A Dynamic Programming Algorithm for High-Level Task Scheduling in Energy Harvesting IoT

Antonio Caruso, Stefano Chessa, Soledad Escolar, Xavier del Toro and Juan Carlos López

***Abstract*—Outdoor IoT applications usually exploit energy harvesting systems to guarantee virtually uninterrupted operations. However, the use of energy harvesting poses issues concerning the optimization of the utility of the application while guaranteeing energy neutrality of the devices. In this context, we propose a new dynamic programming algorithm for the optimization of the scheduling of the tasks in IoT devices that harvest energy by means of a solar panel. We show that the problem is NP-Hard and that the algorithm finds the optimum solution in a pseudo-polynomial time. Furthermore, we show that the algorithm can be executed with a small overhead on three popular IoT platforms (namely TMote, Raspberry PI and Arduino) and, by simulation, we show the behavior of the algorithm with different settings and at different conditions of energy production.**

***Index Terms*—Energy Harvesting Sensor Networks, Scheduling, Dynamic Programming.**

I. Introduction

Several Internet of Things (IoT) applications make use of wireless sensors deployed outdoor that harvest energy by means of solar panels and rechargeable batteries to sustain their operations [1]. Such sensors monitor the surrounding environment by means of on-board transducers, they process and store locally the sensed data and they transmit the results of their elaboration through the Internet to remote users. With the growing capabilities of sensors (in terms of processing, storage and communication), due to the development of the technological platforms on which they are built, it is now possible to embed in the sensors relatively complex data processing functions even based on machine learning technologies [2]. Furthermore, it is possible to make the entire software of the sensors flexible enough to meet the constraints coming from their energy production subsystem, and to give them a virtually unlimited lifetime.

In the past years, research on energy harvesting sensors has focused on developing strategies aimed at making them energy neutral, i.e. such that, within a given time frame, their energy consumption equals their energy production, thus leaving the sensor with the same amount of energy it had at the beginning of the time frame. This approach is particularly effective when applied to environmental monitoring sensors that harvest solar energy, because this energy is, to some extent, predictable and this fact fits well the typical operational mode of the sensors. In fact, they usually sample environmental parameters with a given duty cycle and hence have a very well predictable energy consumption, that keeps the same pattern over time. Thus, predicting the energy production allows to modulate the duty cycle of the sensors accordingly and to keep the sensors energy neutral [3]. Other works also observed that it is possible to modulate the energy consumption of the sensors acting on other parameters than just the duty cycle. For example, a sensor may scale down the frequency of its processor to consume less (although with a reduced performance) when the energy production is lower [4], or optimize the radio communications [5], or it may switch to a less consuming transducer (possibly with a degraded resolution or accuracy). A recent survey of power management technique in energy-harvesting sensor networks is published in [6].

Most of these solutions operate at low level with a fine tuning of the duty cycle and assume a strong integration of the software components of the sensors application as in TinyOS [7], for example. On the other hand, recent trends in IoT are pushing the technology towards a (relatively) more capable platforms, like the Raspberry PI, that run traditional operating systems like Linux or WindowsIoT. In these cases, the IoT applications can be modeled by tasks that are scheduled by the operating system. Following this trend, in some of our recent works [8], [9], [10], [11] we modeled all these possible strategies to modulate the energy consumption of a sensor by assuming that the software of the sensor may be structured in several, alternative tasks, each characterized by an energy consumption and a quality of service (QoS). The idea is in part similar to the *Multi-version Tasks System* used in [12], i.e. all the tasks respond to the same monitoring problem, but with a different performance (and thus a different QoS), either in terms of quality of the data produced (for example because tasks may use different transducers with different specifications in terms of resolution, accuracy etc.) and of quantity of the data (for example by modulating the sampling rate). In those works we proposed a greedy heuristic to choose the scheduling of the tasks to guarantee the energy neutrality of the sensor while attempting to maximize the QoS.

In this work, we reconsider the algorithms used to solve the problem of finding an optimal, energy neutral scheduling of the tasks (which is NP-hard), and we propose a new algorithm based on dynamic programming. Differently from our previous works, which adopt greedy algorithms (and thus, in general, they do not find the optimal solution), this new approach based on dynamic programming actually finds

A. Caruso is with the Dept. of Mathematics and Physics *Ennio de Giorgi*, University of Salento, Collegio Fiorini, 73100, Lecce, Italy.

S. Chessa is with the Department of Computer Science, Largo Pontecorvo 3, 56127, Pisa, Italy.

S. Escolar is with the Institute of Technology and Information Systems, University of Castilla-La Mancha, 13051, Ciudad Real, Spain.

X. del Toro is with the Energy Research and Industrial Applications Institute, University of Castilla-La Mancha, 13051, Ciudad Real, Spain.

J. C. López is with the School of Computing Science, University of Castilla-La Mancha, Ciudad Real, Spain

the scheduling of the tasks that maximizes the QoS within the constraint of energy neutrality. This result is possible because our dynamic programming algorithm explores the entire space of possible solutions, but it consequently has a pseudo-polynomial complexity. However, we show that in realistic platforms and conditions, where the space of possible solutions is restricted because it depends on the number of slots and on the number of battery levels that are both limited, our algorithm results feasible and has a small overhead even when implemented for low-power platforms.

The remainder of this paper is organized as follows. After reviewing the related work in Section II, we present in Section III the design of a real-world sensor node platform connected to a solar cell-based energy harvesting system and provide its energy production model as well as the energy consumption model of the applications. Section IV describes the system model that supports the scheduling of applications and formulates the QoS energy-neutral optimization problem. In Section V we present an algorithm aimed at optimally solving the scheduling problem on the basis of the solar energy prediction model proposed and in Section VI we provide the simulation results. Finally, Section VII discusses the conclusions and further research.

II. Related Works

The idea of *energy harvesting* from the environment is studied in several papers [13], [14], [15], [16] and more recently in [17], [6], [18], [19]. A good survey on scheduling for battery powered sensor nodes is in [20].

In general, energy harvesting in IoT devices can be achieved by different energy sources, which are classified in [3] based on their controllability and/or predictability. An example of controllable source is that based on radio-frequency energy transmission, where the energy harvested by a node is that emitted by other special nodes in the same environment. An interesting issue that rises in devices using this kind of energy harvesting is that of relay selection [21], [22], [23], [24], [25]. This problem is a special case of the more general problem of selecting a source-to-sink path in a multihop network of IoT nodes, and it consists in finding the next hop of a path that optimizes some parameter like the throughput or the energy efficiency.

Our work instead, focuses on uncontrollable but predictable energy sources, namely on solar power, and on the problem of managing the nodes activities *tasks*, and thus on their energy expenditure, so that the nodes' lifetime can be extended indefinitely. Most works on energy management deal with the problem of *scheduling* tasks: A task takes low-level decisions like when to sense the environment, the level of duty-cycle of the radio [3], [26] or on the power used in sending packets [27], [28]. In [29] a control-theoretic model selects the best *sensing rate* according to the estimated future production of the panel.

Kansal et al. [30], [31], [32], [3], [33] model the intuitive notion of *Energy Neutrality*. In [3] the problem of selecting the optimal duty-cycle of a node while maintaining an energy-neutral schedule is considered, and formalized using a linear programming model. The model is solved using a heuristic based on a greedy approach, i.e. the finite horizon (a day) is divided in slots and, according to the knowledge of the expected future harvested energy, slots are classified as dark or sunny. The initial duty-cycle assignment is simple: the maximum possible duty-cycle in sunny slots and the minimum in dark slots. Then a greedy-style adaptation phase changes the initial assignment based on the difference between the expected and the real production of the panel, by increasing or decreasing appropriately the value of the duty-cycle while maintaining a feasible (i.e. energy neutral) schedule. Our model is an extension of the Kansal model, since it uses a more abstract quality associated to each schedulable task. The resulting model is more general and complex (solving it optimally is Np-Hard), but we do not use an heuristic or a greedy algorithm to solve it, instead we propose a dynamic programming algorithm that, even if pseudo-polynomial, can compute an optimal solution in all practical IoT scenarios.

An alternative scheduling algorithm for rechargeable sensor nodes and with tasks deadlines is LSA (Lazy Scheduling Algorithm) [34]. LSA introduces the concept of energy variability characterization curve (EVCC), which captures the dynamics of the energy source, and an offline schedulability test. Given the EVCC, the battery capacity and the power requirement of tasks it determines whether all the deadlines can be met or not. The paper [35] from the same authors presents an on-line scheduling algorithm that dynamically assigns power to arriving tasks. The performance of the scheduler is evaluated with simulation and compared, similarly to this paper, to a greedy approach as the base case. The task model with deadlines used in LSA is more appropriate to a scenario with real-time scheduling, with a scheduling algorithm that act frequently. In our work instead, we follow a different approach to study and design a high-level scheduler that acts much less frequently (usually at least every 5 or 10 minutes) and that is based on a different task model without deadlines. From the point of view of the scheduling algorithm, the LSA scheduler is based on a dynamic programming approach as in our proposed scheduling algorithm. However, the LSA model of utility of tasks is significantly different than ours. In particular, it assumes a single task with continuously adaptable service levels and with concave reward functions (this assumption enables an analytic study of the algorithm). In our model instead, we assume several tasks with discrete levels of allowable utility, which provides a simpler abstraction for a programmer since it needs to specify just a discrete value of utility for each task implemented.

Following the concept of energy neutrality, in one of our first works on the problem of efficient tasks scheduling in energy harvesting sensors [10], we introduced a multiversion task programming model, with each task associated with a quality metric measured in terms of its maximum sampling frequency. The work studies how to maximize the QoS, i.e. the extent of the period in which the sensor operates at the user's desired sampling frequency. The paper presents a scheduling algorithm that selects a version of the application to be executed in each duty cycle, according to the QoS metric, the estimation of the battery level, and the expected energy production in each duty cycle. Based on this work, an extended model is presented in [11], considering for each task its execution time, its cost

and its duration. The scheduler produces an energy neutral schedule, built from a first initial assignment of tasks to slots. After this first assignment, the algorithm, which is based on a greedy approach, considers the actual energy production to *upgrade* or *downgrade* the schedule.

The works [9], [36] extend the above model and scheduling algorithm to deal with a networked scenario where all the sensors in the network are powered by solar panels (the first paper considers only a sender and a receiver, while the second considers a star connected network with several slave nodes and a sink). These papers introduce a protocol that exchanges information about the battery levels of the nodes, and the scheduling algorithm, which is a variant of the *Upgrade/Downgrade* algorithm, uses this information to schedule the tasks in the nodes.

Compared to the above work, this paper considers a more general task model with different, alternative tasks implementing a single application and that abstracts from the task execution time and duration. In this model we focus on the problem of finding the optimum tasks scheduling, which we prove to be Np-Hard, and we propose a scheduling algorithm that is provably optimal and that can be efficiently executed in low-power nodes in realistic conditions (despite the problem being Np-Hard), while the *Upgrade/Downgrade* algorithms used in the above works are suboptimal and, in general, do not find the optimum tasks assignment.

III. Solar Production and Consumption Models

In solar energy harvesting, an IoT node is equipped with solar panels able to harvest the energy from the solar light, and convert it into electricity to recharge their batteries, with the goal of prolonging their lifetimes. The amount of energy that a solar cell may transform into electricity by means of the photovoltaic effect depends on several factors, mainly related to the manufacturing technology and the geographic location where it is installed. Thus, a solar panel operates by taking as input a certain solar irradiance D (W/m^2), which is the density of incident power on the surface of the solar cell and provides a maximum power output $P_{out} = D \times \eta \times S$, where η is the efficiency (energy conversion efficiency percentage absorbed by the cell under standard conditions) and S is the surface of the solar panel. Given a certain P_{out}, we compute the energy produced, E, as the time integral of it. Since the energy produced by the solar panel is stored in a battery in the IoT applications under consideration, it is common to use the state of charge of the battery, C (typically measured in mAh), that can be derived from the energy produced by the panel as $C = \frac{E}{V_{mp}}$, where V_{mp} is the maximum power point voltage in which the solar module operates. Note that the solar charger and battery losses are neglected in the previous equation. In the next subsections we detail the hourly solar production model and the energy consumption model of an IoT node.

A. Solar Energy Production Model

We have proposed in [36] an hourly solar energy generation model that enables to compute the hourly irradiance $D(h)$, with $h \in [0, 23]$, in any geographic location on the Earth. This model is based on distributing the irradiance that is accumulated during a day among 24 hours of the day. The output of the model provides a curve of solar production similar to a parabola, with null production at the beginning of the day; approximately at the sunrise, the solar production starts to grow up to reach its maximum at noon and then starts to decrease until the sunset is reached, when the production becomes 0 again. The problem is that such a parabola depends on the current climatic conditions and the solar zenith angle (Θ_Z), i.e. the angle drawn by the Sun and the vertical axis of the Earth surface. In order to compute Θ_Z, we take into account the geographic position where the solar module is deployed (latitude), the solar declination angle, and the hour of the day. Knowing the zenith angle and the daily irradiance, which is measured in some specific location (for instance, obtained from the NASA program RETScreen [37]), we proceed computing the hourly irradiance $D(h)$ as $D \cos \Theta_Z$ where Θ_Z is the zenith angle in any $h \in [0, 23]$. After computing $D(h)$ and by assuming the specifications of each solar panel manufacturer, we can calculate what is the hourly energy $E(h)$ delivered by any solar module. To this purpose, we have used the KL-SUN3W solar module [38] as energy harvester that, according to its manufacturer, delivers a maximum efficiency $\eta = 12.8\%$ and supplies a maximum output power of 3 W (voltage at maximum power $V_{mp} = 5.82\ V$; current at maximum power $I_{mp} = 0.52\ A$; short-circuit current $I_{sc} = 0.55\ A$), and whose active surface is $0.0168\ m^2$. Therefore, the hourly charge that it provides is $C(h) = \frac{D(h) \times \eta \times S}{V_{mp}}$, $h \in [0, 23]$. We use this module because it can be connected to different sensors and it is available in our laboratories. However, note that any other module could be used instead.

B. Energy Consumption Model

A battery is discharged at a rate that depends on the application being executed, specifically, on the currents of the hardware components required (e.g. radio, transceivers, microcontroller) and the time being used. Thus, the time to deplete a battery as consequence of the execution of an application is $\frac{B_{\max}}{I}$, where $B_{\max}$ is the battery capacity (in Ah) and I is the sum of the current consumptions of the components involved in the execution (in A) in an instant of time. Analogously, the energy that is produced by the energy harvester is destined to recharge the battery of the sensor. The amount of hourly solar charge $C(h)$, expressed also in Ah, is accumulated in the battery up to achieve its maximum capacity $B_{\max}$, which cannot be exceeded.

IV. IoT Programming Model

A standard IoT device comprises a microcontroller, a volatile memory for data and code, a radio for wireless communication, a set of environmental sensors and/or actuators and in the case of more complex devices the possibility to access external storage (i.e. sdcards or other non-volatile storage). The node is powered by a battery or supercapacitor of finite capacity, moreover we also assume energy harvesting, hence the node is equipped with a renewal energy harvesting unit that extracts power in some way from the environment, for example a solar

panel. The battery is characterized by an initial level of charge B_1 and a minimum and maximum level $B_{\min}$ and $B_{\max}$ where the first is the minimum level that could operate the node, while the latter is the maximum capacity of the battery.

A. Tasks in IoT Programming

The application executed by an IoT node can be usually specified as a finite automaton that executes predefined operations. For example, if the node is a simple sensor, the application may repeatedly execute the operations of data sampling, processing, storage and transmission. The way in which an application is implemented depends on the underlying platform and operating system: in a TinyOS-class sensor node an operation can express an asynchronous operation occurring inside the runtime; while in a complex node like a Raspberry PI the same concept can be mapped to a kernel thread in Linux.

On the other hand, an application may have several alternative implementations, which may differ in the quality and kind of specific activities performed. For example, an application that has to sense and send sensed data to a base station may be implemented just with sense and transmit operations, or it may also store sensed data in order to improve the reliability of communications. In this case, the second implementation may prove to be more energy-expensive but it would provide a better service. Similarly, a sensing operation may use a high-resolution, high-frequency and energy-costly transducer, or it may use a low-resolution, low-frequency transducer that however consumes less energy. This suggests that alternative implementations of an application (hereafter called *tasks*) can be used to modulate the energy load of a device, so to meet the constraint of energy neutrality in energy harvesting devices.

Following this observation, we consider an application that admits n alternative implementations (tasks) $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, each characterized by a different quality q_i, i.e. the utility that a node gains when task t_i is executed, and an energy cost c_i that measures the overall power consumption of the task. Note that this concept of tasks may map over a Linux thread in a Raspberry PI platform, while it is slightly different than the concept of task in TinyOS or from the concept of tasks adopted by other works, such as [8], [9], [36], which instead define a task as a component of an application.

Given an application and its corresponding set $\mathcal{T}$ of n alternative tasks, we study the problem of scheduling from a high-level perspective. In particular, we assume that a node contains the code implementing all the tasks in $\mathcal{T}$, and that, depending on the energy budget available, the scheduler of the node can decide at any time to switch from one task to the other.

We assume that the scheduler operates by dividing the time in discrete time slots and by assigning a single task to each slot. Considering that the system under consideration is an energy harvesting node powered with a solar cell, which have a cycle of production of 24 hours, it is natural to consider a time frame T of 24 hours divided in K slots of duration $\Delta T = \lceil T/K \rceil$. In this condition the scheduler can be executed once per day to assign the tasks to the slots based on the expected energy production of the solar cell, on the expected residual energy charge in the battery and on the energy costs of the available tasks, with the objective of maximizing the overall quality of the tasks executed and with the constraint of keeping the node energy neutral. Note that, if the conditions of the production and of the battery charge change in the course of the day, the scheduler can be executed again to update, for the remaining part of the day, the task scheduling defined in its previous execution.

Concerning the quality of a task, we note that it is an application specific measure, while the cost of the task is a low-level measure that depends also on the choice of ΔT. This abstract view captures different low-level operations used to reduce the power consumption of a task. In [3] the utility is considered as a linear function of the node duty cycle, and the authors proceed in optimizing the duty cycle to maximize the utility, with the constraint that the obtained schedule must be energy neutral. In this work instead, we consider the utility of a task as a more abstract measure since duty cycling is just a way to optimize the node energy consumption together with other possibilities.

Finally, note that in our model we have deliberately avoided the definition of the specific functionality for each task executed on the IoT platforms. Our task model is generic enough to represent any IoT application with a certain cost and quality. Note also that both cost and quality are directly related to the functionality, i.e., the larger complexity degree of the task, the larger quality and, subsequently, the higher cost. Consider for instance a specific IoT application composed of $n = 4$ tasks, that monitors the environmental temperature and humidity of a field. This means that there exist $n = 4$ alternative implementations of the same application. Each task $t_0 \ldots t_3$ performs the same functionality but with a certain quality q_i and with a certain energy cost c_i. For example, t_0 could implement the most basic functionality, by sampling both sensors at a slow rate (say every 10 minutes) and transmitting the aggregated sampled data once per day; t_1 may sample the sensors and transmit with a higher frequency, for instance, each minute; t_2 could also store the set of sensed data into the flash memory of the IoT device for logging purposes. Finally, t_3 could also use a more sophisticated version of the humidity and temperature sensors with an accuracy meaningfully higher than the basic sensors used by $t_0 \ldots t_2$. In this example, it would be $c_0 \leq c_1 \leq c_2 \leq c_3$ and $q_0 \leq q_1 \leq q_2 \leq q_3$.

B. An ILP Model for Energy Neutral Task Scheduling

Considering the K slots of time, the energy production captured by the energy harvesting unit is represented by the values e_i with $i \in [1, K]$, and assumed to be constant in a slot. The battery is represented with a set of battery levels, from $B_{\min}$ to $B_{\max}$. We denote with B_i the level of the battery *at the beginning* of slot i. In this context, our problem is to find an optimal assignment of tasks to the different slots from 1 to K such that it is feasible, i.e. the battery level never goes below $B_{\min}$, and the overall schedule is energy neutral, i.e. if the node starts with a level of battery of $B_1 > B_{\min}$, at the end of the time window (at time $(K+1)\Delta T$), its residual energy must be at least B_1. This problem can be formulated

as an integer programming problem as follows. Let x_{ij} be Boolean variables, such that $x_{ij} = 1$ iff task j is selected to be scheduled in slot i. The problem can be stated as:

Problem 1 (*Max Energy Neutral Plan Scheduling*)**.**

$$\text{maximize} \quad z = \sum_{i=1}^{K}\sum_{j=1}^{n} x_{ij} q_j \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \tag{2}$$

$$B_{i+1} = \min\Bigg\{B_{\max}, B_i + \eta[e_i - \sum_{j=1}^{n} x_{ij}c_j]^+ - \\ -[\sum_{j=1}^{n} x_{ij}c_j - e_i]^+\Bigg\} \forall i \in [1,K] \tag{3}$$

$$B_{\min} \leq B_i \forall i \in [1,K] \tag{4}$$

$$B_1 \leq B_{K+1} \tag{5}$$

$$x_{ij} \in \{0,1\} \forall i \in [1,K], j \in [1,n] \tag{6}$$

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

The objective function (1) is the sum of the quality of the tasks selected in all slots, the constraint in (2) implies that only one task per slot can be scheduled; Equation 3 [3] gives the battery level at the end of the slot as a function of the battery level at the beginning, the scheduled task and the energy produced by the panel. Note that the new battery level is limited from the above by $B_{\max}$, hence if the energy production for the given slot is greater than the energy cost of the task (c_j) we increase the level of the battery by $\eta(e_i - c_j)$; otherwise we consume all energy produced in that slot and decrease the residual value $(c_j - e_i)$. Note that this equation, together with Inequality 4, guarantees that the battery level cannot go below $B_{\min}$. Inequality 5 is the energy neutrality constraint, it requires that the value of B_{K+1} (i.e. the battery level at the end of our time window of K slots) cannot be lower than the initial value B_1.

This model could be considered an extension of the model of Kansal [32], [3], but it is more complex. The formulation of Kansal finds the optimal duty cycle, and expresses the energy cost and quality directly as a function of the level of duty cycle used in a given slot. This direct formulation has pros and cons: on one hand, it directly finds a low-level parameter of the system (duty cycle); on the other hand, it assumes that the energy cost is simply a constant P_c times the level of duty cycle used, this is less general than our model. In fact, we do not have a single task of which we optimize its duty cycle, but we schedule different tasks, and the energy consumption of each task can be different. The problem of Kansal can be solved with a greedy approach, which starts from the observation that the slots can be partitioned in two sets: the ones with more production than consumption and the others, and it assigns a maximum and minimum duty cycle to the two sets, respectively, then it reassigns the excess of energy of the sun slots to the dark slots with a greedy approach. Instead we prove now that our problem is computationally hard, and no polynomial greedy algorithm can be used to solve it exactly. Note that a proof of this theorem has already been presented in [36], but here we provide a proof based on a different (in some way easier) reduction.

Theorem 1. *The Problem Maximal Energy Neutral Task Scheduling is NP-Hard.*

Proof. We prove the NP-Hardness of Problem 1 by showing that its decision version is NP-complete. Introducing a positive constant Q, the decision version of Problem 1 asks for the existence of a feasible solution with quality at least Q. Since any feasible solution must be energy balanced ($B_{K+1} \geq B_1$), assuming $B_{\max} = \infty$, it must consume at most the energy produced by the harvesting source, thus if we sum over all constraints in (3) we obtain that:

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} e_i$$

This inequality suggests to start the reduction using a version of Knapsack. Since in our problem tasks can be used more than one time, we start the reduction from *Unbounded Knapsack* (UBK) [39]. The decision version of UBK is stated as: given a class of items $S = \{s_1, s_2, \ldots, s_n\}$, with an unbounded number of items for each class, and for each item a value $v_i > 0$ and weight $w_i > 0$, is there a subset of them with total weight at most W and total value at least Q?

Given an instance of UBK, the only real difficulty to reduce it to our problem is the choice of K, i.e. the number of slots, since we select exactly K tasks, while in UBK there is no constraint on the size of the solution. Note, however, that the maximum number of items is bounded by $k* = W/\min w_i$, let the number of tasks be $n+1$, with $q_i = v_i$, $c_i = w_i \; \forall i = 1, \ldots, n$ (so quality maps to value, and energy cost to weight) and $c_{n+1} = w_{n+1} = 0$, i.e. we add an *empty task* with cost and quality set to zero, this task is eventually selected when the optimal schedule comprise less than $k*$ tasks. The initial and minimum battery levels are set to $B_1 = B_{\min} = 0$ and the production is $e_1 = W$ and $e_i = 0$ for $i > 1$. This instance can be created in polytime, now consider a *Yes* instance of our problem: if we remove from the solution all the occurrences of the empty plan we obtain a set of items with total energy consumption (i.e. weight) at most W and quality (i.e. value) greater than Q so this is a *Yes* instance of Multiple Knapsack, and the reduction is complete.

□

V. A Dynamic Programming Algorithm

We propose a solution to the optimization problem (1) based on dynamic programming. We define the state of the system as a pair of integer values (t, b), and associate to each pair a subproblem defined as the optimal energy balanced schedule of the n tasks into slots in the range $[t, K]$, starting with an energy level of b. Let Q^* be the optimal solution of problem 1: it can be calculated using a function of the state $opt(t, b)$, i.e. $Q^* = opt(1, B_1)$. The function opt can be defined using a formulation as in the following Lemma (note that, for the ease of notation and without loss of generality, we assume that the charging efficiency is $\eta = 1$):

Lemma 1. *The value that solves Problem (1) is given by* $Q^* = opt(1, B_1)$ *with opt defined as the following backward recursive Bellman's equations:*

$$opt(K, b) = \max_{i=1\ldots n} \{ q_i \mid b - B_1 + e_K \geq c_i \} \quad (7)$$

$$opt(t, b) = \max_{i=1\ldots n} \{ q_i + opt(t+1, B'(i)) \mid B'(i) \geq B_{\min} \} \quad (8)$$

$$\text{with: } B'(i) = \min\{B_{\max}, b - c_i + e_t\}$$

Proof. The base case is the subproblem with only one (last) slot to schedule. In this case, we select the best feasible tasks, if b is (by definition) the energy at the beginning of this slot, the overall energy available is $b + e_K$ but since we are in the last slot and the entire schedule must be energy neutral, we must save at least B_1. So we schedule the higher quality task with $c_i \leq b - B_1 + e_K$.

The second definition gives the general form of the dynamic programming recursion: We consider slot t with a level of battery b, let $B'(i) = b + e_t - c_i$ the level of the battery if we schedule task i; if $B'(i) \geq B_{min}$ task i is feasible and the quality of the schedule with i assigned to slot t will be q_i plus the optimum schedule that we obtain (recursively) starting from slot $t+1$ with a battery level of $B'(i)$. In the definition, we consider all possible feasible i and select the best one. So evaluating $opt(1, B_1)$ we obtain the optimal schedule. □

```
1  def solve(Tasks,K,Bmax,Bmin,Bstart,E):
2    opt = schedule = np.zeros((K,Bmax+1))
3    for i in range(K-1,-1,-1):
4      for B in range(Bmax+1):
5        opt[i][B] = schedule[i][B] = 0
6        qmax = -100
7        idmax = -1
8        for t in Tasks:
9          if (i == K-1 and
10             B-t['energy']+E[i] >= Bstart and
11             qmax < t['quality']):
12           opt[i][B] = t['quality']
13           schedule[i][B] = t['id']
14         else:
15           Br = min(B-t['energy']+E[i],Bmax)
16           if (Br >= Bmin):
17             q = opt[i+1][Br]
18             if (q!=0 and q+t['quality']>qmax):
19                qmax = q + t['quality']
20                idmax = t['id']
21       opt[i][B] = qmax
22       schedule[i][B] = idmax
23   return (opt,schedule)
```

Table I
DYNAMIC PROGRAMMING ALGORITHM THAT SOLVES PROBLEM 1 IN PYTHON

The equations in Lemma 1 define a recursive function that solves Problem 1 but with an exponential amount of work. A dynamic programming algorithm is presented in Table I with pseudo-polynomial complexity.

A. Complexity Analysis

The code in Table I implements a dynamic programming solution, using a classic tabular approach, i.e. a matrix of $K \times (B_{\max} + 1)$ is filled starting from the base case in row $K - 1$[1]. Note that the algorithm starts from the last slot (K), in line 9, were it selects the best task among those that give an energy balanced budget (first condition). The recursive case is done in the `else`: if the algorithm schedules a task t in slot $k < K - 1$ then the level of battery after the execution Br is computed in line 14; if this value is greater than $B_{\min}$ the algorithm lookup (in line 16) the value of the optimal solutions computed from slot $k + 1$ with that level of energy, and then it computes the maximum among all feasible plan. The computation cost is $O(KB_{\max})$, that is clearly pseudo-polynomial in the input, the space complexity is of the same order, but it can be reduced to $O(B_{\max})$ since the value of each row of the matrix is computed using only the values of one row above. For a practical application of this algorithm we observe that in most common platforms, the range of values of the battery is constant, i.e. the value $B_{\max}$ does not increase in different runs of the algorithm, so we can consider the algorithm as having a polynomial time complexity in all practical applications.

VI. EVALUATION

We have evaluated the optimization algorithm described in Section V by means of three simulations done on several IoT platforms, whose operations are sustained by batteries and with the capability of attaching a solar cell-based energy-harvesting system to implement the proposed optimization strategy. Specifically, we consider the Raspberry PI 2 Model B v1.1 [40], Arduino UNO [41], and Tmote [42] platforms. In the first simulation, we evaluate the execution time of the algorithm on the Arduino and Raspberry PI platforms to assess its feasibility; in the second we evaluate the quality of the solution found by the algorithm under different simulated conditions; in the third we compare the quality of the solution obtained by our algorithm against the algorithm presented in [8]. To the purpose of the first simulation we have implemented the algorithm in C (for Raspberry PI) and in the C-like specific language of Arduino UNO. Instead, to the purpose of the last two simulations, we have devolped a simulator in Python.

For evaluation purposes, we have considered using the same battery for the three platforms: a 3.7 V Lithium-Polymer battery with a capacity of 2000 mAh (Model 803860 manufactured by Shenzen PKCELL Battery Co. Ltd). For simplicity, it is considered that the state of charge of the battery (SOC) is linearly dependent of the battery voltage. The maximum battery level, $B_{\max}$=2000 mAh, corresponds to the maximum voltage of the battery which is 4.2 V. In order to avoid deep discharges and preserve the state of health of the battery, in our application, the battery voltage will not be allowed to go below 3.4 V, which corresponds to 10% of the total capacity (i.e. $B_{\min}$=200 mAh). It is also considered that a voltage regulator is used to provide the appropriate supply voltage to each platform, according to their specifications.

In order to achieve a sustainable operation of IoT platforms considered, the battery level should always be within the range $[B_{\min}, B_{\max}]$. Additionally, we fix a starting battery level $B_1 = B_{\min} + \frac{B_{\max} - B_{\min}}{2}$. The battery pack is recharged using

[1]Python uses zero base indexes for array or matrix.

the KL-SUN3W solar cell-based energy-harvesting system described in Section III. The rate in which the battery level is increased by the panel depends on, among other factors, the solar production in some location. To this end, we consider the city of Madrid in Spain (latitude $40.42°$), and so we take its values of average daily solar irradiance D (in $KWh/m^2/day$) provided by RETScreen [37]: 2.03, 2.96, 4.29, 5.11, 5.95, 7.09, 7.2, 6.34, 4.87, 3.13, 2.13, 1.7, corresponding to the months of the year from January to December. Since it is known that the solar energy production is predictable and uncontrollable, and that its natural cycle is 24 hours, we consider a time frame of $T = 24$ hours and a total number of slots per day $K = \{24, 48, 72, 96, 120, 144, 240, 288\}$, with $k = \frac{K}{24}$ slots per hour respectively, each one with duration $\Delta T = \frac{60}{k}$ minutes. Thus, the amount of energy generated in each slot is computed as $E_{slot} = \frac{D(h) \times \eta \times S}{\Delta T}$, and its corresponding battery charge in mAh per slot is then computed as $\frac{E_{slot}}{V_{mp}} \times 1000$.

In turn, battery is drained at a rate that depends on the energy efficiency of the hardware components being used by the application (e.g. radio, sensors, memory), and on the application execution model of the IoT platform. For instance, a platform may integrate a (low-power) microcontroller that runs a single application (e.g. Arduino, Tmote) or, alternatively, a microprocessor holding a traditional operating system able to run concurrently several applications (e.g. Raspberry PI). For the sake of simplicity, the energy cost due to the execution of an application on each particular platform may be computed as the sum of the consumptions when the platform keeps in an active state I_{active}, which represents the average energy cost (in mA) to run some application plus a consumption in a sleep or standby mode I_{idle}, which represents the average energy cost (also in mA) when the platform is idle, i.e. it is not running any application. This approach is consistent with the fact that a certain operation has a different energy cost depending on the platform where it is executed. In order to compute the cost of a task on each one of the platforms considered, given its duty cycle, our simulator assumes the next average values of current in active and idle state: for RPI I_{active}=230 mA and I_{idle}=100 mA[2]; for Arduino I_{active}=50 mA and I_{idle}=30 mA[3]; for Tmote I_{active}=22 mA and I_{idle}=0.021 mA[4].

Our simulator generates a sufficiently large set of $A = 50$ applications, each one composed of a set of $n \in [6, 10]$ tasks $\mathcal{T}$. Remind that each t_i performs a functionality with a certain degree of quality, thus, t_i is defined by the tuple $t_i = \langle q_i, c_i \rangle$, where q_i is its quality ($q_i \in [1, 100]$) and c_i is its energy cost. In order to define its c_i, i.e. its energy cost in a specific platform, we first generate a duty cycle of the task $dc_i \in [0.01, 0.5]$ (expressed on a per unit basis) that is common to all platforms, and compute its cost per hour in each platform, expressed in mAh, as $c_i = dc_i \times I_{active} + (1 - dc_i) \times I_{idle}$, where I_{active} and I_{idle} correspond to the currents in active and idle state, respectively, of the platform where t_i is executed. The cost of the task in mAh per slot is then easily computed as $\frac{c_i}{k}$.

[2]http://www.raspberrypi.org/help/faqs/#powerReqs

[3]https://www.gadgetmakersblog.com/arduino-power-consumption/ and https://www.arduino.cc/en/Tutorial/ArduinoZeroPowerConsumption

[4]http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

It should be noted that, given two tasks $t_1 = \langle q_1, c_1 \rangle$ and $t_2 = \langle q_2, c_2 \rangle$, if $q_1 < q_2$ and $c_1 \geq c_2$, then t_1 is inefficient and it can be excluded a priori. For this reason we assume that, without loss of the generality, the higher the quality level, the higher cost. A summary of the parameters used by our simulator is presented in Table II.

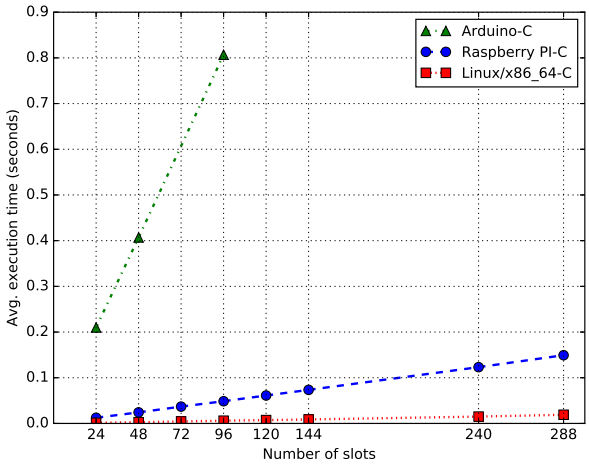

Figure 1. Average execution time of the optimization algorithm on Arduino, RPI and Linux platforms.

A. Execution Times

We have computed the average execution times taken by our optimization algorithm running on a Raspberry PI (RPI), an Arduino, and on a Linux/x86_64 platform, for a set of A applications. To this end, we developed a C version of our Python simulator to be run on the three platforms, since Arduino and Tmote do not support Python (note that Tmote does not support C but one of its dialects, nesC). Figure 1 depicts the average execution time in RPI and Linux for an increasing number of slots in a day $K \in [24, 288]$ and for a subset of the slots $K = \{24, 48, 96\}$ in Arduino, due to its strong memory constraints (2 KB of SRAM and 1 KB of EEPROM [41]). As observed, the average computation time grows linearly with the number of slots in all platforms. The execution times shown obey to the clock speed of each platform: the ATmega328 microcontroller of Arduino uses a 16 Mhz clock, RPI uses a 900 MHz quad-core ARM Cortex-A7 and the Linux system uses 4 2.4Ghz quad-core Intel processor. Note that, even for the maximum execution time, the overload due to the execution of our algorithm is very small: in the worst case, which is given by Arduino with 96 slots or 4 slots per hour of 15 minutes, the execution time is 0.8 seconds; this means a practically negligible overload of 0.08% with respect to the duration of the slot. The increase of the number of slots results into a larger execution time but contrasts with a growing quality (as we show in the next subsections), which pose a tradeoff between execution time and quality.

Table II
SUMMARY OF THE PARAMETERS USED IN OUR SIMULATIONS. (*) SEE DETAILS OF THE TASKS IN TABLE III. FOR THE SIMULATIONS OF *Execution time* AND *Tasks Allocation* ONLY ARE SHOWN THE PARAMETERS THAT DIFFER FROM *Quality*.

Evaluation criteria	Platform	Symbol	Description	Value
Quality	Arduino, RPI, Tmote	A	Number of applications simulated	50
		n	Number of alternative tasks per application	$[6, 10]$
		q	Range of quality for the tasks	$[1, 100]$
		dc	Range of duty cycle for the tasks	$[1, 50]$ (%)
		K	Range of slots per day	$\{24, 48, 72, 96, 120, 144, 240, 288\}$
		B	Range of battery levels	$[200, 2000]$ (mAh)
Execution Time	Arduino	K	Range of slots per day	$\{24, 48, 96\}$
		A	Number of applications simulated	30
		B	Range of battery levels	$[0, 50]$ (mAh)
	RPI, Linux	K	Range of slots per day	$\{24, 48, 72, 96, 120, 144, 240, 288\}$
		A	Number of applications simulated	30
Tasks Allocation	Arduino, RPI, Tmote	n	Number of tasks	7^*

B. *Quality*

With the aim of balancing scalability against performance, we evaluate here the quality achieved by our optimization algorithm. Our simulator returns the average quality obtained by the scheduling of a set of $A = 50$ applications on an RPI, an Arduino, and a Tmote platform, with their corresponding task energy consumptions computed as described before, assuming the same battery boundaries $B_{\min}$ and $B_{\max}$, a certain solar production, and a variable number of slots. For each application, the algorithm finds the energy-neutral assignment of tasks-to-slots such that its overall quality is maximum. Since the amount of solar energy that is daily generated impacts on the assignment of tasks-to-slots and, in turn, on the overall quality, we evaluate twelve different scenarios of generation corresponding to the twelve months of the year.

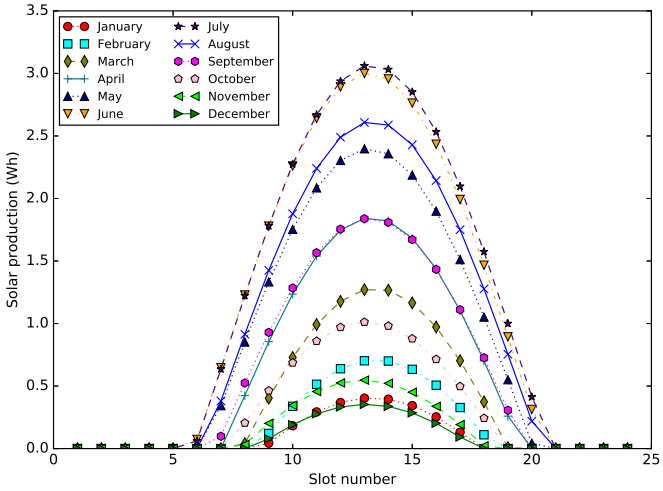

Figure 2. Solar energy distribution for 24 slots in a day and for all months of the year.

Figure 2 shows the hourly energy distributions among $K = 24$ slots when the KL-SUN3W solar panel is used as energy harvester and installed in Madrid. Although the amount of solar energy that is generated is based on the KL-SUN3W panel specifications, in order to guarantee the execution of the whole set of A in the three platforms, i.e. to schedule the task of lowest duty cycle of each A (therefore, with the minimum cost and minimum quality), we had to resize the panel accordingly to each platform. To this end, we proceed to find the lowest integer factor such that A becomes feasible in all platforms. In the case of RPI and Arduino we had to scale the size S multiplying by a factor of 7 and 2, respectively; however, for supporting the execution of A in Tmote we were able to reduce S by a factor of 10.

Figure 3 on the left shows the average quality delivered for the simulation of A applications assuming a RPI platform, working in all months of the year, with different number of slots per day $K = \{24, 48, 72, 96, 120, 144, 240, 288\}$, with values ranging between $[70, 100]$. Regardless the number of slots, the larger energy production, the higher quality. Thus, June and July, the months with the largest production provide the largest quality whilst December, with the lowest production, provides the lowest quality. The effect of increasing the number of slots is shown in this figure. Quality grows with the number of slots up to converge into the overall maximum quality with the larger value of K, regardless the solar production. In general, a larger number of slots results into more assignments of tasks-to-slots, where the tasks assigned execute for less time and the production per slot will be less with regard to a larger number of slots, if we assume that $E(h)$ is uniformly distributed between all the slots in an hour. Thus, when we increase the number of slots we reduce in the same proportion the energy production per slot and the cost of the execution of the task, which depends on the duration of the slot. This means that, at least, we can keep running the same task previously assigned and, alternatively, to exploit the choice of selecting a new better application, therefore with higher cost and higher quality, subject to energy neutrality and feasibility conditions. Since the energy neutrality condition is checked at the last slot (e.g. $B_{K+1} \geq B_1$), in each intermediate assignment of task-to-slot the algorithm may select a not energy-neutral application with regard to the individual slot and with higher cost, which results into a larger average quality. Consider the next simple example where we have three tasks $t_1 = \langle 80, 2\rangle$, $t_2 = \langle 85, 4\rangle$ and $t_3 = \langle 100, 6\rangle$, an only slot with production $p_1 = 4$ and an initial battery $B_1 = 5$, $B_{\min} = 0$ and $B_{\max} = 30$. Under

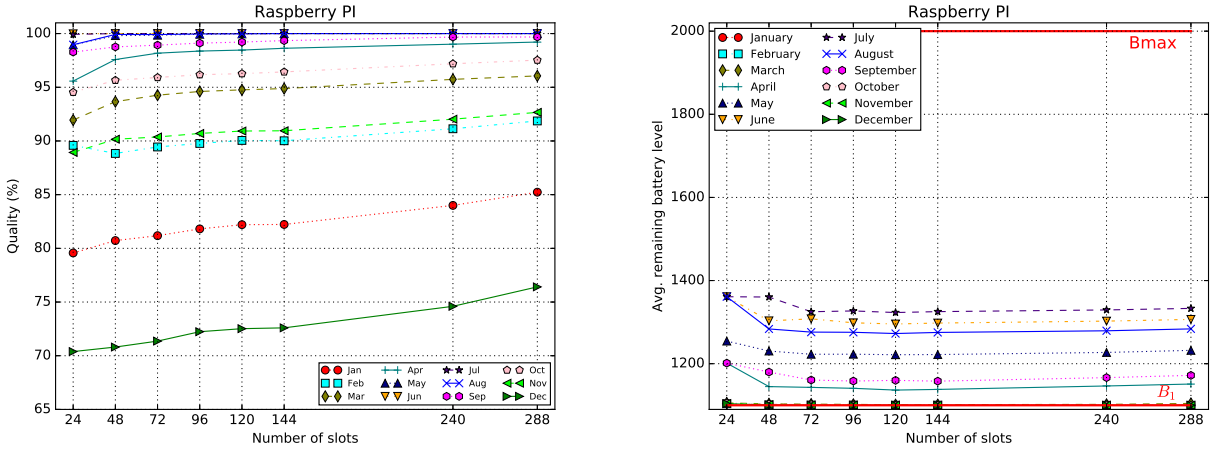

Figure 3. Simulation results of a RPI platform: average quality for a different number of slots K (on the left) and the corresponding average remaining battery level (on the right).

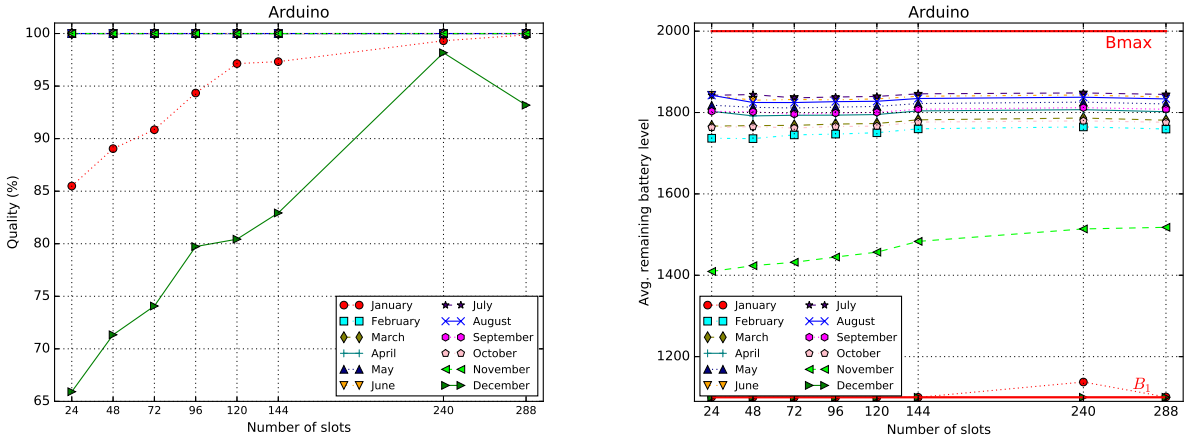

Figure 4. Simulation results of an Arduino platform: average quality for a different number of slots K (on the left) and the corresponding average remaining battery level (on the right).

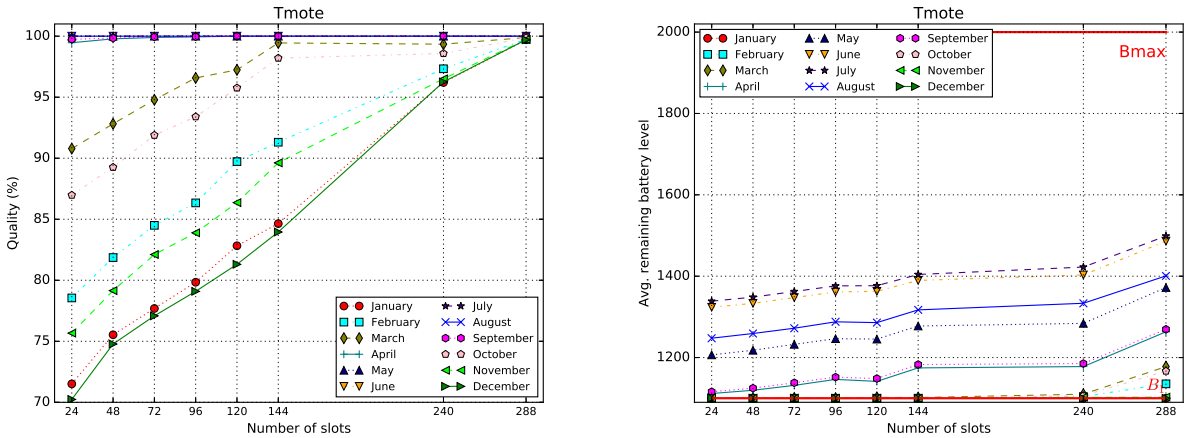

Figure 5. Simulation results of a Tmote platform: average quality for a different number of slots K (on the left) and the corresponding average remaining battery level (on the right).

these conditions, the best choice is to select for execution t_2, which generates a residual battery of $B_1 + p_1 - c_2 = 5$ and

Table III
TEST APPLICATIONS FOR RASPBERRY PI, ARDUINO, AND TMOTE.

Platform/Task	T_0	T_1	T_2	T_3	T_4	T_5	T_6
dc_i (%)	2	13	22	29	31	45	47
RPI	⟨69,103⟩	⟨81,117⟩	⟨88,129⟩	⟨90,138⟩	⟨94,141⟩	⟨96,159⟩	⟨100,162⟩
Arduino	⟨69,31⟩	⟨81,33⟩	⟨88,35⟩	⟨90,36⟩	⟨94,37⟩	⟨96,39⟩	⟨100,40⟩
Tmote	⟨69,1⟩	⟨81,3⟩	⟨88,5⟩	⟨90,7⟩	⟨94,7⟩	⟨96,10⟩	⟨100,11⟩

a quality of 85%. Energy neutrality and feasibility conditions hold. Equivalently, for the same period we could have two slots with the half production $p_1 = 2$ and $p_2 = 2$, where the previously selected application runs and delivers an average quality of 85%. However, a best solution would be to assign t_3 to p_1 and t_1 to p_2, since the battery level at the end of the first slot is $B_2 = B_1 + p_1 - c_3 = 4$ and at the end of the second slot is $B_3 = B_2 + p_2 - c_1 = 5$, which keeps energy neutrality condition and also provides a best average quality of 90%, thus improving the quality achieved with a single slot.

Correspondingly, Figure 3 on the right shows the average remaining battery in the last slot B_{K+1} after simulating A applications. The average remaining battery keeps within the limits given by B_1 and $B_{\max}$, which means that energy neutral solutions were found for A with the different curves of solar production and for the different configurations of slots evaluated. The average residual battery level is lower in the months of lower production, which means that most of the budget is destined to the applications execution. The residual battery tends to grow slightly with K; this is another effect of the division into more slots. The tight deviation between B_1 and the average residual battery in December indicates how close is the solution to the limit of energy neutrality in this case.

The simulation results of the Arduino platform are represented in Figure 4. On the left, it is shown the average quality obtained after scheduling the same A applications for any $K \in [24, 288]$. In this case, most of the solar productions with the exception of December and January provide the maximal quality. On the right, the corresponding average remaining battery after simulation is shown. Both results confirm the observations described for RPI. As observed, the average residual battery is larger than the obtained one on the RPI platform (the lowest battery corresponds to the two months indicated), which means that in Arduino we may schedule A with the maximum quality and we have still also an excess of battery, which pose the possibility to run more expensive applications (with highest quality) than those used on the RPI platform or, alternatively, to resize the solar module. Finally, the simulation results on the Tmote platform are presented in Figure 5: on the left, it shows the optimal quality achieved by the scheduling of the same A applications and, on the right, it shows the corresponding average remaining battery values, which are closer to B_1 in this case.

Finally, a note about the accuracy of the quality results provided. The average quality results on the three platforms, all the months of the year, and for all K, presented a 95% confidence interval of $\pm\alpha$, with α=0.87 for RPI, α=0.56 for Arduino, and α=0.44 for Tmote, where α is the maximum deviation found for each platform and whose values correspond to the simulation of the month of December with $K = 24$ slots.

C. Comparison Against Upgrade/Downgrade Strategy

This subsection compares our optimization algorithm with respect to our upgrade/downgrade strategy (hereafter U/D strategy) previously proposed in [8]. This latter approach is based on determining the energy-neutral scheduling that maximizes the overall quality from an initial assignment, where the most efficient application, i.e. the one with the highest ratio between quality and cost, is assigned to all slots. After such initial scheduling, the algorithm may greedily both increase (e.g. upgrade) and decrease the quality (e.g. downgrade) by assigning slot-by-slot the next application less efficient but with higher quality (in the first case) or the application with the next lower cost (in the second case). For comparison purposes, we have randomly generated a single test application composed of $n = 7$ tasks with different costs and qualities, where cost is computed from the duty cycle of the task (see details in Table III). Next, we report the comparison results in terms of quality and assignments provided by both scheduling strategies.

Figure 6 on the left shows the quality delivered by both strategies for the test application described in the third row of Table III, in the months of March, July, September, and December, with $K \in [24, 288]$ slots. As observed, the quality delivered by the optimization algorithm is always higher than the one provided by the U/D strategy regardless the solar production. Note that the U/D strategy does not report an energy-neutral solution for the production of December and for any K. On the right, the corresponding assignments of tasks-to-slots are presented for $K = 96$ slots. In this figure, the y axis represents the task (from T_0 to T_6) and the x axis represents the assignment slot (from 1 to 96). Remind that the larger index of the task, the higher quality. As shown, the most expensive assignment (with highest quality) corresponds to the month of July with our optimization strategy (T_6 with quality 100% is assigned to all K), whilst the less expensive one (with lowest quality) corresponds to December in our strategy (T_0 and T_1 with qualities 69% and 81%, respectively, are assigned). These two assignments provide the largest quality deviation (27%).

The same experiment is repeated assuming the Arduino platform for the test application shown in the fourth row of Table III. Figure 7 on the left compares the quality achieved by the optimization strategy against the obtained one by the U/D strategy. As in the RPI simulation, the quality achieved by the optimization algorithm is higher than or equal to the

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/JIOT.2018.2828943

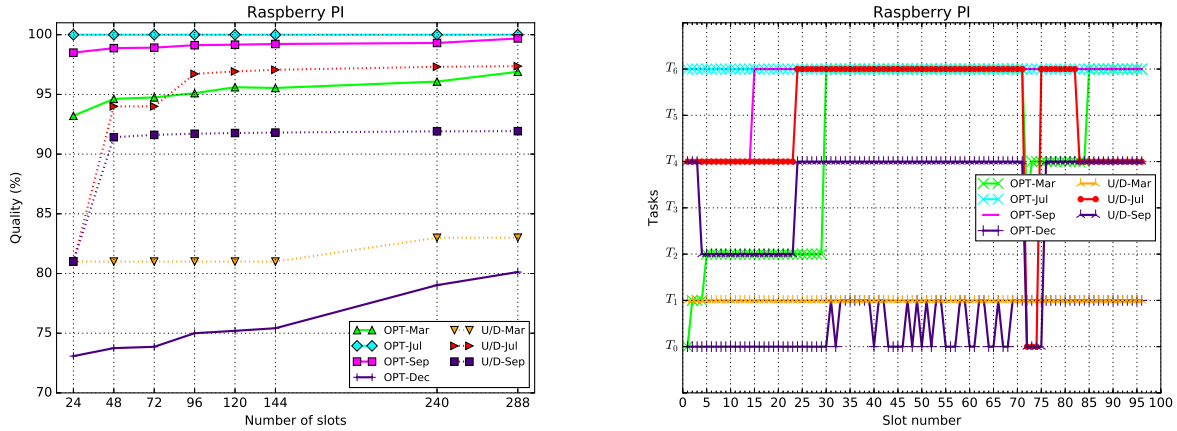

Figure 6. Quality achieved by the RPI application shown in Table III, and scheduled according to the optimization algorithm and the U/D strategy (on the left) and the corresponding assignment of tasks-to-slots (on the right). **Legend: OPT:** Optimization strategy. **U/D:** Upgrade/Downgrade strategy.

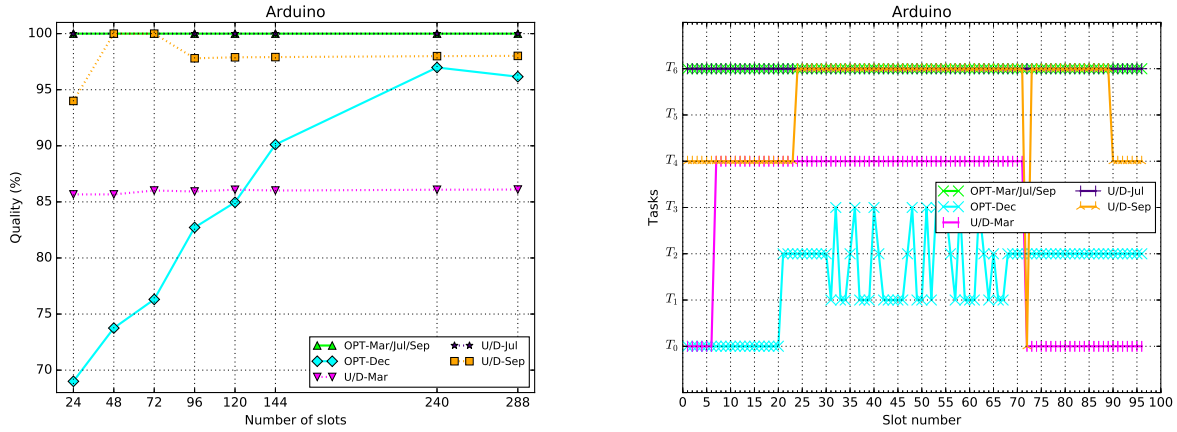

Figure 7. Quality achieved by the Arduino application shown in Table III, and scheduled according to the optimization algorithm and the U/D strategy (on the left) and the corresponding assignment of tasks-to-slots (on the right). **Legend: OPT:** Optimization strategy. **U/D:** Upgrade/Downgrade strategy.

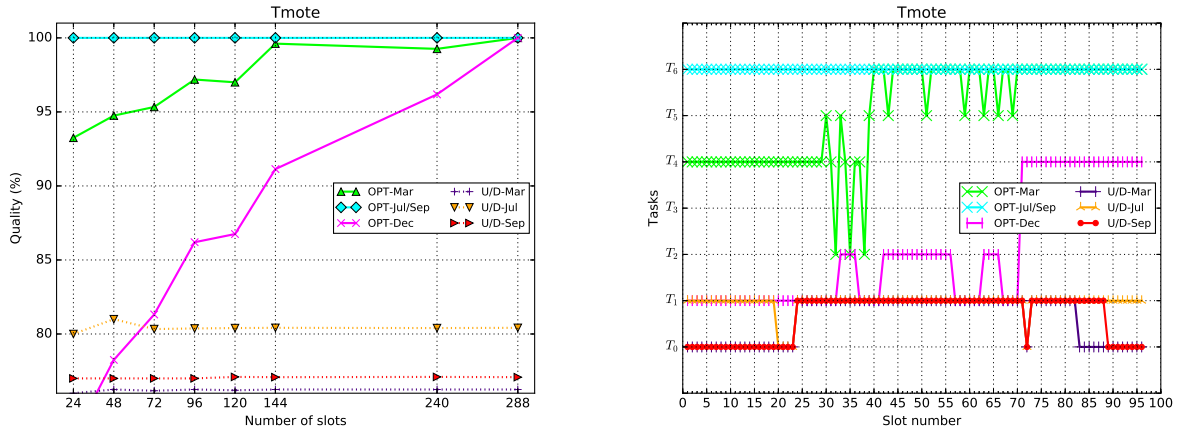

Figure 8. Quality achieved by the Tmote application shown in Table III, and scheduled according to the optimization algorithm and the U/D strategy (on the left) and the corresponding assignment of tasks-to-slots (on the right). **Legend: OPT:** Optimization strategy. **U/D:** Upgrade/Downgrade strategy.

one provided by U/D; and as in the RPI simulation no solution is found by U/D for the production of December. In July, with both strategies, and during the months of March and September with the optimization algorithm, the maximum quality is achieved (100%) regardless K. Note that the lowest quality corresponds to the month of lowest production

(December) with our optimization algorithm for only a subset of $K = \{24, 48, 72, 96, 120\}$ whilst that for the remaining values of K, it corresponds to the month of March with the U/D algorithm. This fact shows that our algorithm better exploits the choice of the number of slots K to deliver a larger quality even with a lower solar production. The corresponding assignments of tasks-to-slots for $K = 96$ slots can be viewed in Figure 7 on the right. The best schedule delivers a highest quality of 100% and assigns T_6 to K slots: this schedule corresponds to the month of July in both strategies and to the months of March and September with our optimization algorithm). Finally, in Figure 8 on the left we present the results for the application executed on a Tmote (see fifth row of Table III). Similarly to the RPI and Arduino tests, the optimization algorithm achieves always a higher (or equal) quality than U/D strategy. For $K = 96$ slots, the two approaches generate the assignments of tasks-to-slots presented in Figure 8 on the right.

D. Continue Optimization

Our optimization algorithm uses estimations of the solar energy that is produced in each slot of time to compute the optimal scheduling in advance. However, during the real energy-harvesting system execution it could be observed an excess (or deficit) of production with regard to the estimation. In such situations, where the energy that is really harvested differs from the energy estimated in a certain slot of time, the scheduling originally computed may become invalid or inefficient. Therefore, a re-optimization is necessary to compute a new assignment targeted to the new conditions. This subsection explains how our algorithm behaves under the event of re-optimization.

The real solar energy production during a day has been measured to analyze the effect of the divergences from the estimated one by using the solar module KL-SUN3W, which is described in Section III. The location chosen is the city of Ciudad Real in Spain (latitude 38.98°) and the day was the 27th of March 2017, which was characterized by frequent intervals of clouds, resulting in a rather challenging scenario in terms of solar production deviations, as it can be seen in Figure 9. The methodology adopted to obtain the measurements was to measure the short-circuit current (I_{sc}) of the KL-SUN3W photovoltaic panel every minute by using a digital multimeter with data-logging functionality (see Figure 10). The photovoltaic panel was oriented towards the South (i.e. 0° azimuth angle) and a tilt angle of 39° equal to the latitude of Ciudad Real. The DC current measurement accuracy of the Keysight U1242C digital multimeter is 0.1%+2 counts. The solar irradiance can be derived from I_{sc}, since they are highly correlated and almost proportional [43]. This is a simple, but yet effective way of measuring the solar irradiance without the use of expensive instrumentation (i.e. pyranometers), and serves for the purposes of this analysis. Assuming that the short-circuit current and the irradiance are proportional and neglecting the effect of temperature, the irradiance value that corresponds to the current measurement can be obtained multiplying the latter by the constant 1818 W/Am^2, which is obtained from the quotient between the standard condition

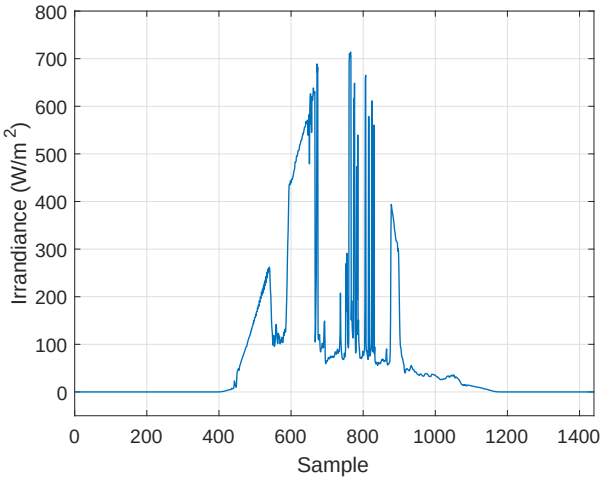

Figure 9. Measured irradiance on the 27th of March 2017 in Ciudad Real (Spain).

Figure 10. Measurements setup employing the KL-SUN3W PV panel and a Keysight U1242C with data logger functionality.

irradiance 1000 W/m^2 and the I_{sc} value 0.55 A. Power is finally obtained by multiplying the calculated irradiance by the effective area and efficiency of the panel.

Figure 11 compares the curves of real and estimated solar production on 27th March, distributed among $K = 24, 144$ slots. We have calculated the quality obtained by the execution of an application on the three platforms of study, when re-optimization is applied, i.e., we assume the real production shown in Figure 11. The application is composed of $n = 7$ tasks whose duty cycles range between $[13, 46]$ and qualities range between $[51, 93]$. In this specific case, in which the real production is lower than the estimations, the algorithm selects a less expensive scheduling to keep the system energy-neutral. The results of the simulation are shown in Figure 12. The re-optimization process adjusts the assignment by means of a new scheduling whose quality is not always lower than the ideal scheduling based on estimations, in spite of the real energy harvested is lower than the energy estimated. Indeed, as observed in the figure, the qualities delivered by the optimization algorithm in both cases are very close or even equal (e.g. Arduino) in all configurations of slots. This means

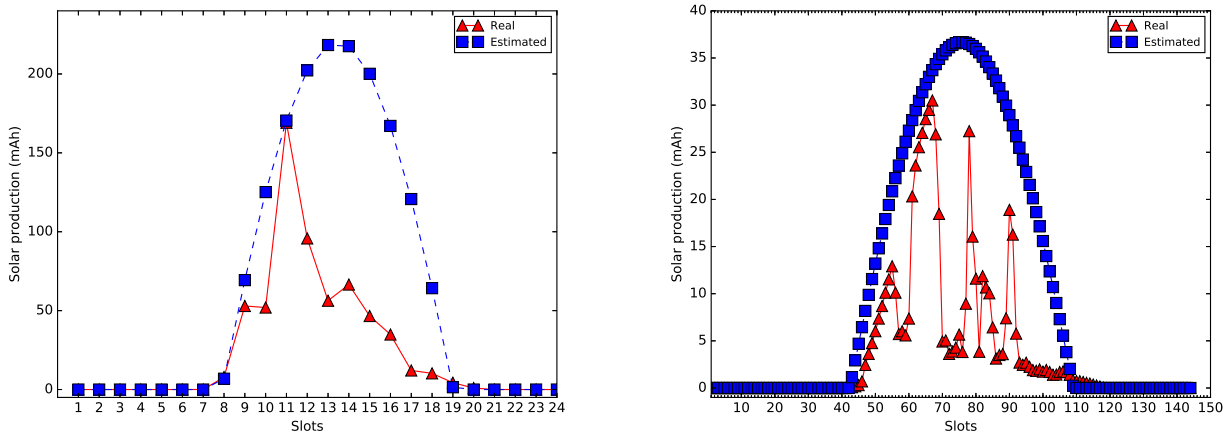

Figure 11. Distributions of the real vs. estimated solar production (in mAh) measured on the 27th of March 2017 in Ciudad Real between $K = 24, 144$ slots.

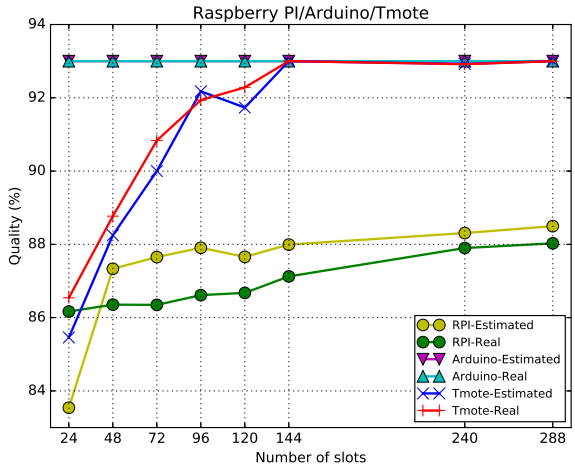

Figure 12. Ideal (based on estimations) and real quality achieved by the optimization algorithm in conditions of underproduction, for an application executed on RPI, Arduino and Tmote platforms.

that, in case of underproduction, the optimization algorithm may still, in some case, achieve the same quality than the one obtained with the expected energy production.

VII. Conclusions

A new generation of IoT devices, equipped with more and more sensing, processing, and communication resources, is now available to support complex and expensive tasks. In many outdoor IoT scenarios, such as smart cities and smart agriculture, the problem of empowering such devices may be faced by means of energy-harvesters, which provide an endless energy source to sustain the device's operations, since the energy is naturally extracted from their surroundings. However, the energy produced by these energy-harvesting systems, as the ones based on solar cells, is not controllable. Hence it is needed to adequately manage the cycles of available energy and the energy consumption of the applications to maximize the application utility while ensuring energy-neutrality of the devices. This paper deals with the optimality of the task scheduling in solar panel-based energy harvesting systems, i.e. with the problem of finding the optimal assignment of tasks along a time frame discretized in a certain number of slots, such that the overall quality of the tasks executed in a day is maximized and the device is energy neutral. After proving this problem to be NP-Hard, we offer a dynamic programming algorithm able to solve it with pseudo-polynomial complexity on the number of slots and on the number of levels representing the battery charge. Note that, in principle, this complexity does not guarantee an efficient implementation, because the algorithm has to explore the entire space of possible solutions to find the optimum. However, we show that, in practical conditions the space of solutions is limited (since both the battery levels and the number of slots per day are limited) and then the algorithm can be implemented efficiently even in low-power platforms such as Arduino, Tmote or Raspberry PI. On the contrary, the existing algorithm that are based on greedy algorithms (and that are efficient in terms of time of computation) does not find, in general, the energy neutral scheduling of the tasks which has the optimum quality.

We have implemented our dynamic programming algorithm in Python to simulate the execution of a large number of applications on three real-world IoT platforms: Raspberry PI, Arduino, and Tmote, all equipped with a photovoltaic panel, and we also have implemented the algorithm in C to test its execution times in the three platforms. The execution tests on the real platform showed that the algorithm can be used in all the three platforms with a small overhead. Concerning the simulations, they were done with different configurations of slots, tasks (cost and quality) and solar energy productions. The results demonstrate that the maximum quality is achieved with the largest number of slots evaluated, regardless the solar production, even under the event of re-optimization (i.e. real production diverges from the expectations). The optimization strategy is also compared against our previously published greedy upgrade/downgrade strategy: the results have demonstrated that the dynamic approach outperforms the greedy approach. As future works, there are some research line that

we intend to pursue. The first concerns the evaluation of our optimization algorithm with real, long-lasting IoT applications in challenging scenarios of real solar production. A second line of research concerns the extension of our approach to devices equipped with several energy-harvesters able to collect energy from different natural sources (e.g. solar, wind, movement, etc.) and with the ability of using them in an alternative way when available.

ACKNOWLEDGMENT

This work has been funded by the Programme for Research and Innovation of University of Castilla-La Mancha, co-financed by the European Social Fund (Resolution of 25 August 2014) and by the Spanish Ministry of Economy and Competitiveness under project REBECCA (TEC2014-58036-C4-1-R) and the Regional Government of Castilla-La Mancha under project SAND (PEII_2014_046_P).

REFERENCES

[1] R. Haight, W. Haensch, and D. Friedman, "Solar-powering the Internet of Things," *Science*, vol. 353, no. 6295, 2016. [Online]. Available: http://science.sciencemag.org/content/353/6295/124/tab-pdf 1

[2] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1996–2018, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6805162 1

[3] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 4, Sep. 2007. [Online]. Available: http://doi.acm.org/10.1145/1274858.1274870 1, 2, 4, 5

[4] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Operating Systems Review (ACM)*, vol. 35, no. 5, 2001. 1

[5] G. Amato, A. Caruso, and S. Chessa, "Application-driven, energy-efficient communication in wireless sensor networks," *Computer Communications*, vol. 32, no. 5, pp. 896–906, 2009. 1

[6] J. A. Khan, H. K. Qureshi, and A. Iqbal, "Energy management in Wireless Sensor Networks: A survey," *Computers & Electrical Engineering*, vol. 41, pp. 159–176, Jan. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0045790614001773 1, 2

[7] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for sensor networks," in *in Ambient Intelligence*. Springer Verlag, 2004. 1

[8] S. Escolar, S. Chessa, and J. Carretero, "Energy management in solar cells powered wireless sensor networks for quality of service optimization," *Personal and Ubiquitous Computing*, vol. 18, no. 2, pp. 449–464, 2014. 1, 4, 6, 10

[9] Escolar, Soledad and Chessa, Stefano and Carretero, Jesus, "Energy-neutral networked wireless sensors," *Simulation Modelling Practice and Theory*, vol. 43, pp. 1–15, 2014. 1, 3, 4

[10] S. Escolar, S. Chessa, and J. Carretero, "Optimization of quality of service in wireless sensor networks powered by solar cells," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, July 2012, pp. 269–276. 1, 2

[11] ——, "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. 1, 3

[12] C. Rusu, R. Melhem, and D. Mossé, "Multi-version scheduling in rechargeable energy-aware real-time systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 271–283, 2005. 1

[13] S. Basagni, M. Y. Naderi, C. Petrioli, and D. Spenza, *Wireless Sensor Networks with Energy Harvesting*. John Wiley & Sons, Inc., 2013, pp. 701–736. [Online]. Available: http://dx.doi.org/10.1002/9781118511305.ch20 2

[14] J. M. Gilbert and F. Balouchi, "Comparison of Energy Harvesting Systems for Wireless Sensor Networks," *international journal of automation and computing*, vol. 5, no. 4, pp. 334–347, 2008. 2

[15] M. K. Stojčev, M. R. Kosanović, and L. R. Golubović, "Power Management and Energy Harvesting Techniques for Wireless Sensor Nodes," in *Telecommunication in Modern Satellite, Cable, and Broadcasting Services, 2009. TELSIKS'09. 9th International Conference on*. IEEE, 2009, pp. 65–72. 2

[16] S. Sudevalayam and P. Kulkarni, "Energy Harvesting Sensor Nodes: Survey and Implications," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 3, pp. 443–461, 2011. 2

[17] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the internet of things," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 375–380. 2

[18] A. A. Babayo, M. H. Anisi, and I. Ali, "A Review on energy management schemes in energy harvesting wireless sensor networks," *Renewable and Sustainable Energy Reviews*, vol. 76, pp. 1176–1184, sep 2017. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1364032117304598 2

[19] G. V. Merrett and B. M. Al-Hashimi, "Energy-driven computing: Rethinking the design of energy harvesting systems," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 960–965. 2

[20] L. Wang and Y. Xiao, "A Survey of Energy-Efficient Scheduling Mechanisms in Sensor Networks," *Mobile Networks and Applications*, vol. 11, no. 5, pp. 723–740, 2006. 2

[21] D. Zhang, Z. Chen, H. Zhou, L. Chen, and X. S. Shen, "Energy-balanced cooperative transmission based on relay selection and power control in energy harvesting wireless sensor network," *Computer Networks*, vol. 104, pp. 189–197, 2016. 2

[22] A. A. Nasir, X. Zhou, S. Durrani, and R. A. Kennedy, "Relaying protocols for wireless energy harvesting and information processing," *IEEE Transactions on Wireless Communications*, vol. 12, no. 7, pp. 3622–3636, 2013. 2

[23] Z. Ding, S. M. Perlaza, I. Esnaola, and H. V. Poor, "Power allocation strategies in energy harvesting wireless cooperative networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 2, pp. 846–860, 2014. 2

[24] M. W. Baidas and E. A. Alsusa, "Power Allocation, Relay Selection and Energy Cooperation Strategies in Energy Harvesting Cooperative Wireless Networks," *Wireless Communications and Mobile Computing*, vol. 16, no. 14, pp. 2065–2082, 2016. 2

[25] Y. Luo, J. Zhang, and K. B. Letaief, "Relay selection for energy harvesting cooperative communication systems," in *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 2013, pp. 2514–2519. 2

[26] D. C. Harrison, W. K. G. Seah, and R. Rayudu, "Rare Event Detection and Propagation in Wireless Sensor Networks," *ACM Computer Survey*, vol. 48, no. 4, pp. 58:1–58:22, May 2016. [Online]. Available: http://doi.acm.org/10.1145/2885508 2

[27] D. Noh, J. Kim, J. Lee, D. Lee, H. Kwon, and H. Shin, "Priority-based Routing for Solar-Powered Wireless Sensor Networks," in *Wireless Pervasive Computing, 2007. ISWPC'07. 2nd International Symposium on*. IEEE, 2007. 2

[28] Q. Tan, W. An, Y. Han, H. Luo, Y. Liu, S. Ci, and H. Tang, "Achieving energy-neutral data transmission by adjusting transmission power for energy-harvesting wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 16, no. 14, pp. 2083–2097, 2016. 2

[29] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time Scheduling for Energy Harvesting Sensor Nodes," *Real-Time Systems*, vol. 37, no. 3, pp. 233–260, 2007. 2

[30] A. Kansal and M. B. Srivastava, "An Environmental Energy Harvesting Framework for Sensor Networks," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, ISLPED'03*. IEEE, 2003, pp. 481–486. 2

[31] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design Considerations for Solar Energy Harvesting Wireless Embedded Systems," in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, April 2005, pp. 457–462. 2

[32] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan, "Adaptive Duty Cycling for Energy Harvesting Systems," in *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, ser. ISLPED '06. New York, NY, USA: ACM, 2006, pp. 180–185. [Online]. Available: http://doi.acm.org/10.1145/1165573.1165616 2, 5

[33] A. Kansal, J. Hsu, M. Srivastava, and V. Raghunathan, "Harvesting Aware Power Management for Sensor Networks," in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 651–656. [Online]. Available: http://doi.acm.org/10.1145/1146909.1147075 2

[34] C. Moser, J.-J. Chen, and L. Thiele, "Dynamic power management in environmentally powered systems," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*. IEEE, 2010, pp. 81–88. 2

[35] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management in energy harvesting systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '07. San Jose, CA, USA: EDA Consortium, 2007, pp. 773–778. [Online]. Available: http://dl.acm.org/citation.cfm?id=1266366.1266532 2

[36] S. Escolar, S. Chessa, and J. Carretero, "Quality of service optimization in solar cells-based energy harvesting wireless sensor networks," *Energy Efficiency*, pp. 1–27, 2016. [Online]. Available: http://dx.doi.org/10.1007/s12053-016-9458-3 3, 4, 5

[37] NASA, "Surface meteorology and Solar Energy (RETScreen)," https://eosweb.larc.nasa.gov/sse/RETScreen/, 2013. 3, 7

[38] KL., "KL Solar Company Pvt Ltd." http://www.klsolar.com/, 2014, kL Solar Company Pvt. Ltd., 1/482-B,Transport Nagar, Neelambur, Coimbatore, Tamil Nadu, PIN : 641062 India. 3

[39] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. 5

[40] Raspberry Pi Foundation, "Raspberry Pi 2 Model B," https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, February 2015. 6

[41] Arduino, "Arduino UNO," https://www.arduino.cc/en/Main/ArduinoBoardUno, September 2010. 6, 7

[42] Moteiv Corporation, "Tmote Sky DataSheet: Ultra low power IEEE 802.15.4 compliant wireless sensor module." http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf, June 2006. 6

[43] V. L. Vigni, D. L. Manna, E. R. Sanseverino, V. Dio, P. Romano, P. Buono, M. Pinto, R. Miceli, and C. Giaconia, "Proof of Concept of an Irradiance Estimation System for Reconfigurable Photovoltaic Arrays," *Energies*, pp. 6641–6657, 2015. 12