A DYNAMIC PROGRAMMING SCHEDULE TRADING OFF QUALITY AND STABILITY IN TASK ALLOCATION FOR ENERGY-NEUTRAL INTERNET OF THINGS DEVICES HARVESTING SOLAR ENERGY

Antonio Caruso[1], Stefano Chessa[2], Soledad Escolar[3], Fernando Rincón[3], Juan Carlos López[3]
[1]Dept. of Mathematics and Physics "Ennio de Giorgi", University of Salento, Lecce, Italy, [2]Computer Science Department, University of Pisa, Pisa, Italy, [3]School of Computing Science, University of Castilla-La Mancha, Ciudad Real, Spain

NOTE: Corresponding author: Antonio Caruso, antonio.caruso@unisalento.it

Abstract – *Energy neutrality in an energy harvesting Internet of Things (IoT) device ensures continuous operation of the device by trading performance with energy consumption, and a way to achieve this is by adopting a task-based model. In this model, the device embeds several alternative tasks with different ratio energy-cost/quality and a scheduler that, depending on the current energy production and battery level, runs at any time the best task to maximize the performance while guaranteeing energy neutrality. In this context, this work proposes a novel scheduling algorithm that takes into account also the stability of the device, by minimizing the leaps of quality between two consecutive tasks in the scheduling. We show by simulation and by experiments on a low-power IoT platform that the proposed algorithm greatly improves the stability of the device with respect to the state-of-the-art algorithms, with a marginal worsening of the overall quality of the tasks executed.*

Keywords – Internet of Things, multi-objective scheduling problem, scheduling stability, solar energy harvesting

1. INTRODUCTION

Energy harvesting of solar power [1] is a common solution to power Internet of Things (IoT) devices in outdoor environments, due to the reliability of harvesters (solar panels) and to the great power density provided by this source. However, even this solution may not be sufficient to guarantee continuous operation of a device as there is no guarantee on the amount of energy that can be harvested over time. In this condition, the simple solution of over-sizing the solar panel and the battery results in increased costs and environmental impact of the device [2]. A promising alternative, consists of the energy-neutral design of the device [1], which can dynamically trade device performance (and thus the *quality* of its output, whatever it is) with power consumed (its *load*). Its objective is to let the device consume less energy than that produced by the harvester over each reference time frame (in the case of solar energy harvesting this time frame is usually one day, following the natural cycle of the sun). In the seminal work [1], energy neutrality was achieved by acting on the duty cycle of the device to modulate the load to match the forecast of the energy production throughout a day. For example, if the device operates as a sensor, increasing the duty cycle improves the quality of sampling and increases the power consumption, while decreasing the duty cycle reduces the quality of sampling and decreases the power consumption.

The concept of energy neutrality gave rise to a rich and intense research [3, 4, 5, 6, 7, 8, 9, 10]. In particular, we introduced a task-based model [11]; in this model the device embeds several alternative versions of the program to run (called tasks), each characterized by a different profile in terms of power consumption and quality of its output. In this way, energy neutrality can be achieved by finding a suitable schedule of the tasks throughout a day that also maximizes the overall quality. As the consequent optimization problem is NP-Hard, the first approaches [12, 13, 4, 14] adopted greedy-based strategies to implement the optimization. More recent work [5, 15] made further progress by introducing a dynamic programming algorithm that solves the optimization problem and that, despite its nature, can be implemented efficiently even in low-power devices.

The current work fits this trend of research by addressing the problem of stability of the task schedule found by the optimization algorithm. This problem arises when the optimal task schedule contains very large quality leaps between two consecutive tasks. These leaps occur due to the need of achieving the energy neutrality constraint, but they may be a problem from the point of view of the usability of the device because they make its output widely variable making it more difficult to interpret/manage by the user. Consider for example a device that operates as a sensor: a schedule may contain two consecutive tasks, one operating at maximum duty cycle (and thus at a maximum sampling rate) and the next with the lowest duty cycle (and thus with a very low sampling rate), which may not be the best option from the point of view of analysis/use of the sensed data.

Our approach, of which a preliminary version appeared recently in [16], formalizes the problem of optimal task scheduling as a multi-objective scheduling problem, which aims to (i) maximize the overall task quality and to (ii) minimize the leaps between consecutive tasks. Since with this formulation the problem requires powerful machines on edge/cloud servers to be solved, we introduce a relaxation of the problem where these

two objectives (maximization of quality and minimization of leaps) are combined into one single objective that takes both into account. We also show that, with this relaxed formulation, the problem can be solved by using a dynamic programming algorithm that can be efficiently tuned to be executed even on very low power IoT devices. We also present simulative and experimental results obtained with a low-power IoT device of the Arduino class to show that our approach significantly reduces the leaps as compared to the state of the art, and that this reduction comes with a very small penalty in terms of quality of the overall schedule. Note that the present work significantly extends the preliminary work presented in [16] in the state of the art and in the evaluation section, by extending the simulations and by adding a novel experimental part conducted over real IoT devices.
Summarizing, the main contributions of this work are:

- a novel multi-objective scheduling problem for energy harvesting IoT devices that maximizes the quality of the scheduling and minimizes the leaps among consecutive tasks;
- a (novel)relaxed formulation of the multi-objective scheduling problem that combines both objectives into one single objective function;
- a novel dynamic programming algorithm that solves the relaxed problem even on low-power devices;
- an experimentation over Arduino class devices that shows the feasibility of the approach and that shows the reduction in terms of number of leaps in the scheduling with respect to the state of the art.

The rest of the paper is organized as follows: Section 2 presents the state of the art, Section 3 explains the model of energy harvesting, sections 4 and 5 introduce the optimization problem and the scheduling algorithm, respectively, Section 6 presents the simulation results and Section 7 draws the conclusions and future work.

2. RELATED WORK

In a seminal work Kansal [1] introduced the condition for energy neutrality. It states that, given a certain reference time period, the battery level at the end of the period must always be greater than or equal to the battery level at the beginning of the period. To satisfy this condition, there exist four major strategies: Dynamic Voltage and Frequency Scaling (DVFS), duty cycle adaptation, sensing and sampling rate adaptation and task scheduling. A review of these techniques can be found in [9, 10], while recent specific surveys on task scheduling for energy neutrality are in [17, 18].
Specifically, we have contributed to this last category with several works that propose algorithms devoted to find the optimal scheduling that maximizes the utility of the applications running on top of the device while keeping the energy neutrality condition, and a real testbed for them [3, 15]. We also considered progressively richer scenarios, from a simple one with one energy harvesting device [11, 14] to scenarios involving more devices connected to a gateway or sink, that also has an energy harvesting capability [12, 13, 4]. In a recent work [5] we introduced a scheduling algorithm based on dynamic programming that finds the optimum scheduling, and we proved that this algorithm is even feasible in very low-power devices by considering the limited resolution of these devices in sampling the battery charge and the power production of the photovoltaic panel.
Similar to [5, 19], other work has addressed the problem of optimization of the tasks scheduling under different scenarios [20]. In [6] a Mixed Integer Linear Programming (MILP) formulation is devoted to the optimization of the quality of security of the tasks scheduled under energy-constraints on a battery-powered MultiProcessor SoC (MPSoC). The security may be achieved by a set of cryptographic algorithms, each one with a time overhead and an energy consumption, and a quality associated with the service. The experimentation results show how the algorithm proposed saves energy and improves quality of security for MPSoC compared to other approaches. In [7] the IoT devices, powered by renewable energy, make a computational offloading using resource-rich fog servers (which are powered by the grid), and propose a strategy for the optimization of real-time tasks executed on the fog aimed at maximizing both the system QoS and the QoS of individual real-time applications. The experimental results demonstrate an improvement of the overall and individual application QoS by up to 101.93% and 59.30%, respectively. The work in [8] describes an approach in two-stages for real-time electricity pricing and scheduling. To this end, an optimization strategy for IoT device scheduling modeled as a 0-1 Knapsack problem is proposed (i.e. the device can be or not be scheduled). The authors leverage our work in [19] to define an energy-harvesting environment where the devices may also buy electricity when their harvester does not provide enough power. Their results demonstrate that the accepted requests ratio of the energy-harvesting scenario is up to 20% greater than the ratio of the scenario where the energy harvesting is not applied, due to the fact that the energy consumption of a device per time slot is reduced by the harvested energy, and thus more requests can be accepted. In [21] the authors present and evaluate different approaches for solar power energy prediction suitable for the application in IoT devices harvesting solar power. They show that using public weather forecast and solar angles derived by the deployment position and orientation can improve the energy budget of the devices by more than 20%.
In Table 1 we summarized the major '*features*' presented in the papers discussed above. This paper starts with the algorithm presented in [5], in particular we observed a critical limit of the optimal computed schedule. In some scenarios, when the production of the solar panel drops for some reason, the optimal schedule is comprised of a

Table 1 – A comparison of different topics studied in the related works. The four papers in the last columns are survey papers.

	[1]	[11]	[12]	[13]	[14]	[4]	[5]	[19]	[15]	[3]	[20]	[6]	[7]	[8]	[21]	[9, 10]	[17, 18]
Survey paper																	•
Greedy-algorithm	•	•	•	•	•	•											
Model of solar prediction	•	•	•	•	•	•	•	•	•	•	•	•	•		•		
Dynamic programming							•	•	•	•	•	•		•			
Adaptive duty-cycle	•																
Tasks scheduling		•	•	•	•	•	•	•	•	•	•	•	•	•			
Tested on device		•						•	•	•	•		•				
MultiNode scheduler				•	•						•			•			
Statistical approach								•	•	•	•		•	•	•		
Security model												•					
IoT-fog-cloud game												•	•				

sequence of tasks with an abrupt change in quality from consecutive times. This kind of strong oscillations in the quality of tasks, even in the case of an optimal schedule cannot be desirable if the designer prefers a *graceful degradation* of the quality over time. We decided, that a new approach that provides a more progressive degradation of the behavior and quality of the tasks executed on the device across the time was required, and presented a preliminary work on it in [16]. The trade-off between quality and stability is already studied in the theory of scheduling, but it seems, for the author's knowledge, that has never been addressed before in the case of energy-neutral scheduling in energy harvesting Internet of Things.

3. MODEL OF ENERGY HARVESTING DEVICES

An energy-harvesting system [22] comprises four major elements: a *harvester*, a *power converter*, a *load*, and an *energy storage*. A solar harvester is a panel that embeds photovoltaic modules that convert sunlight into electricity by means of the photovoltaic effect. The output power of a solar panel depends on the the surface S of the panel itself, on the irradiance D (i.e. density of incident power on the surface of the panel), and on the efficiency of the energy converter μ (i.e. the percentage of conversion of the irradiance into electrical power). A power converter is an electronic device that regulates the power emitted by the harvester to avoid uncontrolled fluctuations in the output voltage and in the output current that may cause damage to the powered device. The *load* is the computational device of the system that demands energy to carry on its operations. It is usually composed by a microcontroller unit, one radio unit for wireless communications, one or several sensors, and an external memory for permanent data storage. Finally, the energy-storage provides an energy buffer to sustain the load when the harvester provides insufficient power. In general, the energy storage is a rechargeable battery, but in some cases it may be replaced by capacitors or super-capacitors as primary or secondary energy storage because of their longer lifetimes and larger range of operating temperatures.

4. SCHEDULING PROBLEM

We consider a low-power IoT device equipped with a micro-controller, memory, wireless interface and a set of sensors/actuators and an energy harvesting subsystem comprising a rechargeable battery (of maximum capacity B_{max}) and a solar panel. We assume that the minimum charge of the battery that lets the device operate is B_{min} and we denote with η the charging efficiency of the battery (this accounts for loss of energy due to its conversion and battery charging process).

In order to optimize the use of the battery and thus to achieve energy neutrality, the device is programmed in order to modulate the load (that is, its energy consumption), so to consume, within a predefined time frame, the same amount of energy that it produces. Considering that the solar energy has a natural cycle of 24 hours, we take this time frame to be exactly 24 hours.

In order to modulate the load of the device, we adopt the same approach of [5]. In particular we assume that the device is preloaded with a set of N alternative tasks $\mathcal{T} = \{t_1, \dots, t_N\}$ each characterized by a different energy consumption $c_i, i \in [1, N]$ and quality of service $q_i, i \in [1, N]$. For example, different tasks may implement the functionality of the device sampling at different frequencies, they may execute a lighter/more complex data aggregation or they may communicate less/more frequently with the base station. It is important to stress that in our model the tasks are alternative, that is, they all implement an equivalent functionality but with different performances (and thus with different qualities of service and energy consumption).

Still to the purpose of load modulation, the device also embeds a scheduler that selects the tasks to be executed in order to guarantee the energy neutrality of the device. We stress that, even in IoT devices that are really constrained in memory or computational capacity, it is possi-

ble to build a very small scheduler, see for example FUSIX[1] a small Unix-like OS implementation for the Raspberry Pico. In particular, the scheduler assumes that the 24 hour time frame is divided into K time slots of the same duration $24h/K$. At the beginning of each day the scheduler produces an energy-neutral assignment of the tasks to each slot based on the expected energy production in each slot across the day.

Considering that the expectation on the energy production in each slot may not be met, if the device is deviating from the energy neutrality target then the scheduler can be executed again to recompute the assignment of the tasks to the remaining slots in the day. Note that the scheduler executes an optimization: on the one hand it should meet the energy neutrality constraint, on the other hand it has to maximize the overall quality of the assignment.

Note that, if we consider values of K from 12 to 288 (that we consider reasonable for low-power IoT devices) the duration of a slot (and thus the actual scheduling of the tasks) ranges from 2 hours to 5 minutes, which is much higher than a classical OS scheduler. Hence, we consider it a high-level scheduler that works jointly with the usual low-level scheduler running every few milliseconds to support interrupts, device asynchronous events, and other usual OS activities. In this light, the concept of task presented in this work can also be interpreted differently; a high-level task can be mapped to different low-level tasks with the constraints that when they are scheduled in a time slot their total energy requirements and quality are expressed by the high-level task cost and quality, respectively.

For each $i \in [1, K]$, let e_i be the expectation for the energy to be produced in slot i, B_i be the energy charge of the battery at the beginning of slot i, b_{K+1} be the battery charge at the end of the last slot, and let x_{ij} be a Boolean variable which is $x_{ij} = 1$ iff task t_j (with $j \in [1, N]$) is assigned by the scheduler to slot i. Given assignment x_{ij} for all $i \in [1, K]; j \in [1, N]$, we denote with $\hat{q}_i$ the quality of the task assigned to slot i and with $\hat{c}_i$ the energy consumption of the task assigned to slot i. Hence : $\hat{q}_i = \sum_{j=1}^{N} x_{ij} q_j$ and $\hat{c}_i = \sum_{j=1}^{N} x_{ij} c_j$.

On this basis, and considering that the battery charge cannot exceed B_{max}, the expected battery charge at the beginning of a generic slot $i+1$ can be computed as the minimum between B_{max} and the battery charge at the beginning of the slot i plus the amount of energy that is produced in excess of the consumption minus the energy that is consumed in excess of the production. In a formula, this can be expressed as: $B_{i+1} = \min\{B_{max}, B_i + \eta[e_i - \hat{c}_i]^+ - [\hat{c}_i - e_i]^+\}$ (where we use the notation: $[x]^+ = \max(x, 0)$ for any arbitrary integer x), and the constraint of energy neutrality is expressed as $B_{K+1} >= B_1$.

In [5], the optimization problem of the scheduler has been formulated as an integer linear programming problem,

[1] https://www.raspberrypi.com/news/how-to-get-started-with-fuzix-on-raspberry-pi-pico/

Table 2 – Table of main symbols used in the model and algorithm.

N	Number of tasks
K	Number of slots of time in a period
t_i	task with index $i = 0, \dots, N$
q_i	quality of a task (for each task)
c_i	energy cost of a task (in milliAmpere)
μ	charging efficiency of the battery
B_{min}	Minimum level of the battery in milliAmpere
B_{max}	Maximum level of the battery in milliAmpere
$B_{\text{start}} = B_0$	Initial level of the battery in slot 0.
γ	multi-objective optimization weight factor
$[x]^+$	$\max(0, x)$

where the only objective is to maximize the overall quality of service of the device. However, as already discussed in the introduction, this approach is not fully satisfactory because it may also result in drastic changes of QoS between the tasks assigned to consecutive slots, thus making the overall behavior of the device unstable from a practical point of view. Instead, here we seek an optimization that tends to moderate any degradation of the QoS among consecutive slots by introducing an additional optimization objective. The consequence is that the problem of the scheduler now becomes a multi-objective optimization problem that can be formalized as follows:

Problem 1 (*Multi-objective* Stable *plan Scheduling*)

$$\text{maximize} \quad z = \sum_{i=1}^{K} \hat{q}_i \tag{1}$$

$$\text{minimize} \quad v = \max_{i \in [1,K]} |\hat{q}_i - \hat{q_{i+1}}| \tag{2}$$

$$x_{ij} \in \{0, 1\} \ \forall i \in [1, K], j \in [1, N] \tag{3}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \tag{4}$$

$$B_{i+1} = \min\Big\{B_{\text{max}}, B_i + \eta[e_i - \hat{c}_i]^+ - \\ -[\hat{c}_i - e_i]^+\Big\} \ \forall i \in [1, K] \tag{5}$$

$$B_{\text{min}} \leq B_i \ \forall i \in [1, K] \tag{6}$$

$$B_1 \leq B_{K+1} \tag{7}$$

where $[x]^+ = \max(x, 0)$

In the problem there are two objective functions: (1) is the sum of the quality of the tasks assigned to the slots that should be maximized, while (2) expresses the requirement that the sum of the losses of the quality of service from each slot to the next one should be minimized. For what concerns the constraints, (3) and (4) express the requirement that exactly one task must be assigned per slot; (5) expresses the constraint of the battery charge at the next slot, that depends on the task assignment as already discussed; (6) expresses the requirement that the battery charge should never go below the minimum, otherwise the device would stop working; (7) expresses the energy neutrality constraint.

Although this formulation takes into account the requirement of stability of the scheduling by aiming also at reducing the differences in QoS between consecutive slots, it requires multi-objective optimization and an objective function (2) that is not linear, which is very complex to implement (if ever possible) in a resource-constrained IoT device. For this reason we propose an alternative problem which keeps the same constraints, but that replaces the two objective functions with a single function and with the non-linearity limited only to the absolute value. Specifically, we introduce a penalty term to the sum of the QoS that takes into account the variation of QoS between consecutive slots as follows:

$$\textit{maximize} \quad z = \sum_{i=1}^{K} \hat{q}_i - \gamma \, [|\hat{q}_i - \hat{q_{i+1}}| - 1 \,]^+ \qquad \textit{(8)}$$

Note that, 1) the fuction $[\cdot]^+$ is used to get always a positive penalty factor, and 2) the multiplicative factor γ is used to modulate the impact of the differences between task quality with respect to the goal of maximizing the total quality, 3) the -1 term is introduced to discount the penalty factor when the schedule changes with minimal variation, i.e. when we move from two tasks with minimum quality differences (in particular this factor can be adjusted to be at least the cost of the idle task, or the latter can be normalized to be 1 as we assume in the simulation section). In [5] we proved that the problem of *Energy Neutral Optimal Scheduling* is NP-Hard. However, we show that it can be solved by a proper dynamic programming approach even in low-power devices. Clearly this formulation is more complex than the original one, but it remains NP-Hard. To solve this problem we propose in the next section an algorithm based on dynamic programming that is computationally feasible even in low-power devices.

5. ALGORITHM

We propose a dynamic programming algorithm that is inspired to a variation of the unbounded knapsack problem, with the major difference that the solution is computed proceeding backwards with respect to the time slot. In practice, we compute a table M of size $K \cdot B_{max}$, where each row is associated with a slot and each column to a level of battery, and the value of $M(k, b)$ for $k \in [1, K]$ and $b \in [0, B_{max}]$ is the value of the objective function, i.e. the maximum quality of a schedule that is energy neutral starting with a battery of b and using k slots. The resulting Bellman equations, which adopt a *penalty factor* p, are:

$$opt(k, b) = \max_{i=1\ldots N} \{ \; q_i + opt(k{+}1, B'(i)) - p \mid B'(i) \geq B_{\min} \}$$

with:

$$B'(i) = \min\{B_{\max}, b - c_i + e_i\}, p = \gamma([|\hat{q}_i - \hat{q_{i+1}}| - 1]^+) \qquad \textit{(9)}$$

The Python code with the function `ScheduleWithPenalty` that computes the value (quality) of the optimal solution based on the above recursive equations are in Listing 1, while the function `BuildSchedule` in Listing 2 is used to construct the optimal schedule. In the following we discuss both in detail, note that all indexes for array and matrices are zero-based.

The function `ScheduleWithPenalty` uses two matrices: M and I (lines 3,4) initialized with zeros, of size $K \times (B_{\max} + 1)$. The role of matrix M has been already explained above, while matrix I is used to keep track of the best task associated with each solution in $M(k, b)$; the value of 0 here is used to signal that there is no task that can be scheduled, so the values stored in I are the indexes of the tasks increased by one. For each level b of the battery the algorithm starts with the base case at the beginning of the period associated with the last slot ($k = K-1$) (lines 7-13). In the inner loop (lines 9-13) the algorithm finds the best task (i.e. with highest quality) that results in an energy-neutral schedule, i.e. if it starts with a battery level b it must end with battery level above B_1 and the test in line 10 skips all tasks that do not guarantee this condition. The solutions (quality and index of the task to be scheduled) are stored in lines 14,15. The next loops (line 17-32) compute the best schedule for the remaining slots in decreasing order. Given the slot k and battery level at the beginning of the slot b, the loop in lines 20-30 finds the best task to do this, it computes the value of $B'(i)$ as in Equation 9 (line 21). The algorithm uses this value to: i) check that it satisfies the constraint (6) of the model; ii) compute the penalty factor (line 26-27) and the new quality using Equation *(8)*. At the end the solutions are stored in M, I (lines 31,32).

The function `BuildSchedule` builds the schedule `S`, with K indexes of tasks: it starts with the value of `I[0][Bstart]` (the index of the best task for the first slot, with initial battery equal to `Bstart`$= B_1$); computes the residual battery at the beginning of the next slot; checks that this level is feasible for B_{min} and then it updates the iteration variable for the next iteration, filling all values of S. At the end it returns a Python list `S` with the tasks scheduled in each slot and the quality of this schedule. Note that the scheduling here is computed without taking into account the penalty factor, in order to be comparable with the original algorithm. Note also that the code is not optimized at all for performance (memory or time), but for clarity.

After building the schedule S as shown in Listing 2, we introduce here two metrics to evaluate the degree of stabilization of the algorithm. Since, without loss of generality we ordered the tasks in increasing order of quality, so the difference between task indexes is proportional to their difference in quality. We define the vector of leaps J as the distance between the task index assigned to slot i and to the next slot $i + 1$:

$$J[i] = |S[i] - S[i + 1]| \qquad \forall i \in [1 \ldots K]$$

```python
def ScheduleWithPenalty(K,Bstart,Bmin,Bmax,E,
                        Tasks,gamma=0.5):
  M = np.zeros( (K,Bmax+1))
  I = np.zeros( (K,Bmax+1), dtype=int)
  # last slot k = K-1
  k = K-1
  for b in range(Bmax,-1,-1):
    qmax,idmax = -100,0
    for i,task in enumerate(Tasks):
        if b - task.cost + E[k] >= Bstart and
           task.quality > qmax:
           qmax = task.quality
           idmax = i+1
    M[k][b] = qmax if qmax != -100 else 0
    I[k][b] = idmax
  # other slots in decreasing order
  for k in range(K-2,-1,-1):
    for b in range(Bmax,-1,-1):
      qmax,idmax = -100,0
      for i,task in enumerate(Tasks):
        Bprime = min(b-task.cost+Eprod[k],Bmax)
        if Bprime >= Bmin:
          q = M[k+1][Bprime]
          if (q == 0): continue
          j = I[k+1][Bprime]-1
          dq = task.quality - Tasks[j].quality
          penalty = max(0,gamma*(abs(dq)-1))
          if q + task.quality - penalty > qmax:
              qmax = q + task.quality - penalty
              idmax = i+1
      M[k][b] = qmax if qmax != -100 else 0
      I[k][b] = idmax
```

Listing 1 – Python implementation of the dynamic programming scheduler with *index distance penalty*.

By taking J we define two simple statistics on this vector to measure the stability of a schedule:

- **sum**: the sum of the leaps in J, i.e. sum=$\sum_{i=1}^{K} J[i]$.
- **max**: the maximum gap in the vector J, i.e. max = max(J).

We observe that the schedule that minimizes the jumps, and so both quantity, sum, max, is obtained when we consider a constant schedule, i.e. a schedule with the same task in all slots. This, clearly neglect, in some way, the overall goal of modulating the load in a different way among slots in order to better use the energy produced by the panel. We will see, in the next section, that when $\gamma = 1$ the algorithm approaches this kind of schedule, trying in any case to get the best quality and be energy-neutral.

6. EVALUATION

Differently to our original work described in [16], where we evaluated the algorithm proposed in Listing 1 only by simulation, in this paper we use the Arduino DUE platform [2] for experimentation. Arduino DUE is the first Arduino board based on a 32-bit ARM core microcontroller,

[2]Arduino DUE: `https://docs.arduino.cc/hardware/due`

```python
def BuildSchedule(K,Bmin,Bmax,Bstart,Panel,Tasks):
    S = [0]*K
    B = Bstart
    for i in range(K):
        S[i] = I[i][B]-1
        if (S[i] < 0):
            return (S,0)
        B = min(B + Panel[i] - Tasks[S[i]].cost,
                                          Bmax)
        if B < Bmin:
            return ([],0)
    if B < Bstart:
        return ([],0)
    return(S,sum(Tasks[s].quality for s in S))
```

Listing 2 – Python code that construct the schedule from I, check that everything is feasible and recompute the quality.

with an 84 MHz clock, 96 KB SRAM and 512 KB of memory flash. The objectives of the evaluation are to demonstrate that the proposed algorithm can be executed on a real-world platform with limited capabilities such as an Arduino board and to compare the performance of our algorithm against the one described in [5] (hereafter called *IoT algorithm*), which computes the optimal schedule with the sole objective of maximizing the overall quality of the tasks without taking into account any task stabilization requirement.

Table 3 – Evaluation parameters: day divided in 24 hours, 10 tasks with costs c_i and quality q_i and the values of γ used to compute the penalty.

Parameter	Value
K	24
N	10
B_{min}	160 (mAh)
B_{max}	[2000, 1000] (mAh)
B_1	[1800, 800] (mAh)
c_i	[1, 22, 33, 43, 53, 63, 73, 84, 94, 104]
q_i	[1, 22, 32, 42, 51, 61, 71, 81, 91, 100]
γ	[0.1, 0.2, ..., 1.0]

To this end, we have used the parameters shown in Table 3: the number of slots (K) and the number of tasks (N) with the corresponding cost c_i (in mAh) and quality q_i (in percentage). We assume two different battery configurations: 1) $B_{max} = 2000$ mAh and $B_1 = 1800$ mAh; and 2) $B_{max} = 1000$ mAh and $B_1 = 800$ mAh, both with a minimum battery level $B_{min} = 160$ mAh. The values of γ range between $[0.1 \ldots 1.0]$, which result in different penalties and consequently different schedules. To emulate the energy harvesting capability of the board, we simulate the energy production of the solar panel on two scenarios: 1) the *overproduction* scenario, which represents a scenario where the energy production in each hour is high (taken in summer), the sunrise is 8.00am and the sunset is 19.00pm; and 2) the *underproduction* scenario, which represents a reduction of 20% of the energy production with regard to the first scenario. Table 4 shows the relevant parameters for the energy production in both sce-

Table 4 – Daily energy production in mAh for a sunny scenario E_{over} and a cloudy E_{under}. Energy is generated from slot 8 to slot 19; for the rest of slots the production is 0.

Energy	Slot																							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
E_{over}	0	0	0	0	0	0	0	19	110	224	285	335	350	331	283	134	20	18	8	0	0	0	0	0
E_{under}	0	0	0	0	0	0	0	15	88	179	228	268	280	264	226	107	16	14	6	0	0	0	0	0

narios for the slots in $[8, 19]$ in which there is energy production. The code is available in `https://github.com/sescolar/ngcc`.

Table 5 shows the comparison of the IoT and stabilization algorithms in terms of execution time, and RAM/flash memory occupancy on the Arduino DUE platform for both values of maximum battery capacity B_{max}, i.e. 1000 and 2000 mAh. As shown, the task stabilization algorithm does not increase the RAM occupation and minimally increases the flash memory of the Arduino DUE. Note that the occupied space of RAM/flash memory by the algorithms w.r.t. the total space of Arduino DUE is approximately 55% and 8%, respectively. Note, however, that this space includes the code to generate the tasks' costs and quality (which is something that can be optimized in a real deployment) and the code to simulate the solar panel production (that in a real deployment is not necessary, since in this case the device monitors in real time the status of the battery/solar panel). This means that the algorithm can really be executed on resource-constrained platforms. The average time to calculate the schedule with this algorithm is longer than the time required with the IoT algorithm, approximately 77% higher; however, the absolute timing is in the order of a few seconds to compute a schedule for the entire day; this cost does not represent a problem in any practical scenario. Note also that, for $B_{\text{max}} = 1000$ mAh the average execution time and the RAM usage scaled down of around 50% with respect to the configuration with $B_{\text{max}} = 2000$ mAh. This is because in the algorithm B_{max} determines the number of iterations of the loop `for` in line 6 of Listing 1.

Table 5 – Comparison of the execution times, RAM and flash memory occupancy by the IoT/stabilization algorithms on the Arduino DUE platform.

Algorithm	**IoT**		**Stabilization**	
B_{max} **(mAh)**	**1000**	**2000**	**1000**	**2000**
Execution time (ms)	432	1011	865	1792
RAM usage (Bytes)	26364	54364	26364	54364
Flash usage (Bytes)	41548	41548	41724	41724

We compute the quality provided by the IoT algorithm and the task stabilization algorithm for the different values of γ and for the two configurations of the battery. The results are shown in Fig. 1 and Fig. 2 with $B_{\text{max}} = 2000$ mAh and $B_{\text{max}} = 1000$ mAh, respectively, which show two box diagrams that represent the overall maximum/minimum/avg quality for the overproduction (left) and underproduction scenario (right), the quality is presented as a percentage of the maximum quality achievable, i.e. if we schedule the task with higher quality in all slots. Note that each experiment was executed a number of $n = 10$ times, where in each iteration the energy production panel was randomly and slightly updated. This update aims to reflect the variations on the estimated hourly energy production that may occur in the real world, so we first calculate the increase/decrease on the estimated value and then distribute it evenly over the number of slots per hour. As shown, each box diagram shows at the first value of axis x the maximum/minimum/average qualities provided by the IoT algorithm and, at the rest of values of axis x, the maximum/minimum/average qualities provided by the task stabilization algorithm for the values of γ evaluated.

From the results, we observe that the higher the energy production the higher the quality, so we obtain significantly larger qualities in the overproduction scenario. Additionally, we observe that the qualities achieved by means of task stabilization for every values of γ are very close to the ones achieved by the IoT algorithm, which provides the optimal quality, so the task stabilization algorithm does not imply a significant reduction in quality. We also observe how the battery size impacts the quality achieved by the stabilization algorithm. As the comparison of the two graphs on the right-hand side of figures 1 and 2 shows, no noticeable reduction in quality is observed in the underproduction scenario. However, the comparison of the two plots on the left-hand side of figures 1 and 2) shows that, for the overproduction scenario, the reduction in quality obtained when halving the battery capacity is very significant, approximately 10%, meaning that a smaller battery can only grow to its maximum capacity and therefore will not take advantage of the increased energy production that allows it to plan more costly and higher quality tasks.

Next, we focus on evaluating the strategy developed for task stabilization by computing the number of leaps of a certain schedule S, provided by both the IoT and task stabilization algorithm. We are particularly interested in determining how the number of leaps may be reduced for the different values of γ and B_{max} considered, and compare these against the number of leaps required by the IoT algorithm to compute the optimal schedule. To do that, we take the results of the overproduction scenario and we compute the metric `sum` for each schedule computed for each one of the n iterations performed for each experiment, both for the IoT algorithm and for the task

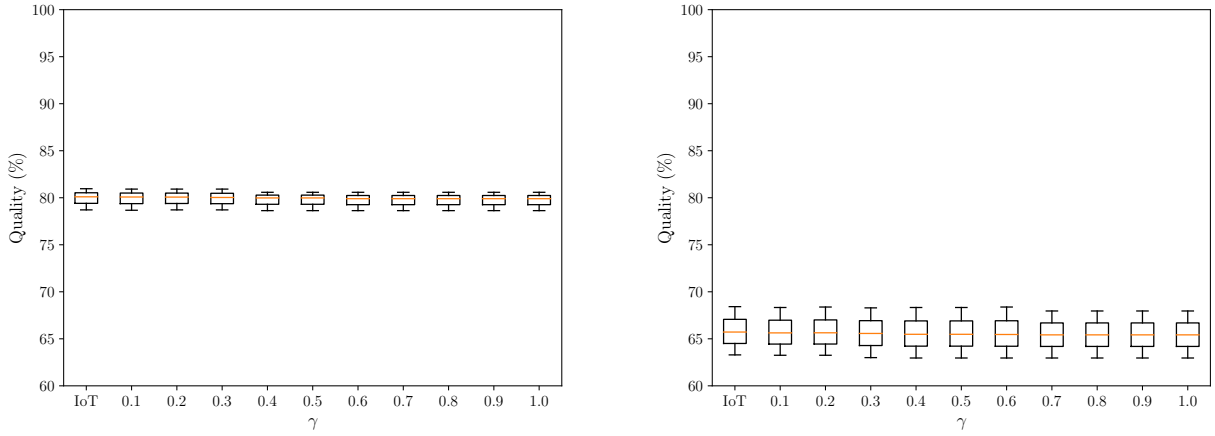

Fig. 1 – Overall maximum/minimum/average quality provided for the overproduction scenario (left) and underproduction scenario (right) with B_{max}=2000 mAh.

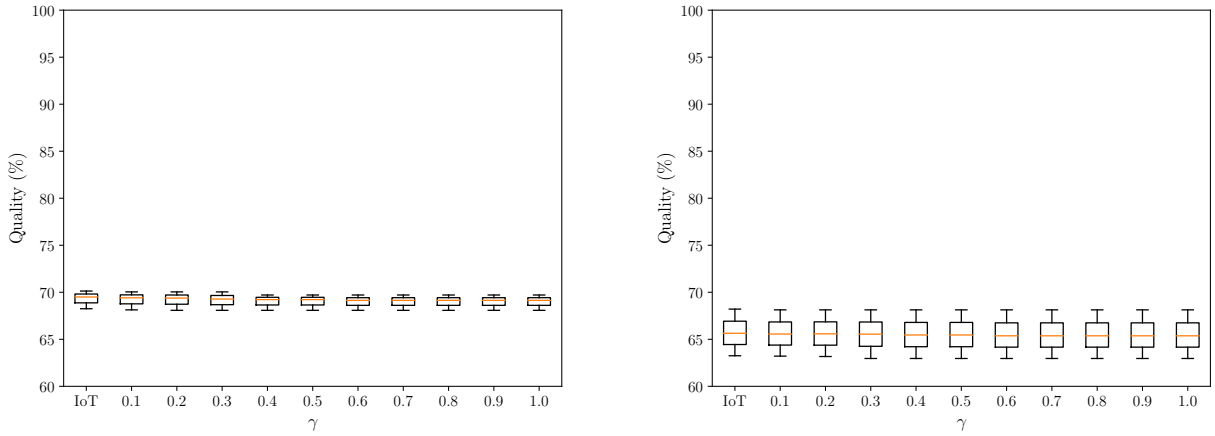

Fig. 2 – Overall maximum/minimum/average quality provided for the overproduction scenario (left) and underproduction scenario (right) with B_{max}=1000 mAh.

stabilization algorithm.

Fig. 3 shows the maximum, minimum, and average sum of leaps for each case, when $B_{\text{max}} = 2000$ (left) and $B_{\text{max}} = 1000$ (right). As expected, regardless B_{max}, as γ increases the sum of leaps is reduced, until a certain value of γ ($\gamma = 0.6$ for $B_{\text{max}} = 2000$ and $\gamma = 0.3$ for $B_{\text{max}} = 1000$), when the sum of leaps achieves the minimum and keeps constant. If we consider the worst case, i.e., the maximum sum of leaps required to compute a schedule (the two blue lines), 9 and 23 leaps are needed with the IoT algorithm with $B_{\text{max}} = 2000$ and $B_{\text{max}} = 1000$, respectively, whilst with the task stabilization algorithm we need 7 leaps for values of $\gamma \geq 0.6$ and $B_{\text{max}} = 2000$ and 11 leaps for values of $\gamma \geq 0.3$ and $B_{\text{max}} = 1000$. Consequently, as battery capacity decreases, the number of leaps increases in both algorithms regardless of the value of γ, due to the loss of granularity resulting from having fewer battery sampling levels, which can lead to scheduling tasks with greater distance in terms of cost and quality.

Finally, we analyze how stable is the schedule given by the task stabilization algorithm. To do that, we take the worst-case schedule from Fig. 3 in terms of number of leaps, i.e. the blue line in the plot of the left side that represents 9 leaps (IoT) and 7 leaps (stabilization with $\gamma \geq 0.6$) with $B_{\text{max}} = 2000$ and the blue line in the plot of the right side with 23 leaps (IoT) and 11 leaps (stabilization with $\gamma \geq 0.3$) with $B_{\text{max}} = 1000$. The output of the execution for both algorithms (IoT and task stabilization with $\gamma = 0.1, 0.3$ and 0.6) provides the schedule shown in Table 6, the vector of leaps shown in Table 7 and the quality shown in Table 8.

As shown in Fig. 4 on the left-hand side, IoT and task stabilization algorithms with $\gamma = 0.1$ produce exactly the same maximum quality $Q = 80.79\%$, with a different schedule but with the same sum of leaps (9), as shown in the figure on the right-hand side. For values of $\gamma \geq 0.3$ we obtain a quality similar but more stable schedules, i.e. a lower sum of leaps and possibly a lower maximum value of gaps,

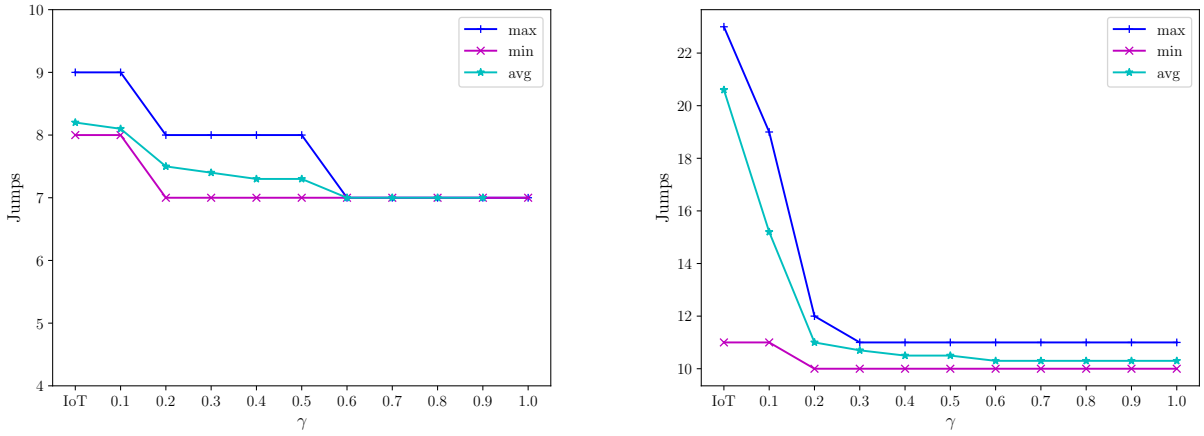

Fig. 3 – Maximum/minimum/average number of leaps for the schedules provided by the IoT algorithm, and by the task stabilization algorithm for all values of γ considered for $B_{\max}$=2000 (left) and for $B_{\max}$=1000 (right)

Table 6 – Scheduling provided for each slot by the IoT algorithm (first row) and for the task stabilization algorithm with values of γ=0.1, 0.3 and 0.6 for $B_{\max}$=2000, and values of γ=0.1 and 0.3 for $B_{\max}$=1000

$B_{\max}$	**Task scheduled in slot**																							
2000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
IoT	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	7	2	2	2	1	1
γ=0.1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	7	7	5	2	2	1	1
γ=0.3	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	4	4	4	4	2	2	2
γ=0.6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	4	3	3	3	3	3	3
1000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
IoT	9	9	9	7	7	7	7	6	2	10	10	10	10	10	10	10	10	7	6	2	2	2	2	2
γ=0.1	9	8	7	6	4	5	6	7	8	9	10	10	10	10	10	10	10	4	4	4	3	3	2	2
γ=0.3	6	7	7	7	7	7	7	7	7	7	10	10	10	10	10	10	10	4	3	3	3	3	3	3

Table 7 – Leaps in each slot provided by the IoT algorithm (first row) and for the task stabilization algorithm with values of γ=0.1, 0.3 and 0.6 for $B_{\max}$=2000, and values of γ=0.1 and 0.3 for $B_{\max}$=1000

$B_{\max}$	**Leap in slot**																							
2000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
IoT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	0	0	1	0	0
γ=0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	2	3	0	1	0	0
γ=0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	2	0	0	0
γ=0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	1	0	0	0	0	0	0
1000	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
IoT	0	0	2	0	0	0	1	4	8	0	0	0	0	0	0	0	3	1	4	0	0	0	0	0
γ=0.1	1	1	1	2	1	1	1	1	1	1	0	0	0	0	0	0	6	0	0	1	0	1	0	0
γ=0.3	1	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	6	1	0	0	0	0	0	0

as shown in Table 5. Note that more stable schedules do not mean the overall quality is significantly decreased, as shown for $\gamma = 0.3$ and 0.6 in the third and fourth row of Table 8. Note also that a decrease of 2 leaps with $\gamma = 0.6$ represents a 22.2% of reduction. In Fig. 4 we depict the associated schedule (on the left side) and the vector of leaps J (on the right side) to the four experiments shown in Table 6. Fig. 5 shows on the left-hand side the schedule and on the right side the stabilization metrics for the IoT and task stabilization algorithms with $\gamma = 0.1, 0.3$

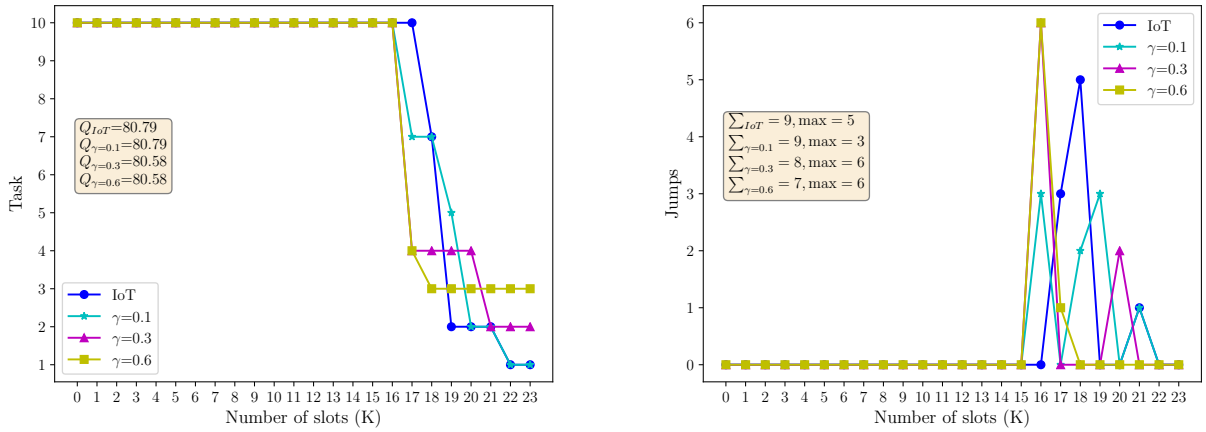

Fig. 4 – Schedule (left) and stabilization metrics (right) obtained by IoT and stabilization algorithm and for $B_{\max}$=2000.

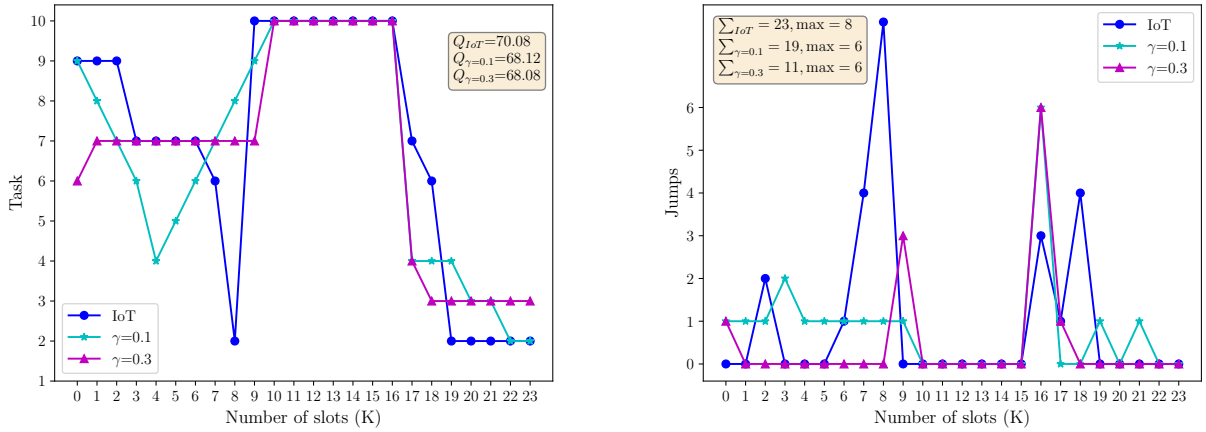

Fig. 5 – Schedule (left) and stabilization metrics (right) obtained by IoT and stabilization algorithms and for $B_{\max}$=1000.

Table 8 – Overall quality (in percentage) and the stabilization metrics (sum, max) provided by the IoT algorithm (first row) and for the task stabilization algorithm with values of γ=0.1, 0.3 and 0.6 for $B_{\max}$=2000 and for $B_{\max}$=1000.

Metrics	**Q (%)**		**(sum,max)**	
$B_{\max}$ **(mAh)**	**1000**	**2000**	**1000**	**2000**
IoT	70.08	80.79	(23,8)	(9,5)
γ=0.1	68.12	80.79	(19,6)	(9,3)
γ=0.3	68.08	80.58	(11,6)	(8,6)
γ=0.6	–	80.58	–	(7,6)

and $B_{\max} = 1000$, where we observe also the same effects, since the increase of γ does not imply the reduction in quality but it does imply more stable schedules, with a lower sum of leaps. We also note that a larger battery capacity significantly impacts both the overall quality obtained, which increases with the capacity, and on the stabilization of the schedule, which is less stable with lower capacities.

7. CONCLUSIONS AND FUTURE WORK

As energy neutrality in energy-constrained IoT devices becomes increasingly important, new approaches are emerging to find different optimization objectives, in addition to maximizing the quality of the tasks scheduled for execution. One of these approaches is to seek task stabilization, in order to avoid fluctuations from the point of view of the application behavior, such as in the sampling and transmission rates that could impact the quality of the data measured by the sensors. This is precisely the objective that we address in this work: an energy-neutral algorithm that gracefully degrades or upgrades the performance of the application and that consequently minimizes the number of leaps between consecutive tasks, i.e. the sum of the differences between the indexes of the scheduled consecutive tasks across the reference time. As far as the authors know, the problem of the stability of the behavior of an IoT device has received no consideration at all in the past works on energy neutrality.

This work extends the original work described in [16] with an implementation and evaluation of two algorithms, the proposed task stabilization algorithm and the quality maximization algorithm described in [5], on a real resource-constrained platform such as the Arduino DUE. The results show that, our approach greatly reduces the leaps among consecutive tasks executed by the device, with no significant reduction in quality as compared with the quality maximization algorithm.

As future work we plan to extend the comparison of this algorithm with others that will arise in future literature, addressing the problem of scheduling with stabilization, and to extend this approach to the concept of statistical energy neutrality [19] and to use other cost-utility patterns to represent different IoT use cases and scenarios, as for instance, the 'utility-like' concave utility function to characterize diminishing returns.

ACKNOWLEDGEMENTS

This paper is partially supported by European Union's Horizon 2020 research and innovation programme under grant agreement no. 857159, project SHAPES (Smart & Healthy Ageing through People Engaging in Supportive Systems), by MCIN/AEI/10.13039/501100011033.under Grant TALENT-BELIEF (PID2020-116417RB-C44) and the Project MIRATAR TED2021-132149B-C41 funded by MCIN/AEI/10.13039/501100011033 and by European Union NextGenerationEU/PRTR. It is also partially supported by the PRA project AUTENS (Sustainable Energy Autarky) of the University of Pisa, and the PON-PNR 2015-2020 project TEBAKA (Territorial Basic Knowledge Aquisition) of the University of Salento.

REFERENCES

[1] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. "Power Management in Energy Harvesting Sensor Networks". In: *ACM Trans. Embed. Comput. Syst.* 6.4 (Sept. 2007), 32–es. ISSN: 1539-9087. DOI: 10.1145/1274858.1274870.

[2] Edoardo Baldini, Stefano Chessa, and Antonio Brogi. "Estimating the Environmental Impact of Green IoT Deployments". In: *Sensors* (2023), pp. 1–29. ISSN: 14248220. DOI: 10.3390/s23031537.

[3] Melisa Kuzman, Xavier del Toro Garcia, Soledad Escolar, Antonio Caruso, Stefano Chessa, and Juan Carlos Lopez. "A Testbed and an Experimental Public Dataset for Energy-Harvested IoT Solutions". In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. IEEE, July 2019. DOI: 10.1109/indin41052.2019.8972219.

[4] Soledad Escolar, Stefano Chessa, and Jesús Carretero. "Quality of service optimization in solar cells-based energy harvesting wireless sensor networks". In: *Energy Efficiency* (2017), pp. 1–27. ISSN: 1570-6478. DOI: 10.1007/s12053-016-9458-3.

[5] Antonio Caruso, Stefano Chessa, Soledad Escolar, Xavier del Toro, and Juan Carlos López. "A Dynamic Programming Algorithm for High-Level Task Scheduling in Energy Harvesting IoT". In: *IEEE Internet of Things Journal* 5.3 (June 2018), pp. 2234–2248. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2828943.

[6] Junlong Zhou, Jin Sun, Peijin Cong, Zhe Liu, Xiumin Zhou, Tongquan Wei, and Shiyan Hu. "Security-Critical Energy-Aware Task Scheduling for Heterogeneous Real-Time MPSoCs in IoT". In: *IEEE Transactions on Services Computing* 13.4 (2020), pp. 745–758. DOI: 10.1109/TSC.2019.2963301.

[7] Kun Cao, Junlong Zhou, Guo Xu, Tongquan Wei, and Shiyan Hu. "Exploring Renewable-Adaptive Computation Offloading for Hierarchical QoS Optimization in Fog Computing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2095–2108. DOI: 10.1109/TCAD.2019.2957374.

[8] Laihyuk Park, Chunghyun Lee, Joongheon Kim, Aziz Mohaisen, and Sungrae Cho. "Two-stage IoT device scheduling with dynamic programming for energy Internet systems". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8782–8791.

[9] Muhammad Moid Sandhu, Sara Khalifa, Raja Jurdak, and Marius Portmann. "Task Scheduling for Simultaneous IoT Sensing and Energy Harvesting: A Survey and Critical Analysis". In: *arXiv: Signal Processing* (2020).

[10] Pierpaolo Loreti, Lorenzo Bracciale, and Giuseppe Bianchi. "StableSENS: Sampling Time Decision Algorithm for IoT Energy Harvesting Devices". In: *IEEE Internet of Things Journal* 6.6 (2019), pp. 9908–9918. DOI: 10.1109/JIOT.2019.2933335.

[11] Soledad Escolar, Stefano Chessa, and Jesús Carretero. "Optimization of Quality of Service in Wireless Sensor Networks Powered by Solar Cells". In: *IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. July 2012, pp. 269–276. DOI: 10.1109/ISPA.2012.43.

[12] Soledad Escolar, Stefano Chessa, and Jesús Carretero. "Energy management of networked, solar cells powered, wireless sensors". In: *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems*. ACM. 2013, pp. 263–266.

[13] Soledad Escolar, Stefano Chessa, and Jesús Carretero. "Energy-neutral networked wireless sensors". In: *Simulation Modelling Practice and Theory* 43 (2014), pp. 1–15. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2014.01.

002. URL: https://www.sciencedirect.com/science/article/pii/S1569190X14000033.

[14] Soledad Escolar, Stefano Chessa, and Jesús Carretero. "Energy management in solar cells powered wireless sensor networks for quality of service optimization". In: *Personal and Ubiquitous Computing* 18.2 (2014), pp. 449–464. DOI: 10.1007/s00779-013-0663-1.

[15] Antonio Caruso, Stefano Chessa, Soledad Escolar, Xavier del Toro, Melisa Kuzman, and Juan C. López. "Experimenting Forecasting Models for Solar Energy Harvesting Devices for Large Smart Cities Deployments". In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. 2019, pp. 1177–1182. DOI: 10.1109/ISCC47284.2019.8969684.

[16] Antonio Caruso, Stefano Chessa, Soledad Escolar, Fernando Rincón, and Juan C. López. "Task Scheduling Stabilization for Solar Energy Harvesting Internet of Things Devices". In: *2022 IEEE Symposium on Computers and Communications (ISCC)*. 2022, pp. 1–6. DOI: 10.1109/ISCC55528.2022.9913061.

[17] M.M. Sandhu, S. Khalifa, R. Jurdak, and M. Portmann. "Task Scheduling for Energy-Harvesting-Based IoT: A Survey and Critical Analysis". In: *IEEE Internet of Things Journal* 8.18 (2021), pp. 13825–13848.

[18] T. Bu, Z. Huang, K. Zhang, Y. Wang, H. Song, J. Zhou, Z. Ren, and S. Liu. "Task scheduling in the internet of things: challenges, solutions, and future trends". In: *Cluster Computing* (2023).

[19] S. Escolar, A. Caruso, S. Chessa, X. d. Toro, F. J. Villanueva, and J. C. López. "Statistical Energy Neutrality in IoT Hybrid Energy-Harvesting Networks". In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018, pp. 00444–00449. DOI: 10.1109/ISCC.2018.8538532.

[20] Elizabeth Liri, K. K. Ramakrishnan, Koushik Kar, Geoff Lyon, and Puneet Sharma. "An Efficient Energy Management Solution for Renewable Energy Based IoT Devices". In: *Proceedings of the 24th International Conference on Distributed Computing and Networking*. ICDCN '23. event-place: Kharagpur, India. New York, NY, USA: Association for Computing Machinery, 2023, pp. 20–27. ISBN: 978-1-4503-9796-4. DOI: 10.1145/3571306.3571387. URL: https://doi.org/10.1145/3571306.3571387.

[21] F.A. Kraemer, D. Palma, A.E. Braten, and D. Ammar. "Operationalizing Solar Energy Predictions for Sustainable, Autonomous IoT Device Management". In: *IEEE Internet of Things Journal* 7.12 (2020), pp. 11803–11814.

[22] T. Sanislav, G.D. Mois, S. Zeadally, and S.C. Folea. "Energy Harvesting Techniques for Internet of Things (IoT)". In: *IEEE Access* 9 (2021), pp. 39530–39549.

AUTHORS

Antonio Caruso received his MS degree ("cum laude") in computer science from the University of Pisa, Italy, and PhD in computer science from the same University. From 2005 he joined the Mathematical and Physics Department of the University of Salento as assistant professor. He received the Innovation Award from Italian-Canada in 2017. He has been member of several program committees of conferences and workshops, and published on international journals and conference proceedings mainly in the area of: mobile distributed systems, internet of things, smart environment, wireless sensor and ad-hoc networks, underwater networks, and distributed algorithms.

Stefano Chessa is full professor at the Department of Computer Science of the University of Pisa. He is member of the Council of the Doctorate in Computer Science and chair of the MSc curricula in Cybersecurity of the University of Pisa. He has worked in several EU projects (SatNex, SMEPP, InterMedia, PERSONA, universAAL, Engaged, TEACHING), and he has been the scientific leader (for the University of Pisa) of the EU projects RUBICON and DOREMI. He is co-author of almost 200 publications appeared on international, peer-reviewed journals, conferences, and books chapters. His research interests are in the fields of smart environments, Internet of Things, pervasive computing and in their applications to e-health, ambient assisted living, crowdsensing and participatory sensing.

Soledad Escolar holds a Ph.D. in computer science and technology from the University Carlos III de Madrid in 2010 with honors. In September 2015, she joined the University of Castilla-La Mancha (UCLM) as a post-doctoral researcher and in 2020 she took possession of the position of associate professor at the same university. Since 2021, she has been appointed as the deputy director of the Academic Planning and Smart ESI at the School of Computer Science. Her research career has focused mainly on the study of a wide range

of problems related to distributed systems, more specifically, sensor networks and their evolution towards the Internet of Things, specifically application portability in heterogeneous environments, communication protocols, synchronization, mobility, models, and energy efficiency management algorithms in energy-harvesting devices.

Fernando Rincón obtained a PhD in computer engineering from the University of Castilla-La Mancha in May 2003, after graduating in computer science from the Autonomous University of Barcelona in September 1993. Currently, he serves as an assistant professor at the University of Castilla-La Mancha. His scientific contribution includes 20 publications in international impact journals, 8 book chapters, two patents, and over 50 contributions to national and international conferences. He has extensive experience in both publicly funded research projects and has participated in over 20 contracts with leading Spanish companies. He served as the director of the Department of Information Technology and Systems from 2010 to April 2017. His current lines of research include edge computing, electronic design automation, and dynamically reconfigurable systems based on FPGA devices.

Juan Carlos López received his MS and Ph.D. degrees in telecommunication (electrical) engineering from the Technical University of Madrid in 1985 and 1989, respectively. From September 1990 to August 1992, he was a visiting scientist in the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA (USA). His research activities center on embedded system design, distributed computing and advanced communication services. From 1989 to 1999, he was an associate professor at the Department of Electrical Engineering at the Technical University of Madrid. In 1999, Prof. López joined the University of Castilla-La Mancha as a professor of computer architecture, where he served as dean of the School of Computer Science from 2000 to 2008. Since 2006, he is the director of the Indra Chair at this university. He is and has been member of different panels of the Spanish National Science Foundation and the Spanish Ministry of Education and Science, regarding information technologies research programs. He is member of IEEE and ACM.