contraintes énergétiques. Dans ce contexte, l'objectif d'un algorithme d'ordonnancement est de minimiser le temps de réponse des tâches apériodiques en donnant garantie aux tâches périodiques de s'exécuter dans ses propres échéances selon ED-H [8], dans les

systèmes centralisés monoprocesseur. La problématique décrite, reposant sur la minimisation du temps de réponse, étend le serveur classique dit en arrière plan au contexte de récupération d'énergie, en proposant deux nouveaux serveurs. Comme ces derniers offrent des performances limitées, nous proposons un nouveau serveur basé sur la technique du vol de temps creux avec contraintes énergétiques. Cette approche tend à tirer profit des temps creux libérés et de la quantité d'énergie excédée, de manière àaméliorer le temps de réponse des tâches apériodiques. Optimalité établie, il présente une implémentation relativement complexe. C'est pourquoi, nous nous proposons un nouveau serveur dit à préservation de bande, basé sur l'attribution d'échéances fictives avec une implémentation plus simple.

8.2 Ordonnancement temps réel

Nous considérons un système temps réel et nous nous intéressons au problème de l'ordonnancement monoprocesseur de tâches périodiques et de tâches apériodiques non critiques sous contraintes temporelles et énergétiques.

8.2.1 Modèle de tâches périodiques

Le modèle de tâches périodiques classique est le plus utilisé dans la modélisation des systèmes temps réel. Ce modèle permet de définir plusieurs paramètres pour une tâche. Ces paramètres sont de deux types : paramètres statiques relatifs àla tâche elle-même et les paramètres dynamiques relatif à chaque instance de la tâche. Dans un système de tâches périodiques Γ, une tâche périodique τ_i est caractérisée par les paramètres statiques $(r_i, C_i, D_i, T_i, E_i)$:

- r_i: sa date de réveil,
- C_i : sa durée d'exécution au pire cas (WCET),
- D_i : son échéance relative,
- T_i : sa période,
- E_i : sa demande énergétique au pire cas (WCEC).

Dans cette définition, la tâche τ_i fait son initiale requête à l'instant 0 et ses requêtes suivantes aux instants kT_i; $k = 1; 2; ...$ appelé dates de réveil r_i. T_i représente la séparation temporelle entre deux jobs successifs d'une tâche τ_i. Donc le k^{ime} job de la tâche τ_i est relâché à l'instant $r_i + (k-1)T_i$ et devra s'exécuter pour C_i unités de temps avant l'instant $r_i + (k-1)T_i + D_i$.

Les paramètres dynamiques d'une tâche déterminent le comportement pendant l'exécution de la tâche τ_i. Ces paramètres sont définis pour chaque instance ou job i noté J_i par :

- C_i : sa durée d'exécution au pire cas (WCET),
- d_i : $d_i = R_i + D_i$ son échéance absolue, c'est la date dont le dépassement entraîne une faute temporelle,
- E_i : sa demande énergétique au pire cas (WCEC),

Un job peut être interrompu et reprend exécution plus tard à n'importe quel instant et aucune perte de durée ou d'énergie est associée à une telle préemption.

8.2.2 Modèle de tâches apériodiques non critiques

Nous considérons de plus une configuration de tâches apériodiques non critiques Ap dans laquelle une tâche apériodique Ap_i est caractérisée par :

- a_i: la date coïncidant l'arrivée d'une tâche apériodique non critique,
- c_i: sa durée d'exécution au pire cas, connue au temps d'arrivée,
- e_i: sa demande énergétique.

f_i est le temps qui s'écoule entre la date d'arrivée et la fin d'exécution de la tâche apériodique.

Les tâches apériodiques sont rangées selon un ordre FIFO (le premier arrivé est le premier servi). Par la suite nous présentons différents serveurs permettant d'exécuter conjointement les tâches périodiques et apériodiques tout en garantissant le respect des échéances des tâches périodiques.

8.2.3 Algorithmes d'ordonnancement

Nous distinguons dans la théorie de l'ordonnancement temps réel à priorité dynamique, trois familles de serveurs qui traitent les tâches apériodiques conjointement avec les tâches périodiques. Nous les citons brièvement dans cette sous-section.

L'algorithme BG (Background Server)

Le serveur des tâches apériodiques *Background* (BG) [9] ordonnance les tâches apériodiques non critiques lorsqu'il n'y a aucune activité périodique au sein du système. Le principal avantage de ce serveur réside dans sa simplicité de mise en œuvre, en plus du fait qu'il garantisse que les tâches n'affecteront pas le comportement des tâches périodiques. Par contre, l'inconvénient majeur provient du fait que le temps de réponse aux requêtes apériodiques peut être grand.

L'algorithme EDL (Earliest Deadline as Late as possible)

Le serveur Earliest Deadline as Late as possible (EDL) repose sur la technique de Slack Stealing, capable d'assurer la gestion de tâches apériodiques au sein d'une application temps-réel. Il consiste à exécuter les tâches périodiques au plus tôt lorsqu'il n'y a aucune activité apériodique. Dans le cas contraire, chaque fois qu'une tâche apériodique survient, toutes les tâches périodiques sont ordonnancées au plus tard, dans le respect de l'ensemble des échéances des tâches. En d'autres termes, l'algorithme EDL tire profit de la laxité effective (c'est-à-dire de l'intervalle entre la date de fin d'exécution et l'échéance) des tâches périodiques, afin de minimiser le temps de réponse des apériodiques. Dans [10], Chetto et Chetto présentent une méthode simple pour déterminer la localisation et la durée des temps creux dans n'importe quelle fenêtre d'une séquence produite par EDL.

Il a été démontré [18] que le serveur EDL nécessite un calcul en-ligne des temps creux du processeur uniquement aux instants relatifs à l'occurrence d'une nouvelle tâche apériodique. La propriété fondamentale du serveur EDL est qu'il garantit le maximum de temps creux dans un intervalle donné pour n'importe quel ensemble de tâches. EDL a été démontré optimal [18].

Un autre résultat significatif réside dans la complexité de l'établissement de la séquence EDL. Le calcul en-ligne des temps creux [18] s'effectue en $O(\lceil \frac{R}{p} \rceil n)$ où n désigne le nombre de tâches périodiques, R l'échéance la plus éloignée parmi les tâches actives et p la plus petite période.

L'algorithme TBS (Total Bandwidth Server)

L'approche principale derrière le serveur Total Bandwith Server (TBS) [13, 14] est d'affecter une proche échéance possible à chaque tâche apériodique, de façon que l'utilisation totale du processeur n'excède pas une valeur maximale spécifiée de U_{ps}. Le nom du serveur provient du fait que, à chaque fois une tâche apériodique entre dans le système, la bande passante totale du serveur (en termes de temps d'exécution de CPU), est affectée si c'est possible. Cela se fait simplement en affectant une échéance appropriée à la tâche, qui est ordonnancée conjointement suivant l'algorithme EDF avec toutes les tâches périodiques. L'attribution de l'échéance est faite pour améliorer le temps de réponse de la tâche apériodique non critique, tout en conservant l'ordonnançabilité des tâches périodiques. Lorsque la *kème* tâche apériodique arrive à l'instant t = r_k, elle reçoit une échéance fictive: $d_k = max(r_k, d_{k-1}) + \frac{c_k}{U_{PS}}$, où c_k est le temps d'exécution de la tâche et U_{PS} est le facteur d'utilisation du serveur. Par définition, $d_0 = 0$. La tâche est par suite insérée dans la liste d'attente du système et ordonnancée par EDF comme n'importe quelle tâche périodique. Notons que, à l'attribution d'une nouvelle échéance d_k, la bande passante déjà attribuée aux précédentes tâches est prise en compte par la valeur de d_{k-1}.

Intuitivement, l'affectation des échéances fictives est telle que le rapport alloué aux requêtes apériodiques par EDF dans chaque intervalle de temps ne dépasse jamais U_{ps}. Par conséquent, l'ordonnançabilité

de l'ensemble des tâches périodiques sous TBS peut simplement être testée en vérifiant la condition suivante: $U_{pp}+U_{ps} \leq 1$ [14], où U_{pp} est le facteur d'utilisation des tâches périodiques. En dépit de sa simplicité, le TBS représente un bon rapport de performance/coût et peut facilement être optimisé afin de minimiser le temps de réponse de chaque requête apériodique.

L'algorithme TBS^* (Optimal Total Bandwidth Server)
L'algorithme TBS^* [31] est une amélioration de l'algorithme TBS dans le sens où il permet d'assigner à chaque tâche apériodique une échéance fictive plus petite que celle fournie par TBS. A chaque fois qu'une tâche apériodique arrive, le serveur TBS^* lui assigne en premier lieu une échéance selon l'algorithme TBS. Celui-ci va ensuite essayer de raccourcir au maximum cette échéance de manière à optimiser le temps de réponse des tâches apériodiques sans compromettre l'exécution des tâches périodiques. Si d_k^0 correspond à la première échéance assignée à la tâche apériodique Ap_k selon TBS, le processus de raccourcissement de cette échéance est appliqué de façon itérative jusqu'à ce qu'aucune amélioration ne soit plus possible, tout en conservant l'ordonnançabilité de la configuration des tâches périodiques. Par conséquent, si d_k^s est l'échéance assignée à la tâche apériodique Ap_k au pas s, l'ordonnançabilité est conservée en assignant à Ap_k une nouvelle échéance donnée par: $d_k^{s+1} = t + c_k + I_p(t + d_k^s)$, où t est le temps courant (correspondant à la date de réveil r_k de la tâche Ap_k ou bien à la date de terminaison de la tâche précédente), c_k est la durée d'exécution au pire cas requise par Ap_k, et $I_p(t + d_k^s)$ est l'interférence due aux jobs périodiques dans l'intervalle $[t, d_k)$. L'interférence périodique $I_p(t+d_k^s)$ est la somme de deux termes, $I_a(t+d_k^s)$ et $I_f(t+d_k^s)$. $I_a(t + d_k^s)$ correspond à l'interférence due aux jobs périodiques actifs au temps courant ayant des échéances strictement inférieures à d_k^s. $I_f(t + d_k^s)$ correspond à l'interférence future due aux jobs périodiques ayant leur date d'arrivée supérieure au temps courant t et ayant leur échéance inférieure à d_k^s. Leurs formules sont données ci-après,

$I_a(t, d_k^s) = \sum_{\tau_{active}; d_i < d_k^s} c_i(t)$

et

$I_f(t, d_k^s) = \sum_{i=1}^{n} max(0, \lceil \frac{d_k^s - next_{r_i}}{T_i} \rceil) C_i$, où $next_{r_i}$ est le prochain réveil de la tâche τ_i supérieur ou égal à t.

8.3 Ordonnancement temps réel et considération énergétiques

L'objectif d'un système autonome est d'assurer un fonctionnement perpétuel sans l'intervention humaine et ceci grâce à des batteries (ou tout autre type de réservoirs d'énergie), qui se rechargent en continu au cours de temps à partir d'une source d'énergie renouvelable. Des sources d'énergie alternatives existantes dans notre environnemnt peuvent être exploitées pour assurer cet objectif: c'est la récupération d'énergie. Elle consiste à convertir l'énergie de l'environnement et remplir un réservoir d'énergie formé d'une batterie ou d'un super-condensateur. Un tel réservoir d'énergie est requis parce que le système embarqué a besoin de fonctionner continuellement sans jamais manquer l'énergie disponible (dans le réservoir).

Donc utiliser une énergie renouvelable (solaire, piezo-électrique,..) pour alimenter des systèmes embarqués demande de concilier performances et consommations énergétiques. Et cette contrainte s'ajoute aux contraintes temporelles dans le cadre des systèmes temps réel où la violation de l'une d'elles conduira à la défaillance du système. Par la suite, nous devons présenter et spécifier l'algorithme d'ordonnancement ED-H qui est optimal pour le modèle RTEH.

8.3.1 Modèle d'énergie RTEH

Le modèle de récupération d'énergie (RTEH, Real Time Energy Harvesting system en Anglais), est composé d'une unité de traitement, d'un ensemble de tâches, d'une unité de stockage d'énergie appelée *réservoir*, d'une unité de récupération d'énergie et de la source d'énergie (voir Figure 3.1):

– Le *réservoir d'énergie* comme une batterie ou un super-condensateur dont le choix est dicté par les dynamiques du système, des contraintes de dimensionnement, de coût, etc.
– Le *récupérateur d'énergie* (harvester, en anglais) dont le choix dépend de la nature de l'énergie environmentale, de la quantité d'énergie requise, etc.
– Le *consommateur d'énergie* que représente ici le support d'exécution des tâches temps-réel. Il désigne la carte électronique construite autour d'un microcontrôleur ou d'un microprocesseur.

Soit $P_p(t)$ le *taux de recharge instantanée* produite par la source environmentale qui inclut toutes les pertes. L'énergie produite sur $[t_1, t_2)$ est notée $E_p(t_1, t_2) = \int_{t_1}^{t_2} P_p(t)dt$. Nous considérons que l'énergie produite dans une unité de temps et l'énergie consommée dans une unité de temps peuvent avoir lieu simultanément. Notre système utilise une unité de stockage d'énergie idéale avec une capacité nominale de C d'unités d'énergie. Le niveau d'énergie au temps t est notée $E(t)$. L'unité de stockage est complètement chargée initialement (c.à.d. $E(0) = C$). L'énergie stockée peut être utilisée à tout instant plus tard et ne perd pas d'énergie au fil de temps .

8.3.2 L'ordonnanceur ED-H

L'EDF classique est un ordonnanceur goulu puisqu'il exécute les jobs au plus tôt et dépense ainsi l'énergie stockée dans le réservoir ignorant les besoins futurs en énergie. Dans cette version d'EDF appelé EDS (Earliest Deadline as Soon as possible), le processeur n'est jamais inactif s'il y a au moins un job en attente pour s'exécuter. Supposons un ensemble de jobs temporellement faisable par EDF, la pénurie d'énergie pour un job J_i ne peut provenir que de l'exécution d'un job J_j qui s'exécute avant l'arrivée de J_i avec $d_j > d_i$. La pénurie d'énergie de J_i causée par J_j avec $d_j \leq d_i$ ne pourrait être évitée par aucun ordonnanceur. Il est évident que la clairvoyance relative à l'arrivée des jobs et à la production d'énergie va pouvoir aider EDF à anticiper une pénurie d'énergie et une violation d'échéance. Par conséquent, l'idée principale de l'ED-H est d'autoriser l'exécution des jobs seulement si aucune pénurie ne peut se produire.

8.3.2.1 la laxité énergétique

Une tâche est exécutée seulement si la laxité énergétique (la quantité d'énergie maximale consommée par une tâche tout en garantissant de l'énergie suffisante pour les tâches de haute priorité) est positive. La laxité énergétique du système nous permet de déterminer des bornes minorantes sur l'énergie future consommée et empêche la violation des échéances en cas de pénurie d'énergie. Si l'énergie n'est pas suffisante pour exécuter les tâches courantes ou qui vont arriver dans le futur, le système reste passif tant que la laxité temporelle est positive et le réservoir n'a pas atteint une valeur plafond pré-spécifiée. Ainsi, le système ne peut pas être actif si la laxité temporelle est positive et si la batterie est vide.

Nous introduisons ici de nouveaux concepts dynamiques pour l'analyse de faisabilité de jobs caractérisés par leur besoin énergétique.

Definition 18 *La laxité énergétique d'un job J_i à l'instant t_c est donnée par $SE_{J_i}(t_c) = E(t_c)+E_p(t_c, d_i)-g(t_c, d_i)$.*

$g(t_c, d_i)$ étant la demande énergétique entre t_c et d_i, $SE_{J_i}(t_c)$ représente la quantité d'énergie maximum qui pourrait être consommée dans $[t_c, d_i)$. S'il existe un job τ_i tel que $SE_{J_i}(t_c) = 0$, alors l'exécution entre t_c et d_i de tout job d'échéance supérieure à d_i provoquera une pénurie d'énergie pour J_i.

Definition 19 *Soit d l'échéance du job actif à l'instant t_c. Nous définissons la laxité énergétique de préemption de l'ensemble J à t_c, comme: $PSE_J(t_c) = \min_{t_c<r_i<d_i<d} SE_{J_i}(t_c)$.*

La laxité énergétique de préemption à l'instant t_c se définit comme la plus grande quantité d'énergie consommable par le job actif sans remettre en question la faisabilité des jobs susceptibles de le préempter.

8.3.2.2 la laxité temporelle

La demande processeur d'un ensemble de jobs J sur l'intervalle de temps $[t_1, t_2)$ se définit par la quantité de traitement requise entre t_1 et t_2, donnée par $h(t_1, t_2)$.

Definition 20 *Soit AT_i la durée d'exécution restante des jobs non terminés à t_c d'échéance inférieure ou égale à d_i. La laxité temporelle du job J_i à l'instant t_c est donnée par: $ST_{J_i}(t_c) = d_i - t_c - h(t_c, d_i) - AT_i$. La grandeur $ST_{J_i}(t_c)$ représente la quantité totale de temps processeur disponible dans $[t_c, d_i)$ après avoir exécuté tous les jobs d'échéance inférieure ou égale à d_i.*

Definition 21 *La laxité temporelle de l'ensemble de jobs J à l'instant courant t_c est donnée par: $ST_J(t_c) = \min_{d_i > t_c} ST_{J_i}(t_c)$.*

La laxité temporelle représente le temps processeur continu à partir de t_c pendant lequel le processeur pourrait rester inactif ou exécuter des jobs autres que ceux de l'ensemble τ. Le calcul de $ST_J(t_c)$ fait appel à la construction de la séquence EDL à partir de t_c décrite initialement dans [10].

ED-H consiste à permettre au processeur d'être inactif si la laxité temporelle est positive. En revanche, le processeur doit impérativement commencer à exécuter un job si la laxité temporelle est nulle. En outre, une laxité d'énergie de préemption positive signifie que le job actuellement actif peut continuer l'exécution. Et une laxité d'énergie de préemption nulle entraîne l'arrêt de l'exécution et impose la recharge de l'unité de stockage d'énergie.

Description de ED-H

- le processeur doit être inactif si le réservoir est vide ou si l'exécution d'un job empêche au moins un job au future d'être exécuté car cette exécution entraîne un pénurie d'énergie, la laxité énergétique de préemption étant insuffisante à l'instant t_c.

- le processeur ne peut rester inactif si le niveau d'énergie du réservoir se trouve au maximum ou si l'oisivité du processeur entraîne une violation d'échéance du fait d'une laxité temporelle nulle à l'instant t_c.

- le processeur peut indifféremment adopter l'état de veille ou d'activité si le réservoir n'est ni plein ni vide et si le système possède à la fois une laxité temporelle et une laxité d'énergie de préemption.

- nous commençons à recharger le réservoir lorsqu'il est vide ou lorsqu'il n'y a pas assez d'énergie pour garantir l'exécution possible de tous les futurs jobs.

8.3.3 Propriétés de ED-H

8.3.3.1 Analyse d'optimalité

Théorème 8.3.1 *[8] L'algorithme d'ordonnancement ED-H est optimal pour le modèle RTEH.*

8.3.3.2 Analyse de clairvoyance

Théorème 8.3.2 *[8] L'algorithme d'ordonnancement ED-H est D-omniscent.*

Nous savons qu'aucun algorithme en ligne ne peut être optimal sans une clairvoyance au moins égale à D unités de temps [108]. Pour prendre une décision à tout instant t_c, ED-H requière de connaître à la fois le processus d'arrivée des jobs et l'énergie récupérée sur les D unités de temps suivantes. Par suite, ED-H est D-omniscent.

8.3.3.3 Test d'ordonnançabilité

Nous présentons ci-après un test pour vérifier que les échéances de jobs de J sont respectées, étant donné un réservoir d'énergie caractérisé par sa capacité et un récupérateur d'énergie caractérisé par une puissance de production instantanée $P_p(t)$. Nous donnons une condition nécessaire et suffisante pour l'ordonnaçabilitéde ED-H. Comme ED-H est optimal, la condition est aussi une condition de faisabilité.

Théorème 8.3.3 *[8] Un ensemble de jobs J conforme au modèle RTEH est faisable si et seulement si $SST_J \geq 0$ et $SSE_J \geq 0$.*

L'objectif du test de faisabilité est de prédire si le temps et l'énergie seront suffisantes pour respecter les exigences de temps de tous les jobs. Dans la conception de systèmes temps réel composées de tâches périodiques bien connus, sans contraintes énergétiques, nous effectuons un test hors ligne et nous utilisons un algorithme en ligne pour ordonnancer et exécuter les jobs. Pour le modèle RTEH, le test peut être effectué hors ligne seulement si les jobs sont des instances de tâches périodiques et puis si le profil énergétique est caractérisé pour toute la durée de vie de l'application. Dans tous les autres cas, le test d'ordonnançabilité doit être réalisé lors de l'exécution en considérant la technique de prédiction. Cela signifie que le test d'ordonnançabilité est effectué afin de vérifier que tous les jobs libérés sur la fenêtre de temps suivante seront faisablement ordonnancés. Sinon, une décision doit être prise dans le but de rendre le système faisable en éliminant des jobs et par conséquent obtenir une moindre qualité de service.

8.4 Nos contributions

Cette section consiste à étudier le problème d'ordonnancement lié à un ensemble hybride de tâches composé de tâches périodiques et de tâches apériodiques souples dans le contexte de la récupération d'énergie. Les algorithmes traités dans la littérature avec des contraintes de récolte d'énergie considèrent seulement les ensembles de tâches périodiques. Cependant, de nombreuses applications nécessitent plusieurs types de tâches.
Notre étude est menée avec le modèle de système qui a été décrit dans la section précédente. L'objectif est de fournir un algorithme d'ordonnancement qui permet de minimiser le temps de réponse des tâches apériodiques. Nous supposons que les tâches périodiques sont ordonancées de manière préemptive selon l'optimal ED-H. Dans la suite, nous présentons quatre serveurs apériodiques appartenent à différentes familles d'ordonanncement, respectivement nommés BES, BEP, SSP et TB-H qui est étendu à l'optimal $TB^* - H$.

8.4.1 L'algorithme BES

Les tâches apériodiques sont exécutées avec BES, seulement s'il n'y a pas de tâches périodiques prêtes et si le réservoir est entièrement plein. Ce qui signifie que la tâche apériodique ne consomme que l'énergie qui doit être gaspillée s'il n'y a pas de tâches périodiques au cours d'exécution. A noter que les tâches périodiques sont ordonnancées selon ED-H.

8.4.2 L'algorithme BEP

Dans cet algorithme, une tâche apériodique est exécutée si et seulement si a) le réservoir d'énergie n'est pas vide, b) aucune tâche périodique est en cours d'exécution, et c) tant que cette exécution n'entraîne pas une pénurie d'énergie pour les futures tâches périodiques. Si à un instant t, aucune tâche périodique n'est prête à être servie et de tâches apériodiques sont en attente, une tâche apériodique est élue pour s'exécuter selon FIFO. Dans ce modèle, les tâches périodiques sont aussi ordonnancées selon l'algorithme ED-H.

Considérons à présent le même ensemble de tâches hybrides que celui utilisé précédemment pour illustrer le serveur BES.

8.4.3 L'algorithme SSP

L'algorithme proposé SSP (Slack Stealing with energy Preserving) est une stratégie qui s'appuie sur l'algorithme EDL avec ED-H. Il utilise la laxité temporelle pour exécuter les tâches apériodiques le plus proche possible et la laxité d'énergie pour exécuter les fulures tâches périodiques sans entraîner une pénurie d'énergie. Si aucune tâche apériodique n'arrive, les tâches périodiques sont ordonnancées selon ED-H. Et si une tâche apériodique arrive, il utilise le surplus d'énergie et la non-activité du processeur pour servir les tâches apériodiques. Si le réservoir d'énergie n'est pas vide, et si le système possède à la fois une laxité temporelle et une laxité énergétique, la tâche apériodique est servie selon FIFO tant que la liste des tâches périodiques est non vide.

Optimalité de SSP
L'optimalité du SSP est énoncée dans les théorèmes suivants.

Théorème 8.4.1 *Toutes les tâches périodiques respectent leurs échéances lorsqu'elles sont ordonnancées selon ED-H avec le SSP pour servir les tâches apériodiques (voir Chapitre 5).*

Théorème 8.4.2 *Pour tout ensemble de tâches périodiques ordonnancé selon ED-H et un ensemble de tâches périodiques traitées selon l'ordre FCFS, l'algorithme SSP minimise le temps de réponse de chaque tâche apériodique, parmi tous les algorithmes qui sont garantis à respecter leurs échéances (voir Chapitre 5).*

8.4.4 L'algorithme TB-H

L'algorithme *TB-H* est une stratégie qui s'appuie sur le serveur Total Bandwith Server (TBS)[14] permettant de servir des tâches apériodiques en améliorant leur temps de réponse. A chaque fois qu'une tâche apériodique entre dans le système, le serveur TBS lui assigne une échéance fictive en fonction de sa largeur de bande CPU U_{ps} telle que $U_{pp} + U_{ps} \leq 1$ [14]. Les tâches apériodiques ayant une échéance fictive assignée seront ordonnancées conjointement avec les tâches périodiques selon EDF. En cas de contraintes énergétiques, l'échéance fictive calculée est vérifiée suivant $U_{ep} + U_{es} \leq P_p$, où U_{es} représente la largeur de bande énergétique du serveur.

Théorème 8.4.3 *Soit une configuration de n tâches périodiques et une configuration de tâches périodiques servies par un serveur* TBS*, l'échéance fictive d'une tâche apériodique Ap est calculée comme suit:*

$$\tilde{d_k} = max(r_k, \tilde{d_{k-1}}) + \lceil \frac{\frac{E_k}{U_{es}} - E(r_k)}{P_p} \rceil$$

U_{es} est la bande passante énergétique telle que $U_{es} = P_p - U_{ep}$.
Par conséquent, nous présentons deux algorithmes basés sur TBS: TB-H et TB*-H.
Ces deux algorithmes diffèrent par le calcul de leur échéance fictive:

1. avec TB-H: Obtenir l'équation de l'échéance fictive de la tâche apériodique Ap est résumée par:

$$\left\{ \begin{array}{c} d_k = max(r_k, d_{k-1}) + \frac{c_k}{U_{ps}} \\ \tilde{d_k} = max(r_k, \tilde{d_{k-1}}) + \lceil \frac{\frac{E_k}{U_{es}} - E(r_k)}{P_p} \rceil \\ d_k^f = max(d_k^{s+1}, \tilde{d_k}) \end{array} \right\}$$

2. avec $TB^* - H$: Obtenir l'équation de l'échéance fictive de la tâche apériodique Ap est résumée par:

$$\left\{ \begin{array}{c} d_k^{s+1} = t + c_k + I_p(t + d_k^s) \\ \tilde{d_k} = max(r_k, \tilde{d_{k-1}}) + \lceil \frac{\frac{E_k}{U_{es}} - E(r_k)}{P_p} \rceil \\ d_k^f = max(d_k^{s+1}, \tilde{d_k}) \end{array} \right\}$$

Les algorithmes TB-H et $TB^* - H$ ont le même principe; ils se diffèrent seulement par la formule du calcul de l'échéance fictive. Donc, une fois l'échéance fictive de la tâche apériodique est calculée soit avec TB-H ou $TB^* - H$, elle est ajoutée à la liste des tâche apériodiques. A chaque instant t, la tâche ayant la plus proche échéance sera choisie pour être exécutée, parmi la liste des tâches périodiques prêtes et la liste des tâches apériodiques prêtes. Seulement si la laxité énergétique du système est positive et si l'énergie disponible dans la batterie est suffisante, cette tâche sera exécutée.

8.4.5 Contexte de simulations et critères usuels d'évaluation

Contexte de simulations

Pour évaluer comparativement la performance des stratégies précédemment décrites, des simulations sont effectuées. Notre simulateur supporte les données suivantes. Un générateur de tâches périodiques prend en entrée les paramètres suivants:

- n : le nombre de tâches périodiques générées constituant la configuration,
- $PPCM_{max}$: le ppcm maximal des périodes des tâches périodiques constituant la configuration,
- U_{pp} : la charge processeur associée aux tâches périodiques,

En sortie, le générateur génère une configuration de tâches $\tau = \tau_i(C_i, D_i, T_i, E_i), i = 1...n$.

- Les tâches périodiques sont indépendantes et préemptables.
- Les périodes des tâches T_i sont aléatoirement choisies.
- Les durées d'exécution des tâches au pire-cas C_i sont générées aléatoirement et le choix dépend de $U_{pp} = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
- Les demandes énergétiques des tâches E_i sont générées aléatoirement et le choix dépend de $\overline{U}_{ep} = \sum_{i=1}^{n} \frac{E_i}{T_i}$.

Un générateur de tâches apériodiques prend en entrée les paramètres suivants:

- m : le nombre de tâches apériodiques générées constituant la configuration,
- U_{ps} : la charge temporelle dédiée aux tâches apériodiques est définie par l'utilisateur,
- U_{es}: la charge énergétique dédiée aux tâches apériodiques est définie par l'utilisateur,,

En sortie, le générateur génère une configuration de m tâches apériodiques non critiques $Ap = Ap_i(a_i, c_i, e_i), i = 1...n$

- les temps a_i coïncidant avec l'arrivée d'une tâche apériodique non critique,
- Les durées d'exécution des tâches au pire-cas c_i sont générées aléatoirement et le choix dépend de U_{ps}.
- Les demandes énérgétiques des tâches e_i sont générées aléatoirement en fonction de U_{es}.

Donc, le simulateur consiste à ordonnancer en-ligne ces différentes configurations de tâches pour les algorithmes d'ordonnancement: BES, BEP, SSP, TB-H et TB*-H.

Critères usuels d'évaluation

Nous présentons les critères utilisés pour évaluer les algorithmes proposés.

Le temps de réponse normalisé moyen : représente la moyenne, sur l'ensemble des tâches apériodiques non critiques exécutées, du temps nécessaire à une tâche pour s'exécuter, normalisé par rapport à sa durée d'exécution: $ART_i = \frac{f_i - a_i}{c_i}$,

Le temps de gigue normalisé moyen : représente la moyenne, sur l'ensemble des tâches apériodiques non critiques exécutées, du temps nécessaire à une tâche pour démarrer, normalisé par rapport à son temps de réponse: $AJT_i = \frac{s_i - a_i}{f_i - a_i}$; s_i étant la date écoulant depuis l'arrivée de la tâche jusqu'à son début d'exécution.

Le temps de latence normalisé moyen : représente la moyenne, sur l'ensemble des tâches apériodiques non critiques exécutées, du temps écoulant entre le début d'exécution de la tâche et sa fin effective, normalisé par rapport à sa durée d'exécution: $ALT_i = \frac{f_i - s_i}{c_i}$.

– Taux de préemption: représente la moyenne du rapport entre le nombre de préemptions et le nombre des instances traitées. Ce facteur permet d'évaluer les surcoûts dus aux changements de contextes. Par définition, il y a préemption lorsque l'exécution d'une tâche est interrompue au profit d'une tâche plus prioritaire. L'exécution de la tâche préemptée est alors reprise plus tard dans le temps: $PE = \frac{nombre\,de\,premptions}{total\,nombre\,des\,instances}$
– Surcoût (ou overhead, en anglais): ici, nous considèrons le surcoût dû au calcul fréquent de la laxité temporelle et de la laxité énergétique.

8.4.6 Synthèse de travail et conclusion

BES et BEP

– BEP améliore significativement les performances du BES sans frais supplémentaires.
– Les deux serveurs sont simples à implémenter.
– Les performances sont les plus médiocres en termes de temps de réponse, de temps de gigue et de temps de latence quelle que soit l'utilisation temporelle ou énergétique, et nécessitent une optimisation.

SSP

– Le SSP est optimal; l'optimisation consiste à fournir le temps de réponse apériodique le plus court parmi tous les serveurs de tâches apériodiques.
– SSP présente de meilleures performances que BES et BEP pour tous les modèles de récupération d'énergie et quelle que soit l'utilisation temporelle ou énergétique
Contrairement aux serveurs basés sur l'arrière plan, le serveur SSP n'est pas si simple à implémenter. C'est parce que nous devons considérer les deux notions dynamiques clés: laxité temporelle et laxité énergétique. Sa complexité est pseudo-polynomiale, ce qui augmente les surcoûts présentés.

TB-H et TB*-H

– Expérimentalement, les deux serveurs offrent un bon comportement en termes de complexité, d'implémentation et de performance.
– Ils offrent un soût supplémentaire quant au serveur classique TBS prénté sans contraintes énergétiques par Butazzo.

LIST OF PUBLICATIONS

International conferences proceedings

[1]. R. El Osta, M. Chetto, *Sustainable Quality of Service for real-time jobs in Autonomous Computing Devices*, IEEE Computer Society. 2014 International Conference on Future Internet of Things and Cloud, pp.453-457, August 2014, Barcelona, Spain.

[2]. R. El Osta, M. Chetto and H. El Ghor, *Minimizing the aperiodic responsiveness in Energy Harvesting Devices*, 12th IEEE International Symposium on Industrial Embedded Systems, 14 - 16 June 2017, Toulouse, France.

[3]. R. El Osta, M. Chetto, H. El Ghor and R. Hage, *Real-Time Scheduling of aperiodic tasks in Energy Harvesting Devices*, International Conference on Sensors, smart and Emerging Technologies, 12-14 September 2017, Beirut, Lebanon.

[4]. R. El Osta, M. Chetto and R. Hage, *Stratégies d'ordonnancement temps réel pour les systèmes embarqués autonomes en énergie*, 20th International Conference of Lebanese Association for the Advancement of Sciences (LAAS), 27-29 March 2014, Beirut, Lebanon.

[5]. R. El Osta, M. Chetto and H. El Ghor, *Aperiodic Task Servicing in Real-Time Energy Harvesting Devices*, 23th International Conference of Lebanese Association for the Advancement of Sciences (LAAS), 6-7 April 2017, Beirut, Lebanon.

National conferences

[6]. R. EL Osta, R. Hage Chehade, M. Chetto, H. El Ghor, *Real-Time Scheduling with Renewable Energy in a Uniprocessor Platform*, Secondes Journées Franco-Libanaises, 22-25 octobre 2013, Dunkerque, France.

International scientific journals

[7]. R. El Osta, M. Chetto and H. El Ghor, *Optimal Slack Stealing Servicing for Real-Time Energy Harvesting Systems*. Submitted to IEEE Transactions on Sustainable Computing in 2016.

BIBLIOGRAPHY

[1] S. Kosunalp. A performance evaluation of solar energy prediction approaches for energy-harvesting wireless sensor networks. *3rd International Conference on Advanced Technology and Sciences (ICAT'16)*, 4:424–427, September 2016. V, 29, 30

[2] S. Basagni, M. Y. Naderi, C. Petrioli, and D. Spenza. *Wireless Sensor Networks With Energy Harvesting*. in Mobile Ad Hoc Networking Cutting Edge Directions, Chapter 20, John Wiley and Sons Inc., Hoboken, NJ, 2012. VII, 33

[3] Jens Eliasson. *Low-Power Design Methodologies for Embedded Internet Systems*. PhD thesis, Luleå University of Technology, 2008. VII, 35

[4] A. Kansal, J. Hsu, M. Srivastava, and V. Raghunathan. Harvesting aware power management for sensor networks. *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 651–656, 2006. 1

[5] S. Zahedi A. Kansal, J. Hsu and M.B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems*, 6(4), 2007. 1

[6] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications 2nd edn*. Springer, Berlin, 2005. 1

[7] J. Liu. *Real-Time Systems*. Prentice Hall, 2000. 1

[8] M. Chetto. *Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems*. IEEE Trans. on Emerging Topics in Computing, 2014. 1, 45, 51, 52, 65, 66, 117, 122, 123

[9] J. p. Lehozcky, L. Sha, and K. k. Strosnider. Enhanced aperiodic responsivess in hard real-time environments. In *inproceedings of the 13th IEEE Real-Time Systems Symposium*, pages 261–270, 1987. 2, 14, 119

[10] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *In IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989. 2, 15, 17, 119, 122

[11] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992. 2, 15

[12] M. Silly-Chetto. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *Real-Time Systems*, 17(1):1–25, 1999. 2

[13] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. pages 2–11, San Juan, Puerto Rice,, 7-9 December 1994. Proceedings Ieee Real-Time Systems Symposium. 2, 15, 20, 96, 119

[14] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, March 1996. 2, 15, 20, 119, 120, 124

[15] G. C. Buttazo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time enviroments. In Italy Como, editor, *Proceedings of the 3rd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'97)*, pages 39–48, September 1997. 2, 15, 22

[16] J. A. Stankovic. A serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19, 1988. 8

[17] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications 2nd edn*. Springer, Berlin, 2011. 10

[18] M. Silly. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *The Journal of Real-Time Systems*, 17:1–25, 1999. 10, 16, 17, 18, 66, 119

[19] S. Sahni E. Horowitz. Exact and approximate algorithms for non identical processors. *JACM*, 23(2):317–327, 1976. 12

[20] and J.-D. Ullman A.-V.Aho, J.-E. Hopcroft. *The design and analysis of computer algorithms*. Addison-Wesley, Reading, Mass, 1974. 12

[21] T. Bouchentouf. *Ordonnancement sous contraintes de prècédence dans les systèmes temps réel*. PhD thesis, Université de Nantes, 1991. 12

[22] C. l. Liu and J. w. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973. 12, 13, 21, 96

[23] J.-R. Jackson. Scheduling a production line to minimize maximum tardiness. Research report 43, management science research project, University of California, Los Angeles, 1955. 13

[24] O. Serlin. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the Joint Computer conference*, pages 925–932, 1972. 13

[25] ML. Dertouzos. Control robotics: the procedural control of physical processes. In *Proc Int Federat Inform Process Cong*, Proc. of 25th Euromicro Conference on Real-Time Systems, 1974. 13

[26] P. Richard F. Ridouard and F. Cottet. Ordonnancement de tches indÃ©pendantes avec suspension. *In Proceedings of the 13th International Conference on Real-Time Systems*, 2005. 13

[27] L. George, P. Muhlethaler, and N. Rivierre P. Optimality and non-preemptive real- time scheduling revisited. Technical report, Research Report RR-2516, INRIA, Le Chesnay Cedex, France, 1995. 13

[28] R.-R. Howell S.-K. Baruah and L.-E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems 2*, pages 301–324, 1990. 13, 14

[29] T.-M. Ghazalie and T.-P. Baker. Aperiodic servers in a deadline scheduling environment. *The Journal of Real-Time Systems*, 9:21–36, 1995. 15

[30] L. A. GutiÃ©rrez, C. A. Franco, R. R. Jacinto, and C. A. GutiÃ©rrez. Minimizing the response times of aperiodic tasks in hard real-time systems with edf. *Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA'06)*, 0-7695-2569-5/06, 2006. 15

[31] G. c. Buttazzo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Software Engineering*, 48(10):1035 – 1052, Oct 1999. 15, 23, 120

[32] How Energy Harvesting Can Keep the IoT Powered Up and Growing. 2015. 28

[33] S. Sudevalayam and P. Kulkarni. Energy harvesting sensor nodes: survey and implications. volume 13, pages 443–461. IEEE Commun. Surv. Tut., 2011. 28, 33

[34] P. D. Mitcheson, E. M. Yeatman, G. K. Rao, A. S. Holmes, and T. C. Green. Energy harvesting from human and machine motion for wireless electronic devices, 96(9). In *Proceedings of the IEEE*, pages 1457–1486, September 2008. 29, 32

[35] E. O. Torres. *An Electrostatic CMOS/BiCMOS Li Ion Vibration-based HarvesterCharger IC*. PhD thesis, Georgia Institute of Technology, May 2010. 29, 30

[36] E. O. Torres and G. A. Rincon-Mora. Long-lasting, self-sustaining, and energyharvesting system-in-package (sip) wireless micro-sensor solution. In *Proceedings of INCEED 2005*, NC,, July 24-30 2005. Charlotte. 30

[37] F. Yildiz. Potential ambient energy-harvesting sources and techniques. *The Journal of Technology Studies*, 35(1):40–48, 2009. 30

[38] S. Chalasani and J. M. Conrad. A survey of energy harvesting sources for embedded systems. In *Proceedings of the IEEE Southeastcon 2008*, pages 442–447, April 2008. 30, 31

[39] S. J. Roundy. *Energy Scavenging for Wireless Sensor Nodes with a Focus on Vibration to Electricity Conversion*. PhD thesis, University of California at Berkeley, Berkeley, CA, May 2003. 30

[40] V. R. Challa, M. G. Prasad, and F. T. Fisher. Towards an autonomous selftuning vibration energy harvesting device for wireless sensor network applications. *Journal of Smart Materials and Structures*, 20(2):1–11, February 2011. 30

[41] J. G. Rocha, L. M. Goncalves, P. F. Rocha, M. P. Silva, and S. LancerosMendez. Energy harvesting from piezoelectric materials fully integrated in footwear. *IEEE Transactions on Industrial Electronics*, 57(3):813–819, March 2010. 30

[42] M. Zhu and E. Worthington. Design and testing of piezoelectric energy harvesting devices for generation of higher electric power for wireless sensor networks. In *Proceedings of the IEEE Sensors*, pages 699–702, Christchurch, New Zealand,, October 25-28 2009. 30

[43] R. Moghe, Y. Yang, F. Lambert, and D. Divan. A scoping study of electric and magnetic field energy harvesting for wireless sensor networks in power system applications. In *Proceedings of IEEE ECCE 2009*, pages 3550–3557, San Jose, CA,, September 20-24 2009. 30, 31

[44] M. K. Stojcev, M. R. Kosanovic, and L. R. Golubovic. Power management and energy harvesting techniques for wireless sensor nodes. In *Proceedings of TELSIKS 2009*, pages 65–72, Niffs, Serbia,, October 7-9 2009. 30

[45] C. He, A. Arora, M. E. Kiziroglou, D. C. Yates, D. Oï¿½Hare, and E. M. Yeatman. Mems energy harvesting powered wireless biometric sensor. In *Proceedings of BSN 2009*, pages 207–212, Berkeley, CA, June 3-5 2009. 30

[46] M. E. Kiziroglou, C. He, and E. M. Yeatman. Flexible substrate electrostatic energy harvester. *IEEE Electronics Letters*, 46(2):166–167, January 2010. 30

[47] J. C. Park, D. H. Bang, and J. Y. Park. Micro-fabricated electromagnetic power generator to scavenge low ambient vibration. *IEEE Transactions on Magnetics*, 46(6):1937–1942, June 2010. 30

[48] O. Zorlu, E. T. Topal, and H. Kuï¿½andlah. A vibration-based electromagnetic energy harvester using mechanical frequency up-conversion method. *IEEE Sensors Journal*, 11(2):481–488, February 2011. 30

[49] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. B. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of ACM/IEEE IPSN 2005*, pages 457–462, Los Angeles, CA,, April 25-27 2005. 31, 35

[50] S. Ayazian, E. Soenen, and A. Hassibi. A photovoltaic-driven and energyautonomous cmos implantable sensor. In *Proceedings of IEEE VLSIC*, pages 148–149, June 2011. 31

[51] M. Barnes, C. Conway, J. Mathews, and D. K. Arvind. Ens: An energy harvesting wireless sensor network platform. In *Proceedings of ICSNC 2010*, pages 83–87, August 2010. 31

[52] C. Chen and P. H. Chou. Duracap: A supercapacitor-based, power-bootstrapping, maximum power point tracking energy-harvesting system. In *Proceedings of ISLPED 2010*, pages 33–318, August 2010. 31

[53] Y. Chen, Q. Wang, J. Gupchup, and A. Terzis. Tempo: An energy harvesting mote resilient to power outages. In *Proceedings of IEEE LCN 2010*, pages 933–934, October 2010. 31

[54] N. S. Hudak and G. G. Amatucci. Small-scale energy harvesting through thermoelectric, vibration, and radio frequency power conversion. *Journal of Applied Physics*, 103(10):1–24, May 2008. 31

[55] J. G. Webster. The measurement, instrumentation, and sensors handbook. The electrical engineering handbook series. CRC Press, December 1998. 31

[56] R. Abbaspour. A practical approach to powering wireless sensor nodes by harvesting energy from heat flow in room temperature. In *Proceedings of IEEE ICUMT*, pages 178–181, October 2010. 31

[57] X. Lu and S. h. Yang. Thermal energy harvesting for wsns. In *Proceedings of IEEE SMC 2010*, pages 3045–3052, October 2010. 31

[58] R. Heer, J. Wissenwasser, M. Milnera, L. Farmer, C. Hoï¿½pfner, and M. Vellekoop. Wireless powered electronic sensors for biological applications. In *Proceedings of IEEE EMBC 2010*, pages 700–703, September 4, 2010. 32

[59] S. Mandal, L. Turicchia, and R. Sarpeshkar. A low-power, battery-free tag for body sensor networks. *IEEE Pervasive Computing*, 9(1):71–77, March 2010. 32

[60] F. Fei, J. D. Mai, and A W. J. Li. wind-flutter energy converter for powering wireless sensors. *Sensors and Actuators A: Physical*, 173(1):163–171, January 2012. 32

[61] S. P. Matova, R. Elfrink, R. J. M. Vullers, and R. van Schaijk. Harvesting energy from airflow with a michromachined piezoelectric harvester inside a helmholtz resonator. *Journal of Micromechanics and Microengineering*, 21(10):1–6, 2011. 32

[62] C. y. Sue and N. c. Tsai. Human powered mems-based energy harvest devices. *Applied Energy*, 93:390–403, 2012. 32

[63] C. Xu, C. Pan, Y. Liu, and Z. L. Wang. Hybrid cells for simultaneously harvesting multi-type energies for self-powered micro/nanosystems. *Nano Energy*, 1(2):259–272, 2012. 32

[64] F. Davis and S. P. J. Higson. Biofuel cells-recent advances and applications. *Biosensors and Bioelectronics*, 22(7):1224–1235, 2007. 32

[65] C. B. Williams, C. Shearwood, M. A. Harradine, P. H. Mellor, T. S. Birch, and R. B. Yates. Development of an electromagnetic micro-generator. In *Proceedings of the IEEE Circuits, Devices and Systems, 148(6)*, pages 337–342, December 2001. 32

[66] S. Sherrit. The physical acoustics of energy harvesting. In *Proceedings of IEEE IUS 2008*, pages 1046–1055, November 2008. 32

[67] F. Liu, A. Phipps, S. Horowitz, K. Ngo, L. Cattafesta, T. Nishida, and M. Sheplak. Acoustic energy harvesting using an electromechanical helmholtz resonator. *Journal of the Acoustical Society of America*, 123(4):1983–1990, 2008. 32

[68] A. Denisov and E. Yeatman. Stepwise microactuators powered by ultrasonic transfer. In *Proceedings of the Eurosensors XXV*, Athens, Greece, September 2011. 32

[69] Y. Zhu, S. O. R. Moheimani, and M. R. Yuce. A 2-dof mems ultrasonic energy harvester. *IEEE Sensors Journal*, 11(1):155–161, January 2011. 32

[70] R. Casas and O. Casas. Battery sensing for energy-aware system design. *Computer*, 38(11):48–54, 2005. 34

[71] D. Linden and T. Reddy. *Handbook of Batteries*, volume 3. 3rd ed. McGraw-Hill, 2001. 34

[72] P. Notten V. Pop, H. Bergveld and P. Regtien. State-of-the-art of battery state-of-charge determination. *Measurement Science and Technology*, 16(11):R93–R110(1), December 2005. 34

[73] T. Zhu, Z. Zhong, Y. Gu, T. He, and Z. l. Zhang. Leakage-aware energy synchronization for wireless sensor networks. In *Proceedings of ACM MobiSys 2009*, pages 319–332, New York, 2009. 35

[74] I. Buchmann. Batteries in a portable world: A handbook on rechargeable batteries for non-engineers. *Burnaby: Cadex Electronics*, 1997. 35

[75] F. Simjee and P. H. Chou. Everlast: Long-life, supercapacitor-operated wireless sensor node. In *Proceedings of ISLPED 2006*, pages 197–202, Tagernsee, Germany, October 4-6 2006. 35

[76] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. In *Proceedings of ACM/IEEE IPSN 2006*, pages 407–415, Nashville, TN,, April 19-21 2006. 35

[77] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of ACM/IEEE IPSN 2005*, pages 463–468, April 25-27 2005. 35

[78] C. Park and P. H. Chou. Ambimax: Autonomous energy harvesting platform for multi-supply wireless sensor nodes. In *Proceedings of IEEE SECON 2006*, pages 168–177, Reston, VA, September 25-28 2006. 35

[79] P. Corke, T. Wark, P. Valencia, and P. Sikka. Long-duration solar-powered wireless sensor networks. In *Proceedings of EmNets 2007*, pages 33–37, Cork, Ireland,, June 25-26 2007. 35

[80] C. Park and P. H. Chou. Power utility maximization for multiple-supply systems by a load-matching switch. In *Proceedings of ISLPED 2004*, pages 168–173, August 9-11 2004. 35

[81] `http://micropelt.com/downloads/datasheet_mpg_d655.pdf`, 2017. [Online; accessed 19-May-2017]. 36

[82] `http://www.mide.com/`, 2017. [Online; accessed 19-May-2017]. 36

[83] EnOcean. The enocean gmbh, germany. `http://www.enocean.com/en/home/`, 2017. [Online; accessed 19-May-2017]. 36

[84] N. Navet and B. Gaujal. *Ordonnancement temps réel et minimisation de la consommation d'énergie*, volume 2 of *Systèmes temps réel*, chapter 4. Hermès, 2006. 42, 43

[85] F. Gruian. *Energy-centric scheduling for real-time systems*. PhD thesis, Lund Institute of Technology, 2002. 43

[86] M. E. Salhiene, L. Fesquet, and M. Renaudin. Adaptation dynamique de la puissance des systÃ¨mes embarquÃ©s : les systÃ¨mes asynchrones surclassent les systÃ¨mes synchrones. *4iÃ¨me journÃ©e d'Ã©tudes Faible Tension Faible Consommation (FTFC'03)*, pages 51–58, 2003. 43

[87] A.Bogliolo L. Benini and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *In IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000. 43

[88] A. J. Demers F. Yao and S. Shenker. A scheduling model for reduced cpu energy. In *In IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995. 43

[89] H. Aydin, R. Melhem, D. MossÃ©, and P. Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Euromicro Conference on Real-Time Systems*, pages 225–232, 2001. 43

[90] D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. *In ASPDAC'03*, pages 653–658, 2004. 43

[91] J. Xing W. Nisar N. Min-allah, Y. Wang and A. Kazmi. Towards dynamic voltage scaling in real-time systems - a survey. *In IJCSES International journal of Computer Sciences and Engineering Systems*, 1(2), 2007. 43

[92] A. Allavena and D. Mosse. Scheduling of frame-based embedded systems with rechargeable batteries,. *in: Workshop on Power Management for Real-time and Embedded systems*, 2001. 43

[93] L. Thiele C. Moser, D. Brunelli and L. Benini. Real-time scheduling with regenerative energy. In 18th Euromicro Conference Real-Time Systems, 2006., 2006. 44

[94] L. Thiele C. Moser, D. Brunelli and L. Benini. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems*, 37(3):233–260, 2007. 44

[95] Eran Sharon Daniel Levin and Simon Litsyn. Lazy scheduling for ldpc decoding. *IEEE COMMUNICATIONS LETTERS*, 11(1), January 2007. 45

[96] R. R. Kompella and A. C. Snoeren. Practical lazy scheduling in sensor networks. Los Angeles, California, USA, November 2003. 45

[97] C. R. Das P. Mohapatra, C. Yu and J. Kim. 45

[98] C. Rusu, R. Melhem, and D. MossÃ©. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing*, 1(2):271–283, April 2005. 45

[99] J.-J. Chen C. Moser and L. Thiele. Reward maximization for embedded systems with renewable energies. In *Proceedings of 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 247–256, 2008. 45

[100] M. Chetto and H. El Ghor. *Real-time Scheduling of periodic tasks in a monoprocessor system with a rechargeable battery*. In 17th IEEE In- ternational Symposium on Real Time Systems (RTSS 2009), wip session, Washington, 2009. 45

[101] M. Chetto H. El Ghor and R. Hage Chehade. A real-time scheduling framework for embedded systems with environmental energy harvesting. Technical report, University of Nantes, IRCCyN, 2010. 45, 60

[102] H. El Ghor. *Ordonnancement monoprocesseur pour les applications temps réel récupérant l'énergie ambiante*. PhD thesis, Université de Nantes, 2012. 45

[103] H. El Ghor, M. Chetto, and R. Hage Chehade. A nonclairvoyant real-time scheduler for ambient harvesting sensors. *International Journal of Distributed Sensor Networks*, 2652:11, 2013. 45

[104] Y. Abdeddaim and D. Masson. Real-time scheduling of energy harvesting embedded systems with timed automata. Proc. of 18th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications, 2012. 45

[105] Y. Chandarli Y. Abdeddaim and D.Masson. The optimality of pfpasap algorithm for fixed-priority energy-harvesting real-time systems. In *Proc. of 25th Euromicro Conference on Real-Time Systems*, 2013. 45

[106] M. Chetto, D. Masson, and S. Midonnet. Fixed priority scheduling strategies for ambient energy-harvesting embedded systems. Chine, 2011. Chenghu. 45

[107] Y. Chandarli. *Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués*. PhD thesis, Université Paris-Est, 2014. 45

[108] M. Chetto and A. Queudet. Clairvoyance and online scheduling in real-time energy harvesting systems. *Real-Time Systems Journal*, 50(2):179–184, 2014. 51, 122

[109] G. Koren and D. Shasha. Skip-over algorithms and complexity for overloaded systems thatv allow skips. In *inproceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, 1995. 52

[110] M. Abdallah. *Ordonnancement temps réel pour l'optimisation de la Qualité de Service dans les systèmes autonomes en énergie*. PhD thesis, Université de Nantes, 2014. 52

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Rola EL OSTA

Contributions à l'Ordonnancement en Temps Réel pour les Systèmes Autonomes en Energie.

Contributions to Real Time Scheduling for Energy Autonomous Systems.

Résumé

La récupération de l'énergie ambiante en temps réel est une technique qui permet d'allonger significativement la durée de vie des systèmes embarqués, aujourd'hui limitée par la quantité d'énergie stockable dans les batteries traditionnelles. La récupération d'énergie renouvelable (energy harvesting) comme celle envisagée pour de nombreux objets sans fil, rend possible un fonctionnement quasi-perpétuel de ces systèmes, sans intervention humaine, car sans recharge périodique de batterie ou de pile. Concevoir ce type de système autonome d'un point de vue énergétique devient très complexe lorsque celui-ci a en plus un comportement contraint par le temps et en particulier doit respecter des échéances de fin d'exécution au plus tard. Comme pour tout système temps réel, une problématique incontournable est de trouver un mécanisme d'ordonnancement dynamique capable de prendre en compte conjointement deux contraintes clés : le temps et l'énergie. Proposer et évaluer de nouvelles techniques d'ordonnancement pour que le système adopte un comportement énergétiquement neutre dans le respect des contraintes temps réel constitue le point central cette thèse.

Plus précisément, nous considérons ici un ensemble de tâches mixtes constitué de tâches périodiques et de tâches apériodiques souples sans échéance. L'architecture matérielle retenue est monoprocesseur. Les tâches apériodiques ne sont connues qu'au moment de leur arrivée et les tâches périodiques sont supposées ordonnançables par l'ordonnanceur optimal ED-H. La question à laquelle nous voulons apporter une réponse se résume comme suit : comment servir les tâches apériodiques pour minimiser leur temps de réponse sans remettre en question la faisabilité des tâches périodiques. Dans cette thèse, nous répondons à cette question de façon incrémentale. Dans un premier temps, nous étendons le serveur classique dit en arrière plan au contexte du energy harvesting avec la proposition de deux nouveaux serveurs. Simples à implémenter, ces techniques offrent toutefois des performances limitées. C'est pourquoi, dans un second temps, nous proposons un nouveau serveur basé sur le vol de temps creux (en anglais, Slack Stealing), au sens des notions de laxité temporelle et de laxité énergétique. Une évaluation théorique de celui-ci nous permet d'établir son optimalité. Vu l'implémentation relativement complexe de ce serveur, dans un dernier temps, nous proposons un nouveau serveur dit à préservation de bande (en anglais, Total Bandwith), basé sur l'attribution d'échéances fictives avec une implémentation plus simple. Une étude expérimentale accompagne nos propositions et permet d'attester la performance de nouveaux serveurs de tâches apériodiques spécifiquement conçus pour les systèmes temps réel autonomes.

Mots clés

Earliest Deadline First, récupération d'énergie, service apériodique, ordonnancement préemptif.

Abstract

Real-time energy harvesting is a technology that significantly extends the lifetime of embedded systems. This technology is limited at present by the amount of energy that can be stored in traditional batteries. Renewable energy harvesting such as that envisaged for many wireless things, allows the quasi-perpetual systems operation without human intervention because it works without periodic recharging of battery. From an energy point of view, the design of this type of autonomous system becomes more complex since this process has in addition a behavior constrained by time, and particularly has to meet latest timing deadlines. As with any real-time system, an unavoidable problem is to find a dynamic scheduling mechanism able of considering jointly two key constraints: time and energy. Thus, the main objective of this thesis is to propose and evaluate new scheduling techniques that enable the system to adopt an energy-neutral behavior while respecting the real-time constraints. More precisely, we consider here a set of mixed tasks consisting of periodic tasks and soft aperiodic tasks without deadline. The hardware architecture chosen is monoprocessor.

Aperiodic tasks are only known at the time of their arrival while periodic tasks are assumed to be schedulable by the optimal ED-H scheduler. In this thesis, we will provide appropriate solutions for the following question: how to serve aperiodic tasks in order to minimize their response time without challenging the feasibility of periodic tasks.

Initially, we extend the conventional server (called Background) to the context of energy harvesting by the proposal of two new servers. These techniques can be easily implemented and offer limited performance. Secondly, we propose a new server based on Slack Stealing which uses the slack time and slack energy concepts. A theoretical evaluation of this one allows us to establish its optimality. Finally, due to the relatively complex implementation of this server, we propose a new server, called Total Bandwidth. This server is based on fictive deadlines assignment with a simpler implementation. All propositions are illustrated by experimental studies that allow us to investigate the performance of new aperiodic task servers specifically designed for autonomous real-time systems.

Key Words

Earliest Deadline First, energy harvesting, aperiodic servicing, preemptive scheduling.