Schedulability Analysis for Fixed Priority Real-Time Systems with Energy-Harvesting

Yasmina Abdeddaïm, Younès Chandarli, Robert I. Davis, Damien Masson

Schedulability Analysis for Fixed Priority Real-Time Systems with Energy-Harvesting

Yasmina Abdeddaïm, Younès Chandarli,
Robert I. Davis and Damien Masson

June 30, 2014

Abstract

This paper introduces sufficient schedulability tests for fixed-priority pre-emptive scheduling of a real-time system under energy constraints. In this problem, energy is harvested from the ambient environment and used to replenish a storage unit or battery. The set of real-time tasks is decomposed into two different types of task depending on whether their rate of energy consumption is (i) more than or (ii) no more than the storage unit replenishment rate. We show that for this task model, where execution may only take place when there is sufficient energy available, the worst-case scenario does not necessarily correspond to the synchronous release of all tasks. We derive sufficient schedulability tests based on the computation of worst-case response time upper and lower bounds. Further, we show that Deadline Monotonic priority assignment is optimal with respect to the derived tests. We examine both the effectiveness and the tightness of the bounds, via an empirical investigation.

1 Introduction

In a context where traditional energy resources are continually decreasing, new and challenging problems arise that need to be tackled by researchers in different fields. Examples include, how to use new energy resources in an optimal way, and how to integrate smart energy management into newly developed electronic applications. Collecting energy from the ambient environment, so called *energy harvesting*, is a solution that has significant benefits, particularly when the powered device is inaccessible or has limited accessibility making the renewal of a traditional energy source either impossible, potentially dangerous, or costly.

In an energy harvesting process, energy is drawn from the environment and then converted, using a harvester, into usable electrical power and stored in the battery. Sources of energy include ambient vibrations (piezoelectric effect), thermal gradients (thermoelectric and pyroelectric effects), Radio Frequency radiation (rectifying antenna), movement (via magnetic induction), solar radiation (photovoltaics), and even blood sugar (via oxidation by enzymes powering an implanted device). Compared to classical forms of energy storage, the environment can provide a continuous and essentially unbounded supply of energy, allowing the energy consumption of the system to be adjusted to maximize performance instead of minimizing overall energy consumption.

In this paper we consider the problem of real-time scheduling for systems using energy-harvesting. The challenge here is to schedule real-time tasks with hard deadlines while making the best use of available energy. Compared to classical real-time scheduling models, we do not neglect the fact that tasks consume energy during their execution. In this paper, we make the simplifying assumption that each task may use energy up to a maximum rate of power dissipation (i.e. energy per unit of execution), but that rate may be different for different tasks.

In real-time systems utilising energy harvesting, the energy needed for task execution is supplied by the storage unit (i.e. a battery or capacitor) which has a fixed capacity. The energy in the storage unit is replenished continuously by the electrical energy produced by the harvester. In general, the energy provided by the harvester in a given time interval can be described by the integral of the replenishment rate over that time. In this paper we assume that the replenishment rate is a constant, or at least lower bounded by a constant. This assumption simplifies the problem; nevertheless, it corresponds to some existing harvesting technologies (see Section 3).

As some tasks may consume energy faster than the replenishment rate (so called *consuming tasks*), the energy in the storage unit may diminish until it is no longer sufficient to support execution. At such times processing must be suspended until sufficient energy has been replenished for execution to continue. Classical real-time scheduling algorithms need to be adapted to cater for this behaviour.

In this work we use PFP_{ASAP}, an energy-aware adaptation of fixed priority preemptive scheduling (FPPS).This algorithm is similar to FPPS in that at any given time it selects the job of the highest priority active task for execution; however, unlike FPPS, PFP_{ASAP} only executes the next execution time unit of that job if there is sufficient energy available to do so. The algorithm PFP_{ASAP} is optimal with respect to all fixed priority algorithms for non-concrete periodic task sets, compliant with a model where all tasks are consuming tasks [2]. In this paper, we consider real-time task sets comprising two types of tasks: (i) *consuming tasks* that have a rate of energy consumption that is higher than the replenishment rate, and (ii) *gaining tasks* that have a rate of energy consumption that is no more than the replenishment rate. We show that for this more general model, the critical instant leading to the worst-case response time of a task does not necessarily correspond to a synchronous release with all higher priority tasks, and so the analysis given in [2] is not applicable. For the more general model, we derive two response time upper bounds providing sufficient scheduling tests. We also prove that Deadline Monotonic priority assignment [14] is an optimal priority assignment policy with respect to these sufficient schedulability tests.

The remainder of the paper is organized as follows. In Section 2, we review related work on real-time scheduling for systems using energy-harvesting. In Section 3 we present the system model, terminology and notation used in the rest of the paper. In Section 4 we briefly recapitulate on classical response time analysis, and show how this was extended to analysis of the PFP_{ASAP} algorithm for energy-constrained systems with only consuming tasks. In Section 5 we introduce sufficient schedulability analysis for the more general task model with both consuming and gaining tasks. Section 6 provides a performance evaluation investigating the effectiveness and tightness of these schedulability tests. Section 7 concludes with a summary and discussion of future work.

2 Related Work

The first work addressing the real-time scheduling problem for systems using energy harvesting was presented by Mossé in 2001 [4]. The proposed algorithm was for a frame-based model where all of the tasks have exactly the same period and the same deadline. In 2006, Moser et al. [17] proposed an optimal algorithm called LSA (Lazy Scheduling Algorithm). Unlike our model, the results of this work rely on the assumption that a task's energy consumption is directly linked to its worst-case execution time. In 2011, Chetto et al. [11] proposed an algorithm called $EDeg$. With $EDeg$, Earliest Deadline First scheduling is used as long as the system can perform; however, execution is suspended when a future energy failure is detected. To detect a future energy failure the notion of slack time [13, 10, 8] was extended to slack energy; however, the computation of the slack energy can lead to a huge overhead. Finally, the PFP_{ASAP} algorithm for fixed priority pre-emptive scheduling was proposed by Abdeddaïm et al. in 2013 [2], in this work all of the tasks were assumed to consume energy faster than it is replenished.

3 Models and Notations

An energy harvesting system is composed in most applications of two main parts: the *harvester* that converts energy from the ambient environment into electrical power and the *storage unit* used to store the electrical energy produced. The choice of the harvested *energy source*, the *harvester* and the *storage unit* must be considered according to the target application characteristics. Concerning the energy source, wind, ocean waves or solar energy can provide a large amount of energy but are characterised by significant variability in the energy produced [18]. On the other hand, mechanical energy sources such as machine vibrations provide a small amount of energy but more consistent and continuous replenishment [3].

To manage the possible variations over time, power management circuits can be used in the harvester to adapt the inputs and outputs of the harvester to meet the desired power rate [3], however, the harvester consumption should stay less than the energy gained from the environment. When the power consumption of the target device is greater than the energy provided from the environment, the harvested energy must be stored in a storage unit to be used at an appropriate time. The storage unit can be a capacitor or a battery, this choice depends on the desired properties, such as the performance at different temperatures, the dissipation of energy, the capacity required and the weight.

In this paper, we consider a hard real-time system equipped with an energy harvesting system. The system comprises a single processor which executes a set of tasks according to the energy-aware fixed priority pre-emptive scheduling algorithm PFP_{ASAP}. In the following subsections, we present our model and give more details about the applications for which our model is the most appropriate.

3.1 Energy Source Model

In this paper we suppose that the quantity of energy that arrives in the storage unit is a function of time which is either known or bounded. The replenishment of the storage unit is performed continuously even during the execution of tasks. $P_r(t)$ is the replenishment function of the battery, then, the energy replenished during any time interval $[t_1, t_2]$

denoted as $g(t_1, t_2)$ is given by (1).

$$g(t_1, t_2) = \int_{t_1}^{t_2} P_r(t)dt \tag{1}$$

As mentioned above, there are many exploitable sources of environmental energy. However, the generated current and voltage differ from one source to an other. Furthermore, the yielded energy is not stable over time in all sources. For example, the energy generated with a solar cell depends on the intensity of light which is highly variable because of the day/night cycles and the weather variations. In this paper we target small embedded systems that do not consume a lot of energy but require a stable source (e.g. Wireless Sensor Networks). According to [3, 18] the most appropriate source that fulfills these requirements is piezoelectric vibration energy. The energy generated with this technique depends on the vibration frequency, and even thought the vibration frequencies can vary over time, because of engine speed changes for example, the generated energy can still be stable thanks to a new generation of piezoelectric vibration energy harvesters that are able to yield an optimal output of energy even with 40% vibration frequency variation [3]. Knowing that in most industrial machines the variation of vibrations is significantly below than 40%, we can consider that in the worst case the storage unit is replenished at a lower bound constant rate.

Thus, in the following we assume $P_r(t)$ to be a constant function, i.e. $P_r(t) = P_r$. Then, the energy replenished during any time interval $[t_1, t_2]$ is given by (2).

$$g(t_1, t_2) = (t_2 - t_1) \times P_r \tag{2}$$

3.2 Energy Storage Unit Model

Nowadays, there are many types of energy storage devices available on the market, from chemical batteries (e.g., Alkaline, Ni-Cd, Ni-MH, Li-ion, etc) to supercapacitor (e.g., Double-layer capacitors, Pseudocapacitors, Hybrid capacitors,etc). If we consider as a targeted application small embedded systems that operate with a small amount of energy and a constant rate of charging, the appropriate storage unit is a supercapacitor because, firstly, it can be replenished linearly which allows the system to fully use the incoming energy from the harvester and secondly, it supports a high number of charge/discharge cycles.

In the following we abuse the term *battery* to indicate the storage unit or the supercapacitor. We consider that the energy stored in the battery may vary between two levels E_{min} and E_{max}, where E_{max} is the maximum capacity of the battery, and E_{min} is the minimum energy level needed to keep the system running. For the sake of clarity, and without loss of generality we assume that $E_{min} = 0$. The battery level at time t is denoted by $E(t)$. We note that as supercapacitors self-discharge due to leakage current, we make the safe assumption that only minimal energy is available when the system is deployed or activated, i.e. $E(0) = E_{min}$.

The energy level in the battery is not permitted to fall below E_{min}; in contrast, if it reaches E_{max}, then any further replenishment above that level is effectively wasted, since the maximum amount of energy that can be stored is capped. In this work, we assume E_{max} to be large enough to warrant that the schedule produced by PFP_{ASAP} is not impacted by this phenomenon. Since it is important in embedded applications to minimize the battery capacity, due to cost and weight concerns, we discuss in Section 5.6 the minimum capacity required given our analysis.

3.3 Task Model

The task set comprises a static set of n sporadic and independent tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by its unique priority i. Without loss of generality, we assume that the tasks are in priority order, thus task τ_1 has the highest priority and task τ_n the lowest. We use the notation $hep(i)$ to mean the set of tasks with priorities higher than or equal to i. Each task τ_i has a worst-case execution time C_i, a minimal inter-arrival time or period T_i, a relative constrained deadline D_i ($D_i \leq T_i$), and a worst-case energy consumption E_i. The worst-case power consumption (i.e. energy used per unit of execution time) of a task τ_i is given by P_i. Thus the worst-case energy consumption equates to executing for the worst-case execution time, at the maximum rate of power consumption (i.e. $E_i = P_i \times C_i$).

The execution time and the energy consumption of tasks are assumed to be independent. For example considering two tasks τ_i and τ_j, we may have $C_i < C_j$ and $E_i > E_j$. The energy dissipation of the system can change according to the energy consumption of the tasks. The set of tasks Γ is seperated into two distinct subsets Γ_c and Γ_g. The first one Γ_c contains the *consuming tasks*, the ones that consume more energy than is replenished during their execution, whereas Γ_g is composed by the *gaining tasks*, that consume no more energy than is replenished during their execution. We have $\Gamma_c = \{\tau_i \in \Gamma, E_i > P_r \times C_i\}$ and $\Gamma_g = \{\tau_i \in \Gamma, 0 \leq E_i \leq P_r \times C_i\}$.

We define the processor utilization of task τ_i as $u_i^p = C_i/T_i$ and its energy utilization as $u_i^e = E_i/(T_i \times P_r)$. The total utilization of the task set is the sum of the utilizations of all its tasks, i.e. $U^p = \sum_{i=1}^{n} u_i^p$. Similarly the energy utilization of the task set is given by $U^e = \sum_{i=1}^{n} u_i^e$.

3.4 The Scheduling Algorithm $\mathbf{PFP_{ASAP}}$

The behaviour of the PFP_{ASAP} scheduling algorithm is formally defined as follows: at any given time instant t, the job of the highest priority active[1] task τ_j is selected for execution; however, that job is only executed during the interval $[t, t+1)$ if there is sufficient energy available for that unit of execution i.e. $E(t) + P_r \geq E_j/C_j$. Note we assume that all task and system parameters are in discrete time and discrete energy units.

The worst-case response time R_i of task τ_i under PFP_{ASAP} scheduling is defined as the longest possible time from the release of a job of that task until the job completes execution, (i.e. for any valid sequence of job releases generated by the task set and any valid initial battery level). Task τ_i is thus schedulable under PFP_{ASAP} if and only if $R_i \leq D_i$. The task set is schedulable if all of its tasks are schedulable.

4 Existing Response time analysis

In this section, we first recapitulate the classical response time analysis for fixed priority pre-emptive scheduling (FPPS) [12, 5] and then show how this analysis can be extended to systems with energy harvesting, but restricted to either all gaining tasks, or all consuming tasks [2].

Let $\{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of real-time tasks (as defined in Section 3) where the tasks do not consume energy i.e. $\forall i\ E_i = 0$. Response time analysis for such systems makes use of the concept of a priority level$-i$ busy period. This is defined as a contiguous interval of

[1] An active task is one that has a job that has been released but not yet completed.

time during which the processor is busy executing jobs of priority level$-i$ or higher, that were released during the interval. In the case of FPPS of tasks with constrained deadlines, the worst-case response time R_i of task τ_i corresponds to the length of the longest priority level$-i$ busy period.

To calculate the longest priority level$-i$ busy period, it suffices to consider only the worst-case scenario or *critical instant* [15] for which a job of task τ_i is subject to the maximum possible delay. This occurs when task τ_i is released simultaneously with all tasks of higher priority, which are then re-released as soon as possible. In this case, the worst-case response time R_i of a task τ_i is given by the smallest $t > 0$ that satisfies:

$$t = F(i, t)$$

where

$$F(i,t) = \sum_{h \in hep(i)} \left\lceil \frac{t}{T_h} \right\rceil \times C_h \tag{3}$$

We note that (3) may be solved using fixed point iteration starting with $t = C_i$ and ending on convergence or when $t > D_i$ in which case the task is unschedulable. (Convergence may be speeded up using the techniques described in [9]).

We note that FPPS is a *work-conserving* scheduling policy, since it never leaves the processor idle when there are any active tasks. In an energy harvesting context, the notion of a work-conserving policy can be usefully refined. We refer to a fixed priority scheduling policy as *energy work-conserving* if while there are any active tasks requiring execution, the scheduling policy only ever leaves the processor idle if there is insufficient energy available to schedule at least one time unit of the highest priority active task.

The energy work-conserving scheduling algorithm PFP_{ASAP} is optimal among all fixed priority scheduling algorithms for the case where all tasks consume energy ($\Gamma_g = \emptyset$) [2]. In this case, the critical instant for task τ_i corresponds to synchronous release with all tasks of higher priority at a time when the battery level is at its minimum. This characterisation of the critical instant greatly simplifies the schedulability analysis problem, allowing task response times to be obtained via fixed point iteration in a similar way to the classical response time analysis for FPPS, but replacing Equation (3) by

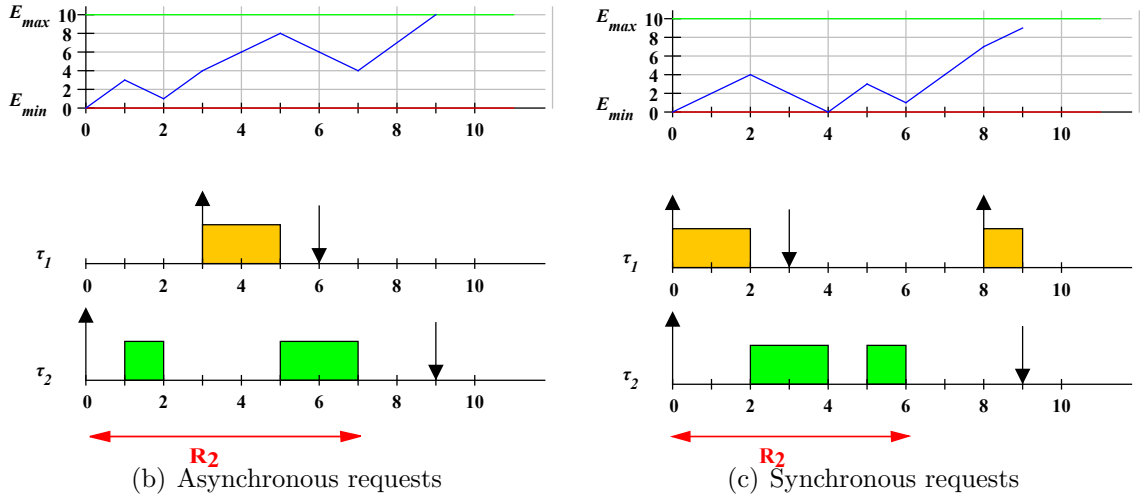
$$F(i,t) = \left\lceil \left(\sum_{h \in hep(i)} \left\lceil \frac{t}{T_h} \right\rceil \times E_h \right) / P_r \right\rceil \tag{4}$$

5 Schedulability Analysis

In this section, we provide sufficient schedulability tests for systems with both consuming and gaining tasks. First we show that the critical instant for such task sets does not necessarily correspond to synchronous release. Lack of information about the actual worst-case scenario makes the schedulability analysis problem much more difficult. We address this problem by introducing the concept of a priority level-i energy busy period (defined in Section 5.2). The worst-case response time of task τ_i must necessarily occur within such a busy period. We derive two upper bounds on the maximum length of this busy period, which we then use to obtain upper bounds on the worst-case response time of the task. We use a similar approach to also derive response time lower bounds.

Tasks	C_i	E_i	T_i	D_i
τ_1	2	2	8	3
τ_2	3	15	10	9

(a) Task set



(b) Asynchronous requests (c) Synchronous requests

Figure 1: Worst-case scenario counter example

5.1 Worst-case scenario

When we consider only consuming tasks or only gaining tasks, the worst-case scenario (critical instant) occurs when all higher priority tasks are released simultaneously and the battery is at its minimum level. For the case when we have only gaining tasks, the response time analysis is the same as the classical formulation given by (3), since there are no delays due to energy considerations.

For the case where we have only consuming tasks, launching the tasks simultaneously with the battery at its minimum level maximizes the idle periods needed for energy replenishment. This increases the time required to complete the execution of higher priority tasks, which leads to the longest response time for each task, given by (4) as proved in [2].

In contrast, when we consider a task set composed of both gaining and consuming tasks the worst-case scenario is not the same for all the tasks, it depends on the composition of the subset of higher priority tasks. If that subset contains both gaining and consuming tasks, then the worst case scenario is not necessarily the synchronous activation of all the tasks with the minimum battery level.

Figure 1 illustrates a situation where the response time of task τ_2 is longer ($R_2 = 7$) when a gaining task of higher priority is requested later, than it is with synchronous release ($R_2 = 6$). This is due to the fact that in the former case, task τ_2 suffers two replenishment delays (at time $t = 0$ and $t = 2$), whereas in the later case it suffers only one replenishment delay (at time $t = 4$). This happens because task τ_1 is a gaining task and there is a net increase in energy as it executes.

5.2 Sequences and Energy Busy Periods

We now introduce terminology and concepts that we use in proving key properties about scheduling systems with energy constraints. We use the term *execution unit* to refer to a non-divisible unit of execution of a job. An execution-unit has the same length as the basic time unit used to describe task execution times, and is of the same length for all tasks. We use the term *replenishment unit* to refer to the minimum indivisible unit of idling time used to replenish energy. Execution-units and replenishment-units are of the same duration.

An execution *sequence* is a vector X of execution units from 1 to L_X, where L_X is the number of execution units in the sequence. Each element $X[m]$ of the sequence indicates the task that the execution unit belongs to. A sequence does not contain replenishment units, and so L_X does not necessarily represent the number of time units needed to execute the sequence. We denote the energy required by execution unit $X[m]$ by $E_X[m]$. Further we use $E^*_X[m]$ to denote the total energy required by execution units from the start of the sequence up to and including execution unit $X[m]$. Thus:

$$E^*_X[m] = \sum_{q=1\ldots m} E_X[q] \tag{5}$$

The minimum number of replenishment units $I_X[m]$ required to provide sufficient energy to execute $X[m]$ at the end of the subsequence $X[1]$ to $X[m]$ is given by:

$$I_X[m] = \max\left(0, \left\lceil \frac{E^*_X[m] - E(0)}{P_r} \right\rceil - m\right) \tag{6}$$

where $E(0)$ is the energy available at the start of the sequence.

We note that an earlier execution unit $X[k]$ may require more prior replenishment units than a later one $X[m]$ due to the presence of execution units of gaining tasks between $X[k]$ and $X[m]$, (i.e. $I_X[k] > I_X[m]$ where $m > k$). We use $I^*_X[m]$ to denote the minimum number of replenishment units required to execute all of the subsequence $X[1]$ to $X[m]$ in order.

$$I^*_X[m] = \max_{k=1\ldots m} (I_X[k]) \tag{7}$$

The elapsed time required to execute sequence X is given by $L_X + I^*_X[L_X]$.

Lemma 1. *For a fixed sequence X of execution units, the elapsed time for the sequence is maximised when the initial energy available is minimised, i.e. $E(0) = 0$.*

Proof. Follows directly from (6) and the formula for the elapsed time to execute the sequence: $L_X + I^*_X[L_X]$. □

Lemma 2. *Any sequence containing only execution units of consuming tasks requires the same elapsed time to execute irrespective of the order of its execution units provided that the set of execution units and the initial energy are the same. Similarly, any sequence containing only execution units of gaining tasks requires the same elapsed time to execute irrespective of the order of its execution units provided that the set of execution units is the same.*

Proof. Case (i) sequence X contains solely execution units of consuming tasks. Since all execution units consume energy, then for every element $X[m]$, we have $E_X[m] > P_r$ and

so $E^*_X[m+1] > E^*_X[m] + P_r$ hence the maximum number of prior replenishment units is required by the last element in the sequence and is given by:

$$I^*_X[L_X] = I_X[L_X] = \left\lceil \frac{E^*_X[L_X] - E(0)}{P_r} \right\rceil \tag{8}$$

Since the total energy $E^*_X[L_X]$ required by all elements in the sequence is independent of the order of the elements, the elapsed time $I^*_X[L_X]+L_X$ required to execute the sequence is also independent of the order of the elements.

Case (ii) sequence X contains solely execution units of gaining tasks. Since all execution units gain energy, no replenishment units are required and the elapsed time for the sequence equates to its length L_X irrespective of the order of the elements.

□

Definition 1. *A* priority level$-i$ energy busy-period *is defined as a contiguous interval of time* $[0, w)$ *during which the processor is busy executing jobs of priority level$-i$ or higher, that were released during the interval, but strictly before its end at time* w*, or there is an active job of priority* i *or higher and the processor is necessarily idling to replenish sufficient energy to execute the next execution unit of the highest priority active job.*

We note that any execution of a job of task τ_i must by definition of a priority level$-i$ energy busy-period occur within such a busy period.

Lemma 3. *Under PFP_{ASAP} scheduling, for a schedulable task τ_i, the worst-case response time R_i of the task equates to the longest possible priority level$-i$ energy busy-period. Further, there exists a busy period of this length that includes a single job of task τ_i, begins at the release of this job and ends with the final execution unit of the job.*

Proof. As task τ_i has the lowest priority of any task executing in such a priority level$-i$ energy busy-period, under PFP_{ASAP} scheduling the busy period necessarily ends with the final execution unit of that task. This is the case because if there were any outstanding higher priority tasks, they would execute in preference to task τ_i.

As task τ_i has a constrained deadline and is schedulable (by the Lemma), it can only have one job starting from the release time in the busy-period, otherwise the completion of the previous job of task τ_i would have to take place after the release of the final job of the task implying (as $D_i \leq T_i$) that the previous job was unschedulable.

Let X be the sequence of execution units representing all execution in the busy period. If the job of task τ_i was not released at the start of the busy period, then we can move its release time back to the start of the busy period. Since task τ_i has the lowest priority of any task in the busy period, such a change cannot make any difference to the actual order of execution as represented by sequence X and so has no impact on the elapsed time required to execute the sequence. Such a change can therefore only increase the worst-case response time of the job. □

Lemma 3 proves that the worst-case response time for a task τ_i occurs in a priority level$-i$ energy busy-period starting with the release of that task. However, as shown in Figure 1, synchronous release of all higher priority tasks may not result in the worst-case response time for task τ_i. In general, we do not know what scenario, or pattern of releases of higher priority tasks will result in the worst-case response time for task τ_i; however, we can derive further information about possible worst-case scenarios.

Lemma 4. *The maximum possible number of jobs of a higher priority task τ_h causing interference in the longest priority level−i busy period (characterising the worst-case response time of task τ_i) is given by $\lceil w/T_h \rceil$ where w is the length of the busy period.*

Proof. Lemma 3 shows that the busy period starts (at time $t = 0$) with the release of task τ_i, hence at $t = 0$, there can be no jobs of higher priority tasks with outstanding execution, other than those also released at $t = 0$, otherwise the busy period would have started earlier. It follows that the maximum number of higher priority jobs of task τ_h in the busy period is given by $\lceil w/T_h \rceil$. □

5.3 Response Time Upper Bounds

Since, we do not know the precise pattern of releases of higher priority jobs that leads to the worst-case response time for task τ_i, we cannot determine the exact worst-case response time. Instead, we derive an upper bound R_i^{UB1} and then a tighter upper bound R_i^{UB2} on the exact worst-case response time R_i, where $R_i^{UB1} \geq R_i^{UB2} \geq R_i$. These upper bounds provide sufficient schedulability tests $UB1$ and $UB2$ respectively, where $UB2$ dominates $UB1$.

The process we use to obtain these upper bounds is similar to the classic formulation of response time analysis presented in section 4. We aim to find the smallest interval w, for which an upper bound on the response time of task τ_i, considering the maximum possible interference from higher priority tasks released in that interval, equates to the length of the interval. The value of w then provides an upper bound on the worst-case response time of task τ_i.

We require a function $F(i, w)$ that upper bounds the length of the longest priority level−i energy busy-period formed by a single job of task τ_i and jobs of higher priority tasks released during an interval of length w. Provided that $F(i, w)$ is a monotonically non-decreasing function of w, then we may obtain an upper bound on the worst-case response time of task τ_i corresponding to the smallest value of $w > 0$ that satisfies:

$$w = F(i, w) \tag{9}$$

Equation (9) may be solved using fixed point iteration starting with $w = C_i$ and ending on convergence or when $w > D_i$ in which case the task is deemed unschedulable.

5.4 Upper Bound R^{UB1}

We now derive a simple upper bound R_i^{UB1} on the worst-case response time of task τ_i. First we prove a Lemma used in its derivation.

Lemma 5. *Let X be some arbitrary sequence of execution units of tasks of priority i or higher, and Y be the equivalent sequence re-ordered such that all execution units of consuming tasks come before all execution units of gaining tasks. The elapsed time required to complete sequence Y is no shorter than that required to complete sequence X.*

Proof. We may obtain sequence Y from sequence X by an iterative process of choosing the first execution unit belonging to any gaining task (at position g) and swapping it with that of the last execution unit of any consuming task (at position k) provided that $g < k$. Repeating this process until all consuming execution units come before all gaining execution units transforms sequence X into sequence Y. Let the new sequences produced

by this process be $X_1 = X, X_2, X_3 \ldots X_n = Y$. Note at most $L_X/2$ swaps are required. We now show that each swap transforming sequence X_p into sequence X_s where $s = p+1$, results in an elapsed time for sequence X_s that is no shorter than that for X_p, and hence by induction that the elapsed time for sequence Y is no shorter than that for sequence X. Let $X_p[g]$ and $X_p[k]$ be the elements being swapped where $g < k$. Since $X_p[g]$ is an execution unit of a gaining task and $X_p[k]$ is an execution unit of a consuming task, the energy required for these execution units has the relationship $E_{X_p}[g] < E_{X_p}[k]$. Recall that $E^*_{X_p}[m]$ is the energy required to execute all execution units in the subsequence from $X_p[1]$ to $X_p[m]$. It follows that:

$\forall m, 1 \leq m < g \; E^*_{X_s}[m] = E^*_{X_p}[m]$
$\forall m, g \leq m < k \; E^*_{X_s}[m] = E^*_{X_p}[m] + E_{X_s}[k] - E_{X_p}[g]$
$\forall m, k \leq m \; E^*_{X_s}[m] = E^*_{X_p}[m]$
$\Rightarrow \; E^*_{X_s}[m] \geq E^*_{X_p}[m]$

Hence the minimum number of replenishment units required to execute the subsequences from the 1st to the $m-$th element of X_p and X_s have the following relationship:$I^*_{X_s}[m] \geq I^*_{X_p}[m]$ (see (6) and (7)). Since the number of execution units in each sequence (X_s and X_p) is the same (i.e. $L_{X_p} = L_{X_s}$), we have: $L_{X_s} + I^*_{X_s}[m] \geq L_{X_p} + I^*_{X_p}[m]$. Thus the elapsed time required to execute sequence X_s is no shorter than that required for sequence X_p. Induction over at most $L_X/2$ steps proves that the elapsed time required to complete sequence Y is no shorter than that required for sequence X.

□

Theorem 1. *An upper bound on the worst-case response time for task τ_i for a set of jobs released in a window of length w can be obtained by assuming that there is one job of task τ_i and $\lceil w/T_h \rceil$ jobs of each higher priority task τ_h. Further, the upper bound is obtained from a sequence Z of the execution units of these jobs where all the consuming execution units are before all the gaining execution units.*

Proof. Let X be the sequence of execution units that results in the longest priority level-i energy busy-period under PFP_{ASAP} scheduling, and hence the longest response time for task τ_i), for a set of jobs released in a window of length w. The elapsed time for the sequence is given by $L_X + I^*_X[L_X]$ where L_X is the length of the sequence and $I^*_X[L_X]$ is the total number of replenishment units required. Lemma 5 shows that the elapsed time required to execute a sequence Y is no shorter than that required to execute sequence X, where sequence Y comprises the execution units in X re-ordered such that all execution units of consuming tasks are placed before execution units of gaining tasks. Note that at this point we do not know how many jobs of higher priority tasks are present in sequence X and therefore also in sequence Y; however, by Lemma 4 we know that the maximum number of jobs of a higher priority task τ_h that could be present is $\lceil w/T_h \rceil$. Hence we add execution units to sequence Y as necessary to account for any shortfall in the number of jobs in X below this value, thus forming sequence Z. (Consuming execution units are added at the start of the sequence and gaining execution units at the end). We note that such additional execution units cannot reduce the elapsed time required to execute the sequence since all execution units of both consuming and gaining tasks require a positive amount of energy. Sequence Z (as described in the Theorem) therefore requires an elapsed time to execute that is no smaller than that of sequence X. □

We use Theorem 1 to formulate the workload function $F^{UB1}(i, w)$ for upper bound R^{UB1}_i. We assume that the initially available energy is zero, the number of jobs of task τ_i and each higher priority task released in an interval of length w is given by $\lceil w/T_h \rceil$,

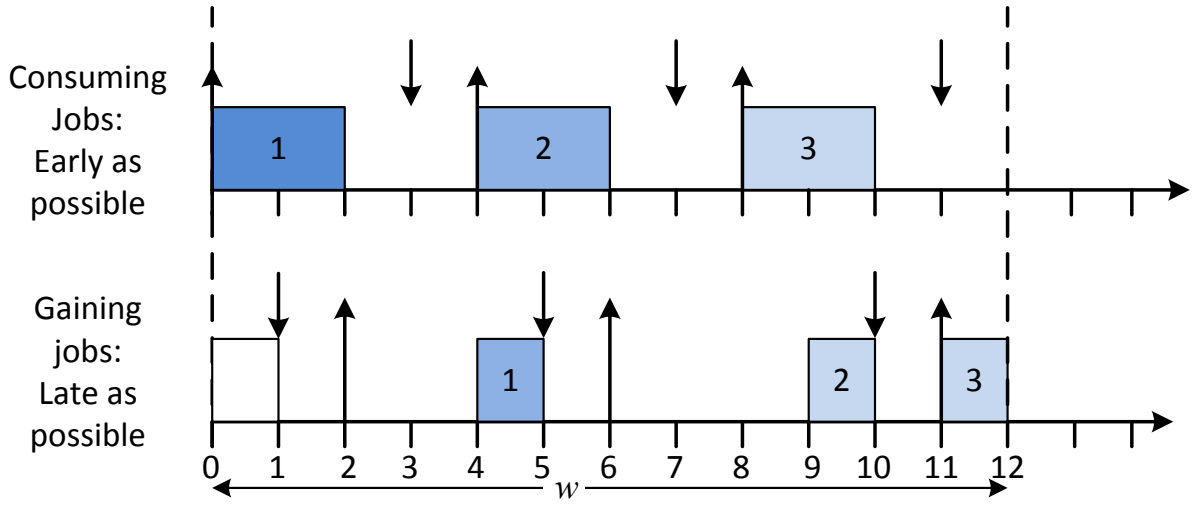


Figure 2: Dummy schedule used in the construction of $UB2$

and that all execution units of consuming jobs are executed before all execution units of gaining tasks. We note that the number of jobs considered equates to synchronous release of all the tasks, with re-release as soon as possible. This is equivalent to the critical instant for classical tasks without energy considerations. Although this is not necessarily the worst-case scenario for tasks that require energy (see Figure 1 for a counter example), it is the worst-case scenario with respect to how our upper bounds are computed. The workload function for R_i^{UB1} is given by:

$$F^{UB1}(i,w)=\left\lceil \frac{\sum_{h\in hep(i)\cap\Gamma_c}\left\lceil \frac{w}{T_h}\right\rceil \times E_h}{P_r}\right\rceil + \sum_{h\in hep(i)\cap\Gamma_g}\left\lceil \frac{w}{T_h}\right\rceil \times C_h \tag{10}$$

where the first term represents the total time to complete the execution units of consuming tasks, which are in effect energy-bound, and the second term is the time taken to complete the execution units of gaining tasks, which are processing time bound.

Observe that $F^{UB1}(i,w)$ is a monotonically non-decreasing function of w since all terms are positive and w only appears in the numerator of ceiling functions. Further $F^{UB1}(i,w) \geq C_i$ since $\lceil C_i/T_i \rceil = 1$ and if task τ_i is a consuming task then $E_i/P_r \geq C_i$, hence C_i serves as a valid initial value for fixed point iteration.

We note that in the case where all tasks are gaining tasks and so energy is not a consideration, (10) reduces to the exact analysis for classical tasks given by (3). Further, in the case where all tasks are consuming tasks (10) reduces to the analysis for that case given by (4).

5.5 Upper Bound R^{UB2}

We can refine the first upper bound by considering a more realistic scenario. More precisely, the idea is to take into consideration the fact that some gaining jobs cannot be executed after some consuming ones, because of their respective deadlines and releases, which define sub-intervals in which they are forced to run when the system is schedulable.

This idea is illustrated in Figure 2, which shows three jobs of a consuming task and three jobs of a gaining task. We know that *provided the tasks are schedulable*, job 1 of the gaining task must run before job 3 of the consuming task. This information can be used to compute a tighter upper bound on the maximum time needed to complete all of the jobs in the interval.

We now derive our second upper bound R_i^{UB2}. The workload function $F^{UB2}(i, w)$ for R_i^{UB2} is derived from a dummy schedule and the sequence of execution units obtained from it. Construction of the dummy schedule is as illustrated in Figure 2. The dummy schedule is measured in time units and covers the interval $[0, w)$. It has a timeline for each task of priority i and higher, with one job of task τ_i and $\lceil w/T_h \rceil$ jobs of each higher priority task τ_h. Jobs of consuming tasks (including task τ_i if it is one) are placed in the dummy schedule starting with a release at $t = 0$, with subsequent releases as soon as possible. These jobs are assumed to execute immediately. For gaining tasks (including task τ_i if it is one) we first align the release of the last job at time $w - C_i$ this job is assumed to execute immediately. Previous jobs of the gaining task are then released as late as possible respecting the release time of the subsequent job, and assumed to execute as late as possible i.e. just prior to their deadlines. Thus jobs of gaining tasks are added from the end of the dummy schedule working backwards in time, and jobs of consuming tasks are added from the start of the dummy schedule working forwards in time. Note that there may be overlaps between the schedules where more than one task appears to execute at the same time. This is shown in Figure 2: intervals $[4, 5]$ and $[9, 10]$.

From the dummy schedule, we derive a sequence Z of execution units. This sequence is composed by starting at the beginning of the dummy schedule with an empty sequence and appending all gaining tasks with execution in that time unit onto the sequence, followed by all consuming tasks with execution in the same time unit. This process is then repeated for all subsequent time units until all execution units have been collected. Note ties between execution units of two or more gaining tasks or two or more consuming tasks may be broken arbitrarily; however, all execution units of gaining tasks associated with some time unit t are placed into the sequence ahead of all execution units of consuming tasks associated with the same time unit. All execution units associated with a later time unit e.g. $t + 1$ appear later in the sequence than those associated with an earlier time unit t (We note that clashes may safely be resolved by giving preference to gaining tasks, since those execution units must necessarily take place by that time otherwise a deadline will be missed. Execution units of consuming tasks could and would have been executed earlier in any real schedule that meets all deadlines).

Finally, the workload function $F^{UB2}(i, w)$ is computed giving the elapsed time required to execute sequence Z, assuming that the initial energy is at its minimum. This can be done via simulation, limited to at most a length of time D_i.

Theorem 2. *An upper bound on the worst-case response time for task τ_i for a set of jobs released in a window of length w, where no higher priority jobs miss their deadlines, can be obtained by assuming that there is one job of task τ_i and $\lceil w/T_h \rceil$ jobs of each higher priority task τ_h, with the upper bound equating to the maximum time required to execute a sequence Z of the execution units of these jobs constructed according to the rules and dummy schedule construction described previously.*

Proof. Let X be the sequence of execution units that results in the longest priority level$-i$ energy busy-period under PFP_{ASAP} scheduling (and hence response time for task τ_i) for a set of jobs released in a window of length w where all higher priority tasks meet their

deadlines. For each task τ_h, let N_h be the number of jobs in sequence X. Consider a sequence Y formed by constructing a dummy schedule of length w including N_h jobs of each task τ_h and one job of task τ_i and applying the rules stated above for ordering execution units (Recall from Lemma 3 that there is only one job of task τ_i in the busy period, and hence in sequence X). Sequence Y and sequence X contain an identical set of execution units. Since no deadlines are missed when sequence X is executed, and the dummy schedule used to construct sequence Y places execution units of gaining jobs as late as possible without missing a deadline, in relation to the execution units of consuming jobs which are placed as early as possible without invalidating minimum inter-arrival constraints. It follows that sequence Y can be obtained from sequence X by a process of swapping earlier gaining execution units for later consuming execution units (Note that re-ordering of sub-sequences consisting of solely gaining execution units or solely consuming execution units may also be needed to obtain precisely the same sequence; however, Lemma 2 shows that this re-ordering among execution units of the same type has no effect on the elapsed time required to execute the complete sequence). Finally, we compare sequence Z obtained as described in the Theorem, and sequence Y. If sequence Y contains the maximum number of jobs $\lceil w/T_h \rceil$ of each task that may be released in a window of length w, then it is identical to sequence Z. Otherwise, sequence Z may be obtained from sequence Y by adding execution units for any missing jobs where $\lceil w/T_h \rceil > N_h$. Since all execution units require energy, addition of execution units into the sequence at any point cannot decrease the elapsed time required to execute the sequence. Hence the elapsed time required to execute sequence Z is no shorter than that required to execute sequence X. □

Theorem 2 shows that $F^{UB2}(i, w)$ provides a valid upper bound on the worst-case response time for task τ_i considering all jobs released in a window of length w. In order to use $F^{UB2}(i, w)$ in a fixed point iteration to determine an upper bound on the worst-case response time of task τ_i we must also show that $F^{UB2}(i, w)$ is a monotonic non-decreasing function of w, and that $F^{UB2}(i, w) > C_i$, so that we may use C_i as an initial value. The latter is trivially the case since a single job of task τ_i is always included in the workload and takes at least time C_i to execute.

Theorem 3. *$F^{UB2}(i, w)$ is a monotonically non-decreasing function of w.*

Proof. Consider increasing the length of the window from some arbitrary value w to $w+v$, comparing the dummy schedules used to derive $F^{UB2}(i, w)$ and $F^{UB2}(i, w + v)$ there are two effects: (i) all execution units of gaining jobs move to a later time e.g. $t + v$ rather than t, (ii) new execution units of gaining jobs may be added near the start of the schedule and new execution units of consuming jobs may be added near the end of the schedule. Consider sequence X formed in deriving $F^{UB2}(i, w)$ and sequence Y formed in deriving $F^{UB2}(i, w+v)$ but omitting all of the execution units of the new jobs from (ii). Sequences X and Y contain the same set of elements. Since all execution units of gaining jobs are v time units later in the dummy schedule used to construct sequence Y, it follows that sequence Y can be formed from sequence X by swapping later gaining execution units in X for earlier consuming execution elements, and as necessary re-ordering sub-sequences containing solely gaining or solely consuming execution units (Lemma 2). Hence the elapsed time required to execute sequence Y is no shorter than that required for sequence X. Consider a further sequence Z, if there were no additional jobs from (ii) then sequence Z is identical to sequence Y, otherwise it may be obtained from sequence Y by adding execution units for the missing jobs. Since all execution units require energy, addition of

execution units into a sequence at any point cannot decrease the elapsed time required to execute the sequence. Hence the elapsed time required to execute sequence Z is no shorter than that required to execute sequence X.

□

We now return to the assumption in Theorem 2 that all deadlines of higher priority tasks are met. This might seem to imply that task schedulability must be checked highest priority first; however, this is not necessarily the case. Consider what happens if we test task schedulability lowest priority first. We tentatively test the schedulability of task τ_i on the assumption that all higher priority tasks will later be found to be schedulable. If task τ_i is deemed schedulable (caveat this assumption), then we go on to check higher priority tasks. If some higher priority task τ_h is subsequently found to be unschedulable, then this undermines the validity of our schedulability test for task τ_i; however, this is now of no consequence, since the task set is in any case unschedulable due to task τ_h. If instead, all higher priority tasks are found to be schedulable, then the schedulability test for task τ_i is validated (We note that the schedulability or otherwise of a lower priority task τ_i has no impact on the schedulability of any higher priority task τ_h).

Theorem 4. *Schedulability test $UB2$ dominates test $UB1$ i.e $R_i^{UB1} \geq R_i^{UB2}$.*

Proof. We prove the theorem by showing that $F^{UB2}(i,w) \leq F^{UB1}(i,w)$. Consider the sequence Y representing $F^{UB1}(i,w)$ and the sequence X representing $F^{UB2}(i,w)$. The sequences contain the same elements; however, in sequence Y all of the consuming execution units are before all of the gaining execution units, hence by Lemma 5, the elapsed time required to complete sequence Y is no shorter than that required to complete sequence X. □

5.6 Battery Capacity

We now return to a consideration of the maximum battery capacity E_{max}. For the sufficient test $UB1$ to be valid, we require that $E_{max} \geq max_{\forall i}(E_i/C_i) - P_r$. This small battery capacity is sufficient, since in computing an upper bound on the worst-case response time, $UB1$ assumes that all consuming execution units come before all gaining execution units. The minimum battery capacity needed to execute this sequence without impinging on the elapsed time required is simply enough to execute the most costly unit of execution in terms of energy, which equates to $max_{\forall i}(P_i) - P_r$ or $max_{\forall i}(E_i/C_i) - P_r$. We note that this is implicitly multiplied by 1 time unit, so that both the left hand side (E_{max}) and the right hand side are in units of energy.

By comparison, for the sufficient schedulability test $UB2$ to be valid, it suffices to have a maximum battery capacity E_{max} that equates to at least the total net energy required to execute all of the consuming jobs in the longest possible priority level-n energy busy period. Such a store of energy upper bounds that which can ever usefully be deployed to execute consuming jobs in any possible busy period. Having a larger battery capacity than this is equivalent in terms of task response times to having infinite battery capacity (Note that by the total *net* energy required by consuming jobs, we mean $E^{tot} - (C^{tot} \times P_r)$, where E^{tot} is the total energy required by consuming jobs in the longest busy period, and similarly C^{tot} is the total processing time that they require).

5.7 Response Time Lower Bound

In this section, we derive an analytical lower bound $R_i^{LB1} \leq R_i$ on the worst-case response time of task τ_i. To obtain the lower bound, we analyse a specific scenario that corresponds to the synchronous release of task τ_i along with all higher priority tasks, which are then assumed to be re-released as soon as possible. Further, we assume that the initial energy is a minimum i.e. $E(0) = 0$. Although this is not necessarily the worst-case scenario, it is a valid scenario and hence suffices to provide a valid lower bound on the longest priority level$-i$ energy busy period and hence the worst-case response time of task τ_i.

We obtain the lower bound response time R_i^{LB1} via fixed point iteration, using a workload function $F^{LB1}(i, w)$ that is monotonically non-decreasing in w and lower bounds the elapsed time needed to execute all jobs of tasks of priority i or higher released in an interval of length w starting with a synchronous release.

Lemma 6. *Let X be some arbitrary sequence of execution units of tasks of priority i or higher, and Y be the equivalent sequence re-ordered such that all execution units of consuming tasks come after all execution units of gaining tasks. The elapsed time required to complete sequence X is no shorter than that required to complete sequence Y.*

Proof. Follows by applying similar reasoning to the proof of Lemma 5. □

Theorem 5. *A lower bound on the worst-case response time for task τ_i assuming synchronous release with all higher priority tasks resulting in a priority level$-i$ energy busy period of at least length w, can be obtained by assuming that there is one job of task τ_i and $\lceil w/T_h \rceil$ jobs of each higher priority task τ_h in the busy period. Further the lower bound equates to the time required to execute a sequence Z of the execution units of these jobs where all the consuming execution units are after all the gaining execution units, and the initial energy is a minimum.*

Proof. By the theorem, the busy period is at least w long, hence under PFP_{ASAP} scheduling all $\lceil w/T_h \rceil$ jobs of each higher priority task τ_h released during the interval $[0, w)$ must complete before the single job of task τ_i. Let X be the sequence of execution units of all of the jobs under PFP_{ASAP} scheduling. The elapsed time required to execute sequence X lower bounds the worst-case response time of task τ_i. Further, let Z be (as per the theorem) the same set of execution units as sequence X re-ordered such that all the consuming execution units are after all the gaining execution units. By Lemma 6, the elapsed time to execute sequence Z is no longer than that required to execute sequence X. □

We use Theorem 5 to formulate our lower bound workload function $F^{LB1}(i, w)$. We assume that the initially available energy is zero, the number of jobs of task τ_i and each higher priority task τ_h released in an interval of length w is given by $\lceil w/T_h \rceil$ and that all execution units of consuming jobs are executed after all execution units of gaining tasks.

$$X_i^g = \sum_{h \in hep(i)}^{\tau_h \in \Gamma_g} \left\lceil \frac{w}{T_h} \right\rceil \times C_h,\ X_i^c = \sum_{h \in hep(i)}^{\tau_h \in \Gamma_c} \left\lceil \frac{w}{T_h} \right\rceil \times C_h$$

$$Y_i^c = \sum_{h \in hep(i)}^{\tau_h \in \Gamma_c} \left\lceil \frac{w}{T_h} \right\rceil \times E_h\ ,\ Y_i^g = \sum_{h \in hep(i)}^{\tau_h \in \Gamma_g} \left\lceil \frac{w}{T_h} \right\rceil \times E_h$$

$$F^{LB1}(i, w) = X_i^g + \max\left(X_i^c, \left\lceil \frac{Y_i^c - (X_i^g \times P_r - Y_i^g)}{P_r} \right\rceil\right) \tag{11}$$

Finally, in order to use the workload function $F^{LB1}(i,w)$ in a fixed point iteration to determine the lower bound R_i^{LB1} on the worst-case response time of task τ_i, we must show that $F^{LB1}(i,w)$ is a monotonically non-decreasing function of w.

Theorem 6. *$F^{LB1}(i,w)$ is a monotonically non-decreasing function of w.*

Proof. Consider the formula for $F^{LB1}(i,w)$. Since w appears only in the ceiling functions, it follows that X_i^g, X_i^c and Y_i^g are all non-decreasing functions of w. Further, $(X_i^g P_r - Y_i^g)$ represents the net energy increase while all the gaining jobs execute. Since every execution unit of a gaining task is by definition energy positive, this quantity is also a non-decreasing function of w. Thus $Y_i^c - (X_i^g P_r - Y_i^g)$ may decrease with increasing w. The largest possible decrease is obtained when Y_i^c remains at the same value, while $(X_i^g P_r - Y_i^g)$ increases, hence $\lceil (X_i^g P_r - Y_i^g)/P_r \rceil$ decreases. However, such a decrease is always at least compensated for by the increasing value of the first term in (11), i.e. X_i^g. This happens because the additional energy made available by each execution unit of an additional gaining job is no more than that available from a replenishment unit. Hence $\lceil (X_i^g P_r - Y_i^g)/P_r \rceil$ cannot decrease by more than X_i^g increases. We note that monotonicity can also easily be seen by considering the sequence Z (in Theorem 5) which can only take a longer elapsed time to execute with the addition of further jobs, since all execution units require a positive amount of energy.

□

We note that a tighter lower bound can be obtained via the simple expedient of simulating the actual schedule of execution starting from synchronous release of task τ_i and all higher priority tasks. We return to this point in Section 6.

5.8 Priority Assignment

Deadline Monotonic (DM) [14] priority assignment is optimal for fixed priority pre-emptive scheduling of constrained deadline tasks conforming to the classical task model where energy is not considered. In this section, we show that DM priority assignment is also optimal with respect to our sufficient schedulability tests $UB1$ and $UB2$, for energy-constrained systems with both consuming and gaining tasks.

Definition 2. *A priority assignment policy P is said to be optimal with respect to a schedulability test S, if for every task set τ where there exists some priority assignment Q such that the task set is schedulable according to test S, then τ is also schedulable according to test S with the priority ordering given by policy P.*

Theorem 7. *Deadline Monotonic (DM) priority assignment is an optimal priority assignment policy with respect to sufficient schedulability test S ($UB1$ or $UB2$) for task sets comprising any arbitrary combination of consuming and gaining tasks.*

Proof. To prove the theorem, we show that any task set τ that is schedulable according to test S ($UB1$ or $UB2$) under some priority ordering Q remains schedulable according to test S under deadline monotonic priority order P. We do this by transforming priority order Q into priority order P by swapping the priorities of tasks that are adjacent to each other in the priority order, but out of DM order. We show that on every swap the task set remains schedulable according to test S. Let τ_A and τ_B be two tasks in τ which are at adjacent priorities under the initial, schedulable priority ordering, with $D_A > D_B$ and τ_A at a higher priority k than τ_B, which has a priority $i = k + 1$ (i.e. the tasks are out

of DM order). Let the upper bound response time of task τ_B according to schedulability test S be R_i^{UB} in the initial priority order. We now swap the priorities of the tasks, so that τ_B has the higher priority. We consider the following groups of tasks:

(i) $hp(k)$: these tasks have higher priorities than either τ_A or τ_B and so their upper bound response times, according to test S ($UB1$ or $UB2$), are unchanged by the swap.

(ii) $lp(i)$: these tasks have lower priorities than either τ_A or τ_B and so their upper bound response times, according to test S ($UB1$ or $UB2$), are unchanged by the swap, since interference from higher priority tasks does not, *according to the test*, depend on the relative priority order of those tasks.

(iii) task τ_B: now has a higher priority than τ_A, and so is only subject to interference from tasks in $hp(k)$, rather than $hp(k) \cup \tau_A$, hence τ_B remains schedulable.

(iv) task τ_A: is now at priority i with τ_B at higher priority. From the previous schedulable priority ordering, we have $R_i^{UB} \leq D_B \leq T_B$ and $D_B < D_A \leq T_A$, hence $w = R_i^{UB}$ was computed by test S by including exactly one job of τ_A, one job of τ_B and $\lceil w/T_h \rceil$ jobs of each higher priority task $\tau_h \in hp(k)$. We observe that the computation of the busy period length w by test S ($UB1$ or $UB2$) depends only on this set of jobs and not on their relative priorities. We now consider $w = R_i^{UB}$ as a possible value for the response time of task τ_A under the new priority ordering. As $R_i^{UB} \leq T_B$, then there is only one job of task τ_B released in an interval of length w, along with $\lceil w/T_h \rceil$ jobs of each higher priority task $\tau_h \in hp(k)$, and the single job of task τ_A. Therefore, according to test S, R_i^{UB} is *also* the upper bound response time for task τ_A when it is at priority i. Since $R_i^{UB} \leq D_B < D_A$, it follows that task τ_A is schedulable at priority i. □

An exact test (4) for PFP_{ASAP} scheduling with only consuming tasks was given in [2]. We note that the $UB1$ and $UB2$ tests reduce to (4) when there are only consuming tasks, and hence it follows from Theorem 8 that DM priority assignment is also optimal in that case. Similarly, if all tasks are gaining tasks, then the $UB1$ and $UB2$ tests reduce to the classical exact test (3) for FPPS without energy considerations. DM priority assignment is again optimal in that case [14].

We note that it remains an open question whether Deadline Monotonic priority assignment is optimal with respect to an exact analysis for constrained deadline task sets with both consuming and gaining tasks scheduled by PFP_{ASAP}.

6 Performance Evaluation

In this section, we present the results of an empirical investigation, examining the effectiveness of our sufficient schedulability tests.

6.1 Taskset generation

To perform these experiments, we randomly generated approximately 40000 task sets, varying the processor utilization, the energy utilization, and the percentage of gaining tasks. We varied U and U^e in the range $[0.05, 1]$ in steps of 0.05. The proportion of gaining tasks was varied from 0% to 100% in steps of 10% for each pair of values (U, U^e), hence we obtained 100 distinct task sets for each pair (U, U^e). Each tasks set comprised 10 tasks. The task parameters were randomly generated as follows: task processor utilization ($U_i = C_i/T_i$) using the *UUnifast* algorithm [7], task energy utilization ($U_i^e = E_i/T_i \times P_r$) using an adapted version of *UUnifast* to control the type of task generated (gaining or

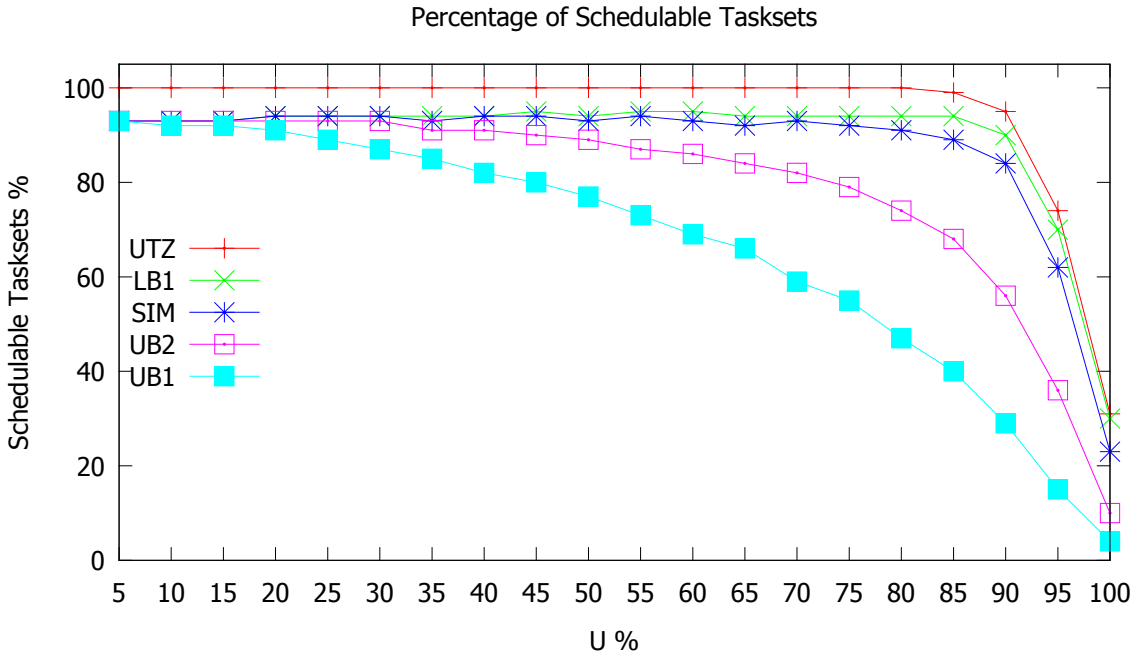


Figure 3: Percentage of Task sets schedulable

consuming), and periods randomly generated between 2 and 25200 time units with a hyper-period limitation technique [16].Task deadlines were implicit and the rate of energy replenishment P_r i was set to 15,

6.2 Schedulability tests investigated

We investigated the performance of the following schedulability tests. UTZ the exact test for FPPS ignoring energy constraints. This was used to provide a schedulability bound, considering only processing time. SIM is an empirical necessary test based on simulating the schedule of PFP_{ASAP} over more than twice the hyper-period, starting with synchronous release and the minimum energy level. This is not guaranteed to reveal the real worst-case scenario, but can be used as a reference for comparison. $UB1$ the sufficient test presented in Section 5.4. $UB2$ the sufficient test presented in Section 5.5. $LB1$ the necessary test presented in Section 5.7.

Figure 3 shows how the percentage of task sets that are deemed schedulable by each of the tests varies with processor utilization. The UTZ test has notionally the highest performance since it ignores energy considerations. When energy is considered, UTZ, $LB1$ and SIM provide necessary tests, upper bounding the number of task sets that could possibly be schedulable. An exact test considering energy would fall somewhere between SIM and $UB2$. We observe that the results confirm that $UB2$ provides a tighter bound than $UB1$, with a larger improvement at higher utilization levels.

6.3 Weighted Schedulability

we present a further set of experiments showing how schedulability depends on different parameters, including energy utilization and the proportion of gaining tasks, via the Weighted Schedulability Measure [6]. As well as processor utilization, task set schedulability is dependent on a number of other key parameters, including: energy utilization, and

the percentage of gaining tasks. Evaluating all possible combinations of these parameters is not possible, instead, the evaluation in this section varies one parameter at a time, with the results presented in terms of the weighted schedulability measure [6].

The figures in this section show the weighted schedulability measure $W_y(p)$ for each schedulability test y as a function of parameter p. For each value of p, this measure combines results for all of the task sets Γ generated for all of a set of equally spaced utilization levels (5% to 100% in steps of 5%).

Let $S_y(\Gamma, p)$ be the binary result (1 or 0) of schedulability test y for a task set Γ with parameter value p:

$$W_y(p) = \left(\sum_{\forall \Gamma} U_\Gamma \times S_y(\Gamma, p) \right) / \sum_{\forall \Gamma} U_\Gamma \tag{12}$$

where U_Γ is the processor utilization of taskset Γ. The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [6]. Weighting the individual schedulability results by task set utilization reflects the higher value placed on being able to schedule higher utilization task sets.

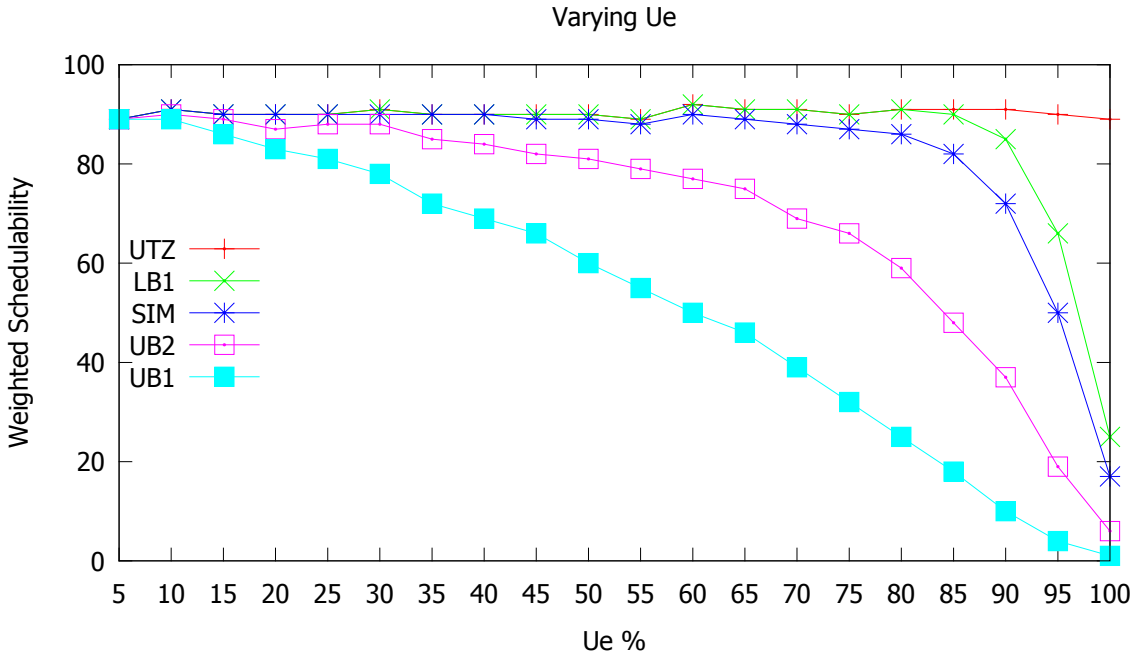


Figure 4: Varying the energy utilization

Figure 4 shows how the weighted schedulability measure for each schedulability test depends on task set energy utilization. The UTZ test ignores energy constraints and hence exhibits minimal variation. The tests that consider energy ($LB1$, SIM, $UB2$, $UB1$) all show the same pattern of behaviour as the classical schedulability tests do against processor utilization, i.e. schedulability reduces at high levels of utilization (energy utilization in this case). We note that the performance of the simple sufficient test $UB1$ degrades with increasing energy utilization.

Figure 5 shows the influence of task set composition. When the task sets comprise 100% gaining tasks, then all of the tests give precisely the same performance. This is

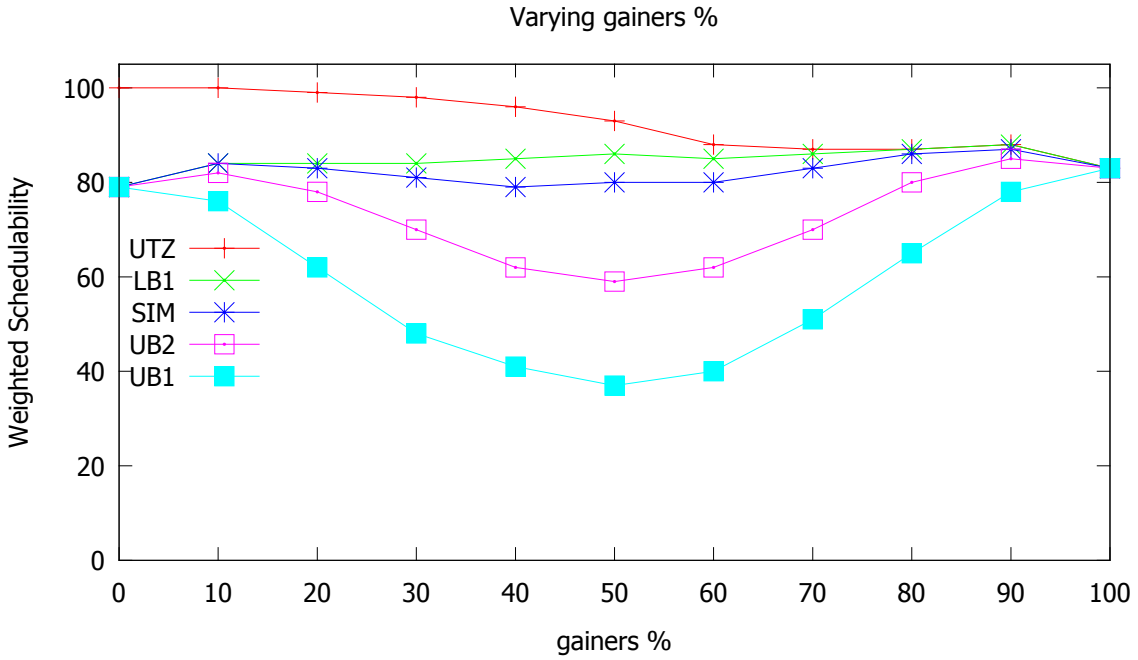


Figure 5: Varying the gaining tasks ratio

because no energy replenishment is needed, and in this case all of the tests reduce to the exact test for fixed priority pre-emptive scheduling with no energy constraints. Similarly, for task sets comprising only consuming tasks (0% gaining tasks), the worst-case scenario is synchronous release with the battery level set to the minimum [2]. This is captured by all of the tests that consider energy ($LB1$, SIM, $UB2$, $UB1$), hence they all have the same performance. (We note that the UTZ test which ignores energy constraints has performance that is notionally better in this case). Between these two extremes, the closer the task sets are to an equal mix of consuming and gaining tasks, the more opportunity there is for consuming tasks to make use of the net energy gain from gaining tasks, and hence the more $UB1$ and $UB2$ diverge from (SIM) and $LB1$. Here, $UB2$ is less impacted since it takes some account of the net energy gain due to gaining jobs that execute ahead of consuming jobs.

Figure 6 shows the impact of constrained deadlines on performance. Here we vary the deadlines from heavily constrained where $D_i - C_i$ is 10% of $T_i - C_i$ to 100% of $T_i - C_i$ (i.e. implicit deadlines). We observe that all of the schedulability tests are influenced by the tightness of deadlines to a similar degree, with heavily constrained deadlines having significant impact on schedulability in all cases.

7 Conclusions and Future Work

In this paper, we addressed the problem of real-time scheduling in energy harvesting systems, where both time and energy constraints have to be met. In such systems, tasks can be classified as *gaining* or *consuming* tasks depending on whether or not the system has a net gain or loss of energy when the task executes. Previous research showed that the energy work-conserving scheduling policy PFP_{ASAP} is optimal among all fixed priority

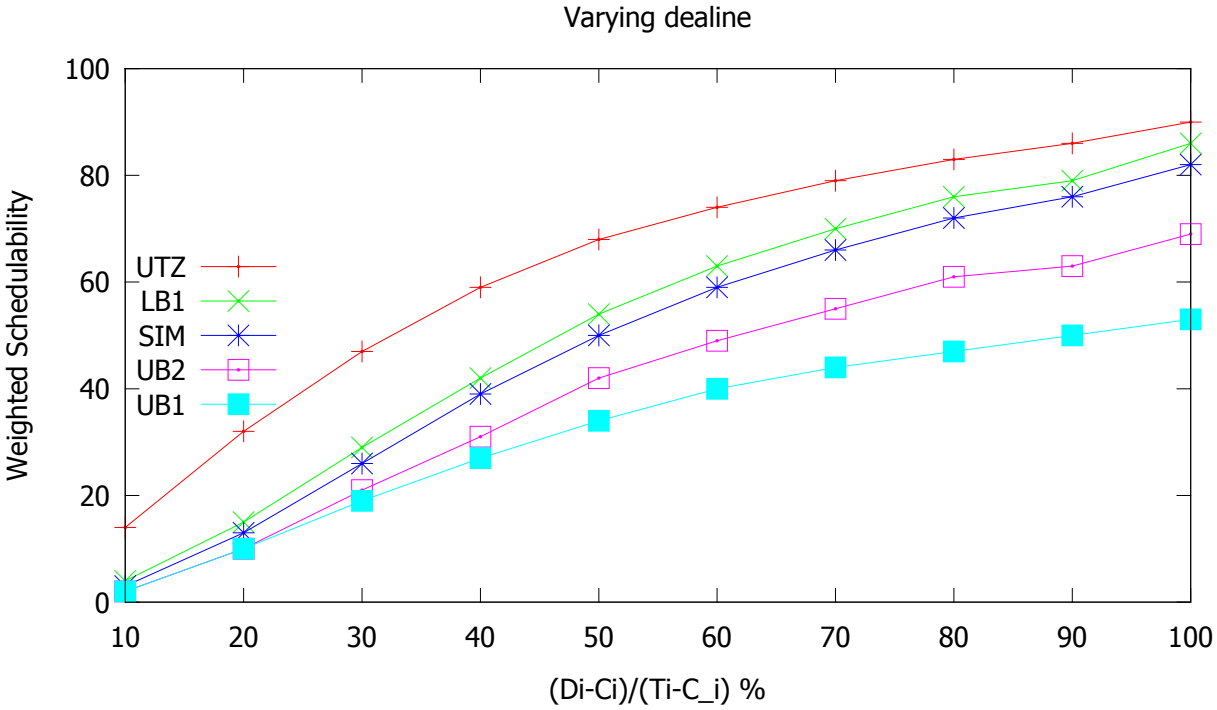


Figure 6: Varying relative deadlines in $[C_i, T_i]$

algorithms for the case where all tasks are consuming tasks.

The major contributions of this paper are as follows: We showed that under PFP_{ASAP}, the critical instant (worst-case scenario) for task sets with both consuming and gaining tasks is not necessarily synchronous release with all other tasks. While we did not identify the specific worst-case scenario for this more general model, we were able to prove a number of properties that it must have. We used these properties to derive two upper bounds on task response times, thus forming two sufficient schedulability tests. In a similar way, we also derived a lower bound response time, and hence a necessary schedulability test. We proved (see technical report [1]) that Deadline Monotonic is the optimal priority assignment policy for PFP_{ASAP} with respect to our sufficient tests. Finally, we evaluated the performance of the sufficient tests in comparison with a number of necessary tests, including an exact test for fixed priority pre-emptive scheduling ignoring energy constraints, and an empirical test based on simulating the schedule for more than a hyperperiod. We found that our tighter upper bound (sufficient schedulability test $UB2$) provides good performance over a wide range of values of different parameters e.g. energy utilization, proportion of gaining tasks etc. (explored using the weighted schedulability measure). There are a number of interesting extensions to this work that we intend to explore in the future. These include an investigation into properties of the worst-case scenario, with the aim of tightening the sufficient tests developed in this paper, and providing further insight into optimal priority assignment policies for this energy-constrained scheduling problem. We also intend to investigate more complex patterns of energy use by tasks, and energy replenishment by the harvester.

References

[1] Y. Abdeddaïm, Y. Chandarli, R. Davis, and D. Masson. Approximate response time for fixed priority real-time systems with energy-harvesting. http://hal.archives-ouvertes.fr/hal-00986340/en, 2014.

[2] Y. Abdeddaïm, Y. Chandarli, and D. Masson. The Optimality of PFP_{asap} Algorithm for Fixed-Priority Energy-Harvesting Real-Time Systems. In *Euromicro Conference on Real-Time Systems*, July 2013.

[3] B. Ahmed-Seddik, G. Despesse, S. Boisseau, and E. Defay. Self-powered resonant frequency tuning for piezoelectric vibration energy harvesters. *Journal of Physics: Conference Series*, 476(1), 2013.

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

[5] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[6] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Annual workshop on Operating Systems Platforms for Embedded Real-Time applications (in conjunction with Euromicro Conference on Real-Time Systems)*, 2010.

[7] E. Bini and G. C. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[8] R. Davis. *On exploiting spare capacity in hard real-time systems.* PhD thesis, University of York, UK, 1995.

[9] R. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, Sept. 2008.

[10] R. I. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *IEEE Real-Time Systems Symposium*, 1993.

[11] H. EL Ghor, M. Chetto, and R. H. Chehade. A real-time scheduling framework for embedded systems with environmental energy harvesting. *Computers and Electrical Engineering*, 37:498–510, 2011.

[12] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *Computer Journal*, 29(5):390–395, 1986.

[13] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems. In *IEEE Real-Time Systems Symposium*, 1992.

[14] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[15] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment, 1973.

[16] C. Macq and J. Goossens. Limitation of the hyper-period in real-time periodic task set generation. In *Conference on Real-time and Embedded Systems*, 2001.

[17] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling with regenerative energy. In *Euromicro Conference on Real-Time Systems*, 2006.

[18] F. Yildiz. Potential ambient energy-harvesting sources and techniques. *Journal of Technology Studies*, 35(1), 2009.

A Proof of Lemma 2

Case (i) sequence X contains solely execution units of consuming tasks. Since all execution units consume energy, then for every element $X[m]$, we have $E_X[m] > P_r$ and so $E^*_X[m+1] > E^*_X[m] + P_r$ hence the maximum number of prior replenishment units is required by the last element in the sequence and is given by:

$$I^*_X[L_X] = I_X[L_X] = \left\lceil \frac{E^*_X[L_X] - E(0)}{P_r} \right\rceil \tag{13}$$

Since the total energy $E^*_X[L_X]$ required by all elements in the sequence is independent of the order of the elements, the elapsed time $I^*_X[L_X]+L_X$ required to execute the sequence is also independent of the order of the elements.

Case (ii) sequence X contains solely execution units of gaining tasks. Since all execution units gain energy, no replenishment units are required and the elapsed time for the sequence equates to its length L_X irrespective of the order of the elements.

B Proof of Lemma 5

We may obtain sequence Y from sequence X by an iterative process of choosing the first execution unit belonging to any gaining task (at position g) and swapping it with that of the last execution unit of any consuming task (at position k) provided that $g < k$. Repeating this process until all consuming execution units come before all gaining execution units transforms sequence X into sequence Y. Let the new sequences produced by this process be $X_1 = X, X_2, X_3 \ldots X_n = Y$. Note at most $L_X/2$ swaps are required. We now show that each swap transforming sequence X_p into sequence X_s where $s = p+1$, results in an elapsed time for sequence X_s that is no shorter than that for X_p, and hence by induction that the elapsed time for sequence Y is no shorter than that for sequence X. Let $X_p[g]$ and $X_p[k]$ be the elements being swapped where $g < k$. Since $X_p[g]$ is an execution unit of a gaining task and $X_p[k]$ is an execution unit of a consuming task, the energy required for these execution units has the relationship $E_{X_p}[g] < E_{X_p}[k]$. Recall that $E^*_{X_p}[m]$ is the energy required to execute all execution units in the subsequence from $X_p[1]$ to $X_p[m]$. It follows that:

$\forall m, 1 \leq m < g \;\; E^*_{X_s}[m] = E^*_{X_p}[m]$

$\forall m, g \leq m < k \;\; E^*_{X_s}[m] = E^*_{X_p}[m] + E_{X_s}[k] - E_{X_p}[g]$

$\forall m, k \leq m \;\; E^*_{X_s}[m] = E^*_{X_p}[m]$

$\Rightarrow \;\; E^*_{X_s}[m] \geq E^*_{X_p}[m]$

Hence the minimum number of replenishment units required to execute the subsequences from the 1st to the $m-$th element of X_p and X_s have the following relationship:$I^*_{X_s}[m] \geq I^*_{X_p}[m]$ (see (6) and (7)). Since the number of execution units in each sequence (X_s and X_p) is the same (i.e. $L_{X_p} = L_{X_s}$), we have: $L_{X_s} + I^*_{X_s}[m] \geq L_{X_p} + I^*_{X_p}[m]$. Thus the elapsed time required to execute sequence X_s is no shorter than that required for sequence X_p. Induction over at most $L_X/2$ steps proves that the elapsed time required to complete sequence Y is no shorter than that required for sequence X.

C Proof of Theorem 3

Consider increasing the length of the window from some arbitrary value w to $w+v$, comparing the dummy schedules used to derive $F^{UB2}(i,w)$ and $F^{UB2}(i,w+v)$ there are two effects: (i) all execution units of gaining jobs move to a later time e.g. $t+v$ rather than t, (ii) new execution units of gaining jobs may be added near the start of the schedule and new execution units of consuming jobs may be added near the end of the schedule. Consider sequence X formed in deriving $F^{UB2}(i,w)$ and sequence Y formed in deriving $F^{UB2}(i,w+v)$ but omitting all of the execution units of the new jobs from (ii). Sequences X and Y contain the same set of elements. Since all execution units of gaining jobs are v time units later in the dummy schedule used to construct sequence Y, it follows that sequence Y can be formed from sequence X by swapping later gaining execution units in X for earlier consuming execution elements, and as necessary re-ordering sub-sequences containing solely gaining or solely consuming execution units (Lemma 2). Hence the elapsed time required to execute sequence Y is no shorter than that required for sequence X. Consider a further sequence Z, if there were no additional jobs from (ii) then sequence Z is identical to sequence Y, otherwise it may be obtained from sequence Y by adding execution units for the missing jobs. Since all execution units require energy, addition of execution units into a sequence at any point cannot decrease the elapsed time required to execute the sequence. Hence the elapsed time required to execute sequence Z is no shorter than that required to execute sequence X.

D Proof of Theorem 6

Consider the formula for $F^{LB1}(i,w)$. Since w appears only in the ceiling functions, it follows that X_i^g, X_i^c and Y_i^g are all non-decreasing functions of w. Further, $(X_i^g P_r - Y_i^g)$ represents the net energy increase while all the gaining jobs execute. Since every execution unit of a gaining task is by definition energy positive, this quantity is also a non-decreasing function of w. Thus $Y_i^c - (X_i^g P_r - Y_i^g)$ may decrease with increasing w. The largest possible decrease is obtained when Y_i^c remains at the same value, while $(X_i^g P_r - Y_i^g)$ increases, hence $\lceil (X_i^g P_r - Y_i^g)/P_r \rceil$ decreases. However, such a decrease is always at least compensated for by the increasing value of the first term in (11), i.e. X_i^g. This happens because the additional energy made available by each execution unit of an additional gaining job is no more than that available from a replenishment unit. Hence $\lceil (X_i^g P_r - Y_i^g)/P_r \rceil$ cannot decrease by more than X_i^g increases. We note that monotonicity can also easily be seen by considering the sequence Z (in Theorem 5) which can only take a longer elapsed time to execute with the addition of further jobs, since all execution units require a positive amount of energy.

E Priority Assignment

Deadline Monotonic (DM) [14] priority assignment is optimal for fixed priority pre-emptive scheduling of constrained deadline tasks conforming to the classical task model where energy is not considered. In this section, we show that DM priority assignment is also optimal with respect to our sufficient schedulability tests $UB1$ and $UB2$, for energy-constrained systems with both consuming and gaining tasks.

Definition 3. *A priority assignment policy P is said to be optimal with respect to a schedulability test S, if for every task set τ where there exists some priority assignment Q such that the task set is schedulable according to test S, then τ is also schedulable according to test S with the priority ordering given by policy P.*

Theorem 8. *Deadline Monotonic (DM) priority assignment is an optimal priority assignment policy with respect to sufficient schedulability test S ($UB1$ or $UB2$) for task sets comprising any arbitrary combination of consuming and gaining tasks.*

Proof. To prove the theorem, we show that any task set τ that is schedulable according to test S ($UB1$ or $UB2$) under some priority ordering Q remains schedulable according to test S under deadline monotonic priority order P. We do this by transforming priority order Q into priority order P by swapping the priorities of tasks that are adjacent to each other in the priority order, but out of DM order. We show that on every swap the task set remains schedulable according to test S. Let τ_A and τ_B be two tasks in τ which are at adjacent priorities under the initial, schedulable priority ordering, with $D_A > D_B$ and τ_A at a higher priority k than τ_B, which has a priority $i = k + 1$ (i.e. the tasks are out of DM order). Let the upper bound response time of task τ_B according to schedulability test S be R_i^{UB} in the initial priority order. We now swap the priorities of the tasks, so that τ_B has the higher priority. We consider the following groups of tasks:

(i) $hp(k)$: these tasks have higher priorities than either τ_A or τ_B and so their upper bound response times, according to test S ($UB1$ or $UB2$), are unchanged by the swap.

(ii) $lp(i)$: these tasks have lower priorities than either τ_A or τ_B and so their upper bound response times, according to test S ($UB1$ or $UB2$), are unchanged by the swap, since interference from higher priority tasks does not, *according to the test*, depend on the relative priority order of those tasks.

(iii) task τ_B: now has a higher priority than τ_A, and so is only subject to interference from tasks in $hp(k)$, rather than $hp(k) \cup \tau_A$, hence τ_B remains schedulable.

(iv) task τ_A: is now at priority i with τ_B at higher priority. From the previous schedulable priority ordering, we have $R_i^{UB} \leq D_B \leq T_B$ and $D_B < D_A \leq T_A$, hence $w = R_i^{UB}$ was computed by test S by including exactly one job of τ_A, one job of τ_B and $\lceil w/T_h \rceil$ jobs of each higher priority task $\tau_h \in hp(k)$. We observe that the computation of the busy period length w by test S ($UB1$ or $UB2$) depends only on this set of jobs and not on their relative priorities. We now consider $w = R_i^{UB}$ as a possible value for the response time of task τ_A under the new priority ordering. As $R_i^{UB} \leq T_B$, then there is only one job of task τ_B released in an interval of length w, along with $\lceil w/T_h \rceil$ jobs of each higher priority task $\tau_h \in hp(k)$, and the single job of task τ_A. Therefore, according to test S, R_i^{UB} is *also* the upper bound response time for task τ_A when it is at priority i. Since $R_i^{UB} \leq D_B < D_A$, it follows that task τ_A is schedulable at priority i. □

An exact test (4) for PFP_{ASAP} scheduling with only consuming tasks was given in [2]. We note that the $UB1$ and $UB2$ tests reduce to (4) when there are only consuming tasks, and hence it follows from Theorem 8 that DM priority assignment is also optimal in that case. Similarly, if all tasks are gaining tasks, then the $UB1$ and $UB2$ tests reduce to the classical exact test (3) for FPPS without energy considerations. DM priority assignment is again optimal in that case [14].

We note that it remains an open question whether Deadline Monotonic priority assignment is optimal with respect to an exact analysis for constrained deadline task sets with both consuming and gaining tasks scheduled by PFP_{ASAP}.