An Energy-Aware Task Scheduler for Energy Harvesting Battery-Less IoT Devices

Adnan Sabovic, Ashish Kumar Sultania, Carmen Delgado, Lander De Roeck and Jeroen Famaey, *Senior Member, IEEE*

***Abstract*—Tiny battery-less Internet of Things (IoT) devices that depend on the harvested energy from their environment provide a promising alternative for a sustainable IoT vision. These devices use small capacitors as energy storage, which together with the unpredictable and dynamic harvesting environment results in intermittent on-off behavior of the device. The crucial issue to effectively use battery-less IoT devices is to find a way of enabling the successful execution of application tasks in face of this intermittency. As the conventional computing models cannot handle this behavior, in this paper we present an energy-aware task scheduler for battery-less IoT devices based on dependencies and priorities, which can intelligently schedule the application tasks avoiding power failures and maintaining forward progress. With the properly defined voltage thresholds for each application task, using our energy-aware task scheduler a safer execution can be ensured. We evaluate our approach based on emulated and real experiments and validate it using two types of power management units (Environment Emulator and Intelligent Power Management Unit based on the AEM10941 chip). Our results show that the energy-aware task scheduler is able to react and adapt the execution to environmental changes, avoiding power failures. Comparing to the state of the art scheduling approaches, which are mostly not aware of the energy, we show that our energy-aware task scheduler can keep the device on during the full time of the experiment, executing more tasks when a relatively small capacitor of 10mF or less is used at harvesting currents as low as 40μA.**

***Index Terms*—sustainable IoT, battery-less IoT, intermittent computing, energy harvesting, BLE, energy-aware task scheduler.**

I. Introduction

THE Internet of Things (IoT) is a concept used to connect objects to the Internet, enabling billions of tiny devices, from smart-enabled devices to sensors for climate and agriculture monitoring, to cooperate and communicate with each other while performing different application tasks such as sensing (e.g., temperature and humidity), processing and transmitting data [1]. Due to the low price and maintenance, as well as easy usage, these devices are involved in a wide range of IoT applications, such as wildlife tracking, healthcare [2] [3], autonomous vehicles [4] or building monitoring [5] [6]. Typically, most IoT devices consist of a microcontroller (MCU), a radio chip (i.e., low power radio technologies), sensors and actuators to interact with the environment, and a battery that acts as a main power source [1] [7].

Adnan Sabovic, Ashish Kumar Sultania, Lander De Roeck and Jeroen Famaey are with University of Antwerp and imec, Belgium (e-mail: adnan.sabovic, ashishkumar.sultania, jeroen.famaey @uantwerpen.be, lander.deroeck@student.uantwerpen.be)

Carmen Delgado is with AI-Driven Systems, i2CAT Foundation, Barcelona, Spain email: (carmen.delgado@i2cat.net)

Stable power can be provided by batteries, but even when rechargeable, they are short-lived, lasting at most a few years. As the number of devices grows, the short lifetime of batteries requires maintenance, disposing and replacing, which becomes expensive and harmful to the environment. These batteries are bulky, which is not suitable for smaller devices, temperature sensitive and dangerous when not carefully protected, which makes their maintenance even more difficult in hard-to-reach areas. Even with some improvements, rechargeable batteries in combination with energy harvesting mechanisms, can still cause capacity degradation due to frequent charge-discharge cycles, reducing their lifetime [1]. Considering all of these, as well as their toxicity and chemically harmful composition that is ecologically unacceptable, they are incompatible with a sustainable IoT vision.

Improvements in processor architectures along with a reduction in energy consumption using low-power radio technologies enable a new type of devices that entirely depend on harvested environmental energy and do not need batteries for operating [8] [9]. These devices are powered by harvesting available energy from different environmental and renewable sources (e.g., solar, thermal or RF energy) which is stored in small capacitors. These capacitors are cheap and more resistant to capacity degradation, which prolongs their lifetime to more than a decade. Battery-less devices are easy to recycle, temperature insensitive and almost maintenance free, which makes them more environmentally friendly and suitable for operating in large-scale deployments.

Besides all advantages and improvements, there are still gaps and challenges that these devices face. Harvestable power sources are usually weak and unreliable with the amount of generated energy depending on current environmental conditions (e.g., solar energy is unavailable at night) [6]. During the inactive period (i.e., sleep state) the device replenishes its stored energy, which is scarce, especially when the capacitor is small. When sufficient energy accumulates, and the operating threshold is reached, the device starts executing tasks. A device works undisturbedly as long as there is enough stored energy. Once the energy is depleted and the device turns off, the volatile state (e.g., stack memory and register contents) and data are lost, and forward progress is interrupted. To ensure forward progress, the program must save the volatile state to non-volatile memory (e.g., FRAM or Flash) before the energy runs out. In the end, all these things cause intermittent execution, as shown in Figure 1, where the device turns on and off frequently, as it depletes and replenishes the stored energy in the capacitor. The charge and discharge cycle intervals

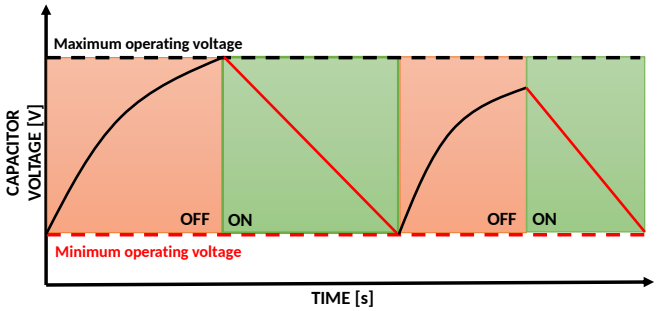


Fig. 1: Battery-less intermittent on/off behavior

depend on the considered hardware, capacitor size, and energy conditions [8].

The crucial issue to effectively use battery-less IoT devices is to find a way of enabling the successful cycle of task handling with their intermittency and reducing the possibility of power failures. Conventional computing models and static sequential applications cannot handle such behavior, as they lose forward progress assuming a stable power supply during execution [1]. This problem can be solved with task-based models [10] [11], where each task performs some atomic function, and its output is saved in non-volatile memory after it successfully completes. However, almost none of them consider the energy awareness. In this paper, we present an energy-aware task scheduler for battery-less IoT devices that can intelligently decide when to execute a specific task considering the harvested and available energy, energy consumed by the task, as well as its priority. Before each execution, the scheduler selects the task with the highest priority to be executed. To preserve the forward progress, after the task execution the output data, together with all dependent succeeding tasks will be stored in non-volatile memory. Considering the energy awareness, our approach can avoid power failures, improving the overall performance of the scheduler.

The main contributions in this paper are: (i) the main concept of our energy-aware task scheduling approach, including system design, algorithm and implementation; (ii) accurate device profiling methodology of different application tasks and device states, determining the impact of the scheduler on current draw; (iii) realistic evaluation of our approach based on a real implementation in a battery-less prototype device, and validation using two types of power management units (PMUs).

The remainder of this paper is structured as follows. Section II reviews the state of the art on battery-less computing and scheduling. In Section III, the energy-aware task scheduler for battery-less IoT devices, along with the application implementation, metadata, and scheduling algorithm is described. Section IV presents the accurate device profiling methodology to determine the current consumption and execution time of different application tasks and device states, as well as the impact of implemented scheduler. Section V shows the evaluation and validation results, together with discussion. Finally, conclusions are provided in Section VI.

II. Related Work

The main characteristic of battery-less IoT devices is their intermittent on/off behavior, where they can lose power at any point of time. As traditional computing models and approaches cannot handle such behavior, new strategies that can enable battery-less devices to operate under unstable power supply are required. Different models and schedulers that can handle battery-less intermittency have been proposed. They are mostly based on two main strategies, checkpointing and task-based models.

Checkpointing-based models capture a system state periodically and after reboot continue with the execution starting from the state captured by the checkpoint [12]. Mementos [13], Clank [14] and Ratchet [15] have tried to preserve the forward progress, but increasing the size of checkpoints and volatile memory, the program's time and energy overhead grows. With this behavior the possibility of depleting the available energy increases, which makes these approaches unsuitable for battery-less devices. Even considering the dynamic checkpointing approach [16], there is a possibility that the code operates with inconsistent values, which makes the memory consistency uncertain.

On the other side, task-based models split the program into different atomic subtasks, saving the output data in non-volatile memory after each execution. These models are more suitable for battery-less devices showing better performance in preserving forward progress. Alpaca [9] is a low-overhead programming model for intermittent computing on battery-less IoT devices, which decomposes the program into a sequence of user-defined tasks preserving forward progress despite power failures. Memory consistency is guaranteed through the privatization of shared data between connected tasks. Another task-based scheduling approach, based on task granularity, was presented by Colin et al. [12]. Similar to Alpaca, Chain guarantees memory consistency, but with higher overhead due to its channel-based memory model. The forward progress is ensured as long as the energy demand never exceeds the total energy storage capacity of the device. Hester et al. [11] presented Mayfly, a language and runtime for timely execution of sensing tasks on tiny, intermittently powered, energy harvesting sensing devices. Their approach ensures the forward progress and defines different types of constraints which keep the data consistency, freshness and utility across multiple power failures. All three mentioned approaches are based on static task flows and if the task cannot be finished due to lack of available energy, it will be executed again. The main problem here is the risk of task starvation because these schedulers do not advance any other task if the previous one cannot be completed. Also, in the first two scheduling approaches the selection of tasks based on their priorities has not been included.

Yildirim et al. [10] proposed an event-driven approach for battery-less IoT devices, introducing building blocks and abstractions that can enable reacting to changes in available energy keeping the sense of time and memory consistency. InK always tries to execute the event/task with the highest priority, without considering energy availability, and if the

available energy depletes during the execution, the task fails. Considering its dynamic strategy along with the predefined task priorities, the issue of task starvation can be avoided. However, none of aforementioned approaches take energy awareness into account. Compared to them, our study allows us to schedule tasks in an energy-aware fashion, taking into account energy harvesting budget and current consumption of the tasks, as well as their priorities. We consider and evaluate the real measured energy cost of every specific state of the device, including the impact of the scheduler. Based on current consumption, execution time, harvesting rate, and capacitor size, we can determine the required starting voltage for every specific task avoiding power failures.

AsTAR [17] is an energy-aware task scheduler that rapidly identifies optimum task scheduling rates and adapts quickly to environmental changes ensuring extremely low performance overhead in terms of memory, energy and execution time. It is fully autonomous and requires no pre-configuration, delivering sustainable operation on heterogeneous platforms. In their work, they considered larger super capacitors (between 1 and 5 Farad), which may be too large to fit into tiny battery-less IoT devices and only considered single-task applications without dependencies or priorities. Instead, we focus on smaller capacitors that are more suitable for tiny battery-less devices, considering different application tasks that are scheduled based on their priorities and dependencies. Majid et al. [8] presented an adaptive task scheduler that can adapt its execution to the incoming energy conditions at runtime. It supports a variety of capacitors and ensures forward progress by grouping tasks together when more energy is available. During the program execution the task priorities are not defined. Two scheduling algorithms for intermittent systems that schedule computational and energy harvesting tasks have been proposed by Islam et al. [7]. They define the equal priority for all considered tasks assuming that computing and harvesting tasks are separate and do not execute simultaneously. For validation, only a 680mF super capacitor has been considered, and task dependencies were not considered. In contrast, we allow the harvester and the device to work simultaneously and define different task priorities along with specific constraints enabling power-hungry tasks to be executed for the specific use case. Maeng et al. [18] presented an event-driven energy harvesting system, named CatNap, which splits the program into time critical code, for which energy must be reserved, and time insensitive code. CatNap executes events atomically without interruption by a recharge or a power failure. In their work, they consider only the worst case while charging the capacitor, and not for every event specifically, which can cause wasting time on useless full capacitor charging periods. Delgado et al. [1] presented an energy-aware task scheduling algorithm that is able to optimally schedule application tasks, avoiding power failures and providing insights on the optimal look-ahead time for energy prediction. Their work is complementary to ours, as they focused on the optimal algorithmic design of the scheduler, and we focus on the system and software that enables the deployment of such energy-aware scheduling algorithms on real IoT devices.

Finally, we have also presented an energy-aware task scheduling approach on battery-less LoRaWAN devices with energy harvesting [19], where we considered two different strategies allowing the device to sleep and turn off between the execution of application task cycles. We defined an optimization problem that determines the optimal capacitor voltage at which the device should start performing its tasks avoiding the possibility of power failure. In this paper, we extend the previous work in several ways. Our new energy-aware task scheduler is more generic and can be used with different applications and technologies, considering the generic dependencies among tasks. Also, after each task execution the output data is saved in non-volatile memory, which was not considered before. The model validation is performed based on an implementation in a real battery-less prototype device.

III. Energy-Aware Battery-Less IoT Devices

In this section, the brief overview of the energy-aware approach for battery-less IoT devices is provided. We start with the description of our energy-aware task scheduler for these devices, along with the application implementation, metadata, and scheduling algorithm. In the end, we describe different stages when power failures can occur, and how our energy-aware task scheduler handles them.

A. Energy-aware task scheduling approach based on dependencies and priorities

The main characteristic of battery-less IoT devices is their unpredictable behavior due to the highly variable energy supply. There are different proposed models and concepts that tried to enable successful task execution on battery-less devices, but only by considering the complete energy lifecycle (i.e., harvesting current, stored energy, task and device energy consumption), it is possible to avoid power failures. Because of that, in this paper, we propose a new energy-aware task scheduling concept that takes into account the harvesting current, current consumption for every specific state of the device, and is able to determine the minimum required voltage threshold that the device needs to reach in order to perform tasks successfully.

1) Application implementation and metadata: Our energy-aware task scheduling concept considers two main inputs:

i) The application implementation, which consists of a pre-compiled binary that implements each task as a function, as well as other functions relevant for the application, such as configuring the network interface or low power functionality. The considered application is divided into connected tasks where the output of a predecessor task acts as the input to its successor task(s). Each task is characterized by its name, execution time, current consumption, deadline, priority, and output. Considering the task parameters, such as execution time and current consumption, as well as the harvesting rate and capacitor size, the required voltage threshold for every specific task can be calculated and used during the program execution.
ii) The application metadata, which the user can define in JSON format. These JSON files are parsed and compiled by the scheduler when a new application is loaded into

Listing 1: Generic JSON representation of application tasks

```
{"Tasks": {

    "functionName": task_name,
    "currentConsumption": {
        "value": measured_value,
        "unit": unit_value
    },
    "executionTime": {
        "value": measured_value,
        "unit": unit_value
    },
    "deadline": {
        "value": set_value,
        "unit": unit_value
    },
    "priority": set_value,
    "initialTask": set_value,
    "requiredVoltage": {
        "value": measured_value,
        "unit": unit_value
    },
    "output": {
        "variable": output_variable,
        "type": output_type
    }
  }
}
```

Listing 2: Generic JSON representation of task flow

```
{"TaskFlow": {

    "parentTask": f_task_source,
    "childTask": f_task_destination,
    "constraint": {
        "variable": constraint_variable,
        "value": constraint_value,
        "type": constraint_type
    }
  }
}
```

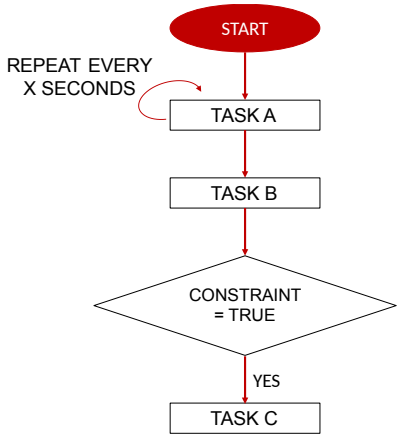


Fig. 2: Atomic task order considering constraints

the device. The metadata is provided in two main JSON structured files, the first file to define task parameters (Listing 1) and the second one to create a task flow (Listing 2).

The metadata ensures that the energy-aware task scheduler is completely independent of the specific application or application flow. It can be used with a wide range of different applications and scenarios, defining different task parameters and flows for different cases. In this way, changing the parameter values, order of tasks or even the complete task flow becomes much faster and easier, saving time and decreasing the burden for programmers.

The task flow description contains the task order and constraints definitions. All tasks are characterized by an order, which is implemented as a parent/child relationship, as illustrated in the example in Figure 2. Task A is the start task, and has two child tasks, Task A and Task B. Due to its periodicity that is defined as a constraint type, Task A can be considered as a parent and child of itself. Task B has only one child task, which is Task C. Task C cannot be executed if the constraint is not satisfied.

Constraints can take several forms:

i) **Repeat**, which defines the periodicity of task execution. Using this constraint, we can define how frequently the specific task should be executed (e.g., execute the temperature measurements every 1 second). Based on this value, the task will always be rescheduled after a predefined period. If the available energy is low, it may not be executed, or execution may be delayed.

ii) **Number of samples**, which defines the required number of output samples that the device has to collect in order to execute the next task in the flow (e.g., the required number of temperature samples is 3).

iii) **Data availability**, which checks if the needed input for the succeeding task in the flow is available. In case that this constraint is satisfied the task will be scheduled, but not necessarily executed first, depending on the scheduling algorithm.

iv) **Mathematical comparison on an output** (e.g., ¿, ¡, =), which checks the output of the preceding task, and based on the obtained value the energy-aware scheduler can decide on the next step. In case that this constraint is satisfied (e.g., the temperature value is lower than 25), the scheduler will schedule the child task for execution. If the constraint is not satisfied, the task will be skipped.

These constraints are presented in the application metadata that creates a task flow (Listing 2). Each task constraint is characterized by its variable, which is associated with the output from the parent task (e.g., average temperature is the output variable from the compute task), value, which defines a specific requirement based on the constraint type (e.g., with the availability constraint there is no need for a value, as we always want to check if the data is available), and type, which defines the constraint type (e.g., available refers to data availability).

2) Energy-Aware Task Scheduling Algorithm: The main goal of our energy-aware task scheduler is to maximize the number of successful task executions based on their predefined priorities, child-parent dependencies, and deadlines. The

Algorithm 1: Initial setup

```
1  if task instances stored in memory then
2      I ← retrieve task instances from memory;
3  else
4      T ← parse tasks and task flows from JSON
         metadata;
5      store T in non-volatile memory;
6      I ← [ ];
7      forall t ∈ T do
8          if t.initialTask == 1 then
9              i ← create task instance from t;
10             i.earliest_start_time ← current_time;
11             i.deadline ← current_time + t.deadline;
12             I ← I ∪ {i};
13         end
14     end
15     store I in non-volatile memory;
16 end
```

energy-aware task scheduler works with two types of lists: (i) the task list (**T**) that refers to the general concept, which contains a description of all considered tasks and flows based on the application metadata shown in Listing 1 and 2, and (ii) the list of task instances (**I**) that refers to a specific task execution, which includes all parameters from the task list with an additional start time parameter.

The energy-aware task scheduler will start with the initial setup by checking the memory status as shown in Algorithm 1 (Lines 1-6). In case there are already some tasks stored in the memory, the scheduler will retrieve the list of task instances from the memory and proceed with the next steps. This will occur if a power failure occurs after the scheduler already successfully finished the setup procedure before. Otherwise, it will need to read and parse tasks and task flows from the JSON metadata, and store them as structures in non-volatile memory.

The next step is to check the initial parent tasks from the task list (Line 8). The initial tasks are all tasks that have their *initialTask* parameter set to 1 in the JSON metadata (cf., Listing 1). Once the initial task is selected, it will be associated with a task instance created from that task (Line 9). The earliest start time of the selected task instance is set based on the current time (Line 10), and as it presents the initial task, the value is equal to 0. Therefore, the task instance deadline is equal to the initial task deadline defined in the JSON metadata (Line 11). After this part is completed, task instances of all initial tasks will be added to the task instance list, and stored in non-volatile memory (Lines 12-15).

Our energy-aware task scheduler works based on task priorities as presented in Algorithm 2. Each task instance is characterized by its priority retrieved from the JSON metadata, and the one with the highest priority from the task instance list will be selected (Line 2). If its deadline cannot be satisfied (taking into account also the time needed to charge the capacitor to the task's threshold voltage), the task instance will be removed from the list, and the next task instance with the

Algorithm 2: Priority-based energy-aware scheduling algorithm

```
1  while I ≠ ∅ do
2      i ← highest priority task instance from I;
3      if i.earliest_start_time +i.execution_time
         ≤ i.deadline then
4          sleep until i.earliest_start_time;
5          while capacitor_voltage ≤ i.required_voltage
             and current_time + t_v +i.execution_time
             ≤ i.deadline do
6              sleep t_v seconds;
7          end
8          if capacitor_voltage ≥ i.required_voltage and
             current_time +i.execution_time ≤ i.deadline
             then
9              execute i;
10             forall c ∈ i.task.children do
11                 if constraint of c is satisfied then
12                     i_c ← create task instance from c;
13                     if c is repeat task then
14                         i_c.earliest_start_time ←
                             current_time + c.repeat_delay;
15                     else
16                         i_c.earliest_start_time ←
                             current_time;
17                     end
18                     i_c.deadline ← current_time +
                         c.deadline;
19                     I ← I ∪ {i_c};
20                 end
21             end
22         end
23     end
24     I ← I \ {i};
25 end
```

highest priority will be selected. Otherwise, the device will sleep until the earliest start time (Line 4). In case that two task instances have the same priority, the one with the earliest deadline will be selected first.

As we consider the energy awareness in our scheduling approach, the required voltage thresholds for each task must be calculated, using the mathematical model and equations presented in our previous work [19]. Based on the obtained value, the scheduler is aware when the enough energy is collected to execute the task. The device will measure the capacitor voltage at predefined intervals (i.e., every t_v seconds) to check if the required voltage threshold of the selected task instance is reached. As long as the capacitor voltage is lower than required value, the device will go into sleep mode, and wait to repeat the measurement task again (Lines 5-7). This will delay execution, increasing the actual start time of the task instance based on the number of repeated voltage measurement cycles. Once the capacitor voltage is equal or higher than the required threshold, the task instance can be executed, if it can still be completed within its deadline (Lines 8-9).

In addition to task priorities, the energy-aware task scheduler also includes dependencies, which are implemented as child tasks. The relationship between the parent and child tasks are presented in the JSON metadata (cf. Listing 2) in the form of an application task flow. Each task instance can have one or more child tasks, which will be added to the list of task instances, but only if their constraints are satisfied, and subsequently, the highest priority task instance will be executed. If the constraint is satisfied, the new task instance will be created from the child task of the executed one (Lines 10-12). The earliest start time of the child task instance depends on the type of the task. In case it is a repeating task, its earliest start time will be based on the current time and required delay after which the task will be rescheduled (Lines 13-15). Otherwise, the earliest start time is equal to the current time (Line 16). The deadline in which this task instance must be executed is based on the current time and deadline specific to that task (Line 18), which is defined in the JSON metadata (cf. Listing 1). Once this part is finished, the child task instances will be added to the list of task instances, and stored in non-volatile memory (Line 19).

It must be noted that the proposed energy-aware task scheduler is modular, and the presented energy-aware priority-based algorithm can be replaced by a more elaborate algorithm, that not only looks ahead a single task execution, but predicts long-term task execution success rate (e.g., the recent scheduling algorithm proposed by Delgado et al. [1]).

3) Power failures and time-keeping: The important part of our energy-aware task scheduler is that all the necessary data is stored in flash, preventing their loss in case of a power failure. There are different stages when a power failure can occur. These are as mentioned below:

i) **After the JSON files are read and parsed** - after reading and parsing JSON files, all the necessary data related to the defined application tasks is stored in the flash as an array of C structs. Then the scheduler selects the initial tasks and saves them as task instances. In case that a power failure occurs immediately after this part is finished, the device needs to wait until it collects enough energy to turn on again and check if there is any content in the flash. The flash is not empty, and the scheduler is ready to continue with the next phase, without having to reparse the JSON files.

ii) **Before task execution** - the next phase is to select the highest priority task. When the scheduler selects the task, it is ready to be executed. If a power failure happens just before the execution, the relevant data is still safe in the flash, because after each task execution, the task instances must be updated with new values (dependent child tasks) and stored. After reboot, the scheduler will again pick the task, if its deadline has not passed, and continue with the execution.

iii) **During the task execution while some tasks are pending** - In case that a power failure occurs during task execution, after the reboot the device checks its flash and repeats the task, if it can still be completed within the deadline. In order to ensure task deadlines are met, it is necessary to keep time across power failures.

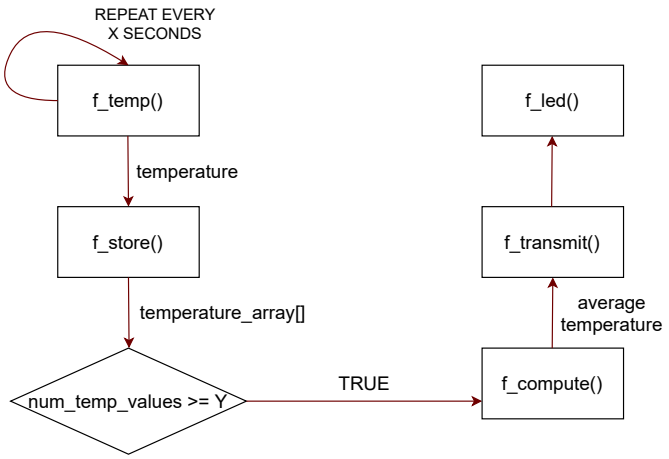


Fig. 3: IoT application flowchart including the defined tasks

Time-keeping is also important, to keep track of deadlines accros power failures. There are different solutions for this problem, such as (i) the approach based on a mathematical model that represents the elapsed time estimated by measuring how much voltage decayed across a small capacitor upon reboot [20], and (ii) a more accurate multi-tier timekeeping architecture based on a high-resolution remanence timekeeper, capable of tracking time across power failures, featuring an array of different RC circuits to be used for dynamic timekeeping requirements [21].

Our energy-aware task scheduler tries to avoid power failures as much as possible. A power failure is expected to only occur if the harvested energy is insufficient to keep the device powered in its lowest power state (i.e., during sleep).

IV. Device and Application Profiling

In this section, we describe the implemented IoT sensor application, including all defined tasks and the order of their execution. Two types of PMUs used for the validation of our energy-aware task scheduler, along with different task scheduling approaches used for comparison, are presented. A brief overview of the used device and application profiling methodology to get the current consumption and execution time of the different states of the devices, as well as the actual results, is provided. In the end, we explain how the harvesting current can be estimated, taking into account different possible cases.

A. IoT sensor application

For evaluation purposes, we implemented an IoT sensor application composed of five main tasks: temperature measurement, store the temperature values, calculate and transmit the average temperature, and in the end confirm the transmission by briefly turning on a LED.

Figure 3 shows the IoT application flowchart, which includes all defined tasks and their relation. The first task in the flow is temperature measurement, which is a periodic task, and can be repeated every X seconds. In our experiments, the sensing task was executed every 1 second and its first

start time was considered to be 0 seconds. When the device measures the temperature, the value is put in a temperature array, with a predefined size Y, and stored. After the store task is finished, the scheduler selects the compute task that is able to calculate the average temperature based on input values from its predecessor, but only if the constraint is satisfied. In this case, as constraint, we assumed the required number of sampled values to be 3. When the device collects 3 samples and calculates the average value, it is ready to transmit the data to the receiver node. After the successful transmission, as a confirmation, the LED will turn on for 500 milliseconds.

B. Power Management Units

There are different types of Power Management Units (PMUs) available in the market. These boards can have an internal mechanism to control charging and power supply periods, or some of them just act as a charger for the capacitor and always supply power to the external device requiring an external trigger to stop the supply. The proposed energy-aware task scheduling approach was validated using two PMUs: (i) Environment Emulator (EE) [22], and (ii) Intelligent-Power Management Unit based on the AEM10941 chip [23] designed by e-peas for solar and thermal harvesters.

1) Non-Intelligent PMU (Environment Emulator): The non-intelligent PMUs present a power source where the control is given to the external entity. As the device is normally powered via USB or batteries, the EE was used to emulate the energy harvesting environment and capacitor. The EE board was developed by De Mil et al. [22]. It acts as a virtual capacitor, and is able to emulate a wide range of energy harvesting and energy storage configurations. The EE is connected to the device under test (DUT) via expansion connectors. It measures the current consumption of the DUT and, based on a configured capacitor size and harvesting current, the EE provides a variable voltage to the DUT in line with the voltage that would be provided by a real capacitor.

2) Intelligent PMU based on the AEM10941 chip: The power management is performed using a single inductor boost/buck regulator. The boost regulator harvests energy from the solar panel to charge the on-board storage using maximum power point tracking. This system uses an RG trigger circuit which makes the capacitor charge only when its voltage is lower than a specific value. The battery-less IoT device is turned on when the capacitor voltage reaches the turn-on threshold (V_{turnon}) and it is turned off when its capacitor voltage drops below the turn-off threshold ($V_{turnoff}$).

The AEM10941 [24] is an integrated energy management circuit designed by e-peas [23], for solar and thermal harvesters, which extracts DC power from up to 7-cell solar panels to simultaneously store energy in a rechargeable element (e.g., capacitor). This solution can supply the system with two independent regulated voltages, the low-voltage output (LVOUT) and high-voltage output (HVOUT). LVOUT generates 1.2V or 1.8V providing a maximum load current of 20mA whereas the HVOUT pin can generate from 1.8V to 4.1V providing a maximum current of 80mA. The evaluation board works with an input voltage ranging from 50mV to 5V, and starts harvesting energy at 380mV with an input power of only 3μA.

The board can be logically divided into four different modes depending on the capacitor voltage:

i) Voltage below $V_{turnoff}$ (Discharged), where the PMU only charges the capacitor without providing supply to the DUT (LVOUT and HVOUT are deactivated).
ii) Voltage between $V_{turnoff}$ and V_{turnon}, where there are two possibilities. First, when the capacitor charges from $V_{turnoff}$ (Discharged), then the PMU only charges the capacitor without providing supply, and the second one, when the capacitor already reached V_{turnon} (Ready-Charged), then the PMU provides the output voltage supply and charges the capacitor.
iii) Voltage between V_{turnon} and the maximum allowed (V_{max}) (Charged), where in the availability of harvesting current, the PMU charges the capacitor up to V_{max} and continues supplying the output voltage.
iv) Voltage above V_{max} (Overcharged), where the capacitor charging will be deactivated and the output voltage supplying is continued.

C. Different task scheduling approaches

In our work, we have considered and compared four different scheduling strategies:

i) Energy-aware task scheduling with flash storage (EAS+FLASH) that writes everything to flash, which is safer when the device fails, avoiding data loss and ensuring forward progress.
ii) Energy-aware task scheduling without flash storage (EAS-FLASH) that only stores everything in volatile memory, which works better if the device does not fail (e.g., when the amount of harvested energy is high enough). If the device turns off, the application cycle needs to restart from scratch.
iii) Energy-unaware task scheduling with flash storage (EUS+FLASH) that includes the flash operations, but does not consider the required voltage thresholds for every specific task. If a power failure occurs, the device will try to re-execute the same task again after the reboot, reading the application state from the flash.
iv) Energy-unaware task scheduling without flash storage (EUS-FLASH) that executes all tasks in a cycle without looking at the energy available, based on the flowchart, and without saving anything to the flash. If a power failure occurs, the application cycle will reset after the reboot, as there is nothing stored in the flash.

D. Task current consumption and execution time for different scheduling approaches

To perform accurate experiments, the current consumption (I_s) and execution time (t_s) of the different states of the device, such as sensing, flashing operations, or transmitting data, are required. The real experiments, validation of our energy-aware task scheduler, as well as a comparison between different task scheduling approaches, were performed using

TABLE I: Current consumption and time values of nRF52840 DK board used with the EE

State	Current Consumption					Execution Time				
	APP	EAS - FLASH	EAS + FLASH	EUS - FLASH	EUS + FLASH	APP	EAS - FLASH	EAS + FLASH	EUS - FLASH	EUS + FLASH
Temperature measurement	0.61 mA	0.66 mA	1.38 mA	0.64 mA	1.37 mA	1.15 ms	1.96 ms	12.31 ms	1.91 ms	11.76 ms
Store 1	1.31 mA	1.4 mA	1.4 mA	1.37 mA	1.37 mA	5.49 ms	7.39 ms	7.39 ms	6.81 ms	6.81 ms
Store 2	1.31 mA	1.4 mA	1.51 mA	1.37 mA	1.46 mA	5.49 ms	7.39 ms	11.98 ms	6.81 ms	11.61 ms
Compute	1.33 mA	1.41 mA	1.54 mA	1.37 mA	1.48 mA	6.31 ms	7.64 ms	12.1 ms	7.22 ms	11.81 ms
Transmit	0.31 mA	0.34 mA	0.57 mA	0.32 mA	0.55 mA	32.43 ms	33.81 ms	35.75 ms	32.74 ms	34.7 ms
Blinking LED	5.87 mA	5.97 mA	5.97 mA	5.95 mA	5.95 mA	502.6 ms	508 ms	508 ms	504.7 ms	504.7 ms

the nRF52840 DK board [25] on which a Bluetooth mesh application was implemented. The current consumption (I_s) and execution time (t_s) of the different states of the nRF52840 DK board were obtained with a Nordic Power Profiler Kit II [26], a standalone unit that can measure the current level on all Nordic DKs, in addition to external hardware. The nRF52840 DK board operates in nRF only mode, which tries to isolate the chip on the board as much as possible, decreasing the total current consumption and making the low-power application more suitable for battery-less IoT devices.

Table I shows the current consumption and duration for different states of the nRF52840 DK board, including all considered scheduling approaches, as well as the scenario without scheduler, used with the EE. These values were measured at 3.3V, using a TX power of 8dBm and considering three parts, (i) the application part (APP) that only includes the defined tasks, (ii) scheduler part without the additional flash operations (EAS-FLASH and EUS-FLASH) that includes the defined tasks, but also shows the impact of the scheduler's overhead, and (iii) the scheduler part with the additional flash operations (EAS+FLASH and EUS+FLASH) that includes the defined tasks and scheduler's overhead, but also shows the impact of the flash's overhead. Compared to the application part, the measured current consumption and execution time are higher when the scheduler and flash features are also included. The scheduler adds some extra operations, such as (i) selects the task with the highest priority, (ii) removes it from the task list when it is finished, and (iii) occupies new places in the list adding all dependent child tasks related to the executed one. In case when the energy-aware scheduling approaches (EAS+FLASH and EAS-FLASH) are used, before every task execution, the device needs to measure the voltage in order to check if the required voltage threshold is satisfied, which also consumes some energy (around 110μA). Including the additional flash functions before and after each task will increase the current consumption, but also the availability of data if a power failure occurs.

The lowest energy cost task is the temperature measurement, but adding the scheduler features, the execution time and current consumption will increase, especially when the additional flash operations are added. There are two different store states: (i) before the required number of temperature samples is collected, the additional flash's overhead is not included in any scheduling approach as the store task in this stage does not have any dependent child task(s) to be added, and (ii) after the required number of temperature samples is collected, the compute task will be added to a list and stored, which requires additional flashing operations (EAS+FLASH and EUS+FLASH). The compute task calculates the average temperature, which does not consume much energy, but it also needs to store that value after the calculation. When this task is finished, its child task(s) will be added and stored, which again includes the additional flash part for the specific scheduling implementations (EAS+FLASH and EUS+FLASH). Before the transmission, the scheduler (EAS+FLASH and EUS+FLASH) will again check the flash status, and select the transmit task as the next one, which will add some additional current consumption. The final task in the cycle is the LED task, and as it does not have any dependent task that must be stored in the flash or added to the list, all scheduling approaches show almost the same results. In this case, compared to the application part, the overhead can only occur from the scheduler and voltage measurements, depending on the implemented scheduling strategy.

Using the scheduling approaches with additional flash operations (EAS+FLASH and EUS+FLASH) ensures a safer task execution, avoiding data loss and preserving forward progress if the device fails, but also increases the current consumption and execution time of tasks. This overhead can be reduced by removing the extra flash operations, and based on that we considered the additional version of each scheduling approach (EAS-FLASH and EUS-FLASH). The EUS-FLASH solution is the lowest energy cost compared to other considered approaches, but also with the highest risk of power failures and data loss to occur. On the other side, the energy awareness aspect of the EAS-FLASH solution that allows it to avoid power failures as much as possible, makes this approach a good alternative to the EAS+FLASH in terms of the lower current consumption.

The current consumption and execution time for different states of the nRF52840 DK board used with the Intelligent Power Management Unit based on the AEM10941 chip are presented in Table II. These values were obtained at 1.8V, based on the low-voltage output (Section IV-B2), using a TX power of 8dBm and including all considered scheduling approaches. In this case, the device is not able to directly read the voltage from the power management unit, as is a case when the EE is used. In order to enable our device to read the voltage, an additional voltage divider with resistors is added that increases the current consumption of this task (around 1.5mA), which in the end has the impact on the total current consumption of each task when the energy-aware task schedulers are used (EAS+FLASH and EAS-FLASH). This full setup will be described in more detail in Section V-B.

TABLE II: Current consumption and time values of nRF52840 DK board used with the Intelligent Power Management Unit based on the AEM10941 chip

State	Current Consumption					Execution Time				
	APP	EAS - FLASH	EAS + FLASH	EUS - FLASH	EUS + FLASH	APP	EAS - FLASH	EAS + FLASH	EUS - FLASH	EUS + FLASH
Temperature measurement	1.12 mA	1.4 mA	2.1 mA	1.2 mA	1.61 mA	1.22 ms	1.73 ms	12.83 ms	1.45 ms	11.81 ms
Store 1	1.9 mA	2.16 mA	2.16 mA	1.97 mA	1.97 mA	5.84 ms	7.69 ms	7.69 ms	7.26 ms	7.26 ms
Store 2	1.9 mA	2.16 mA	2.43 mA	1.97 mA	2.24 mA	5.84 ms	7.69 ms	12.23 ms	7.26 ms	11.2 ms
Compute	1.94 mA	2.18 mA	2.58 mA	2.02 mA	2.3 mA	6.08 ms	7.75 ms	12.4 ms	7.33 ms	11.2 ms
Transmit	0.47 mA	0.61 mA	0.84 mA	0.51 mA	0.72 mA	28.41 ms	34.31 ms	36.5 ms	32.65 ms	34.38 ms
Blinking LED	0.22 mA	0.48 mA	0.48 mA	0.29 mA	0.29 mA	502.6 ms	506.1 ms	506.1 ms	504.8 ms	504.8 ms

E. Harvesting current

In addition to the current consumption (I_s) and execution time (t_s) of the different states of the device, we also need to know the harvesting current (I_h). Using these values, the required voltage threshold calculations can be performed accurately, reducing the possibility of power failures. Considering these parameters for every specific task, the energy-aware task scheduler knows when to start with the execution.

There are different ways how to calculate the required voltage threshold V_0 based on Equation 1 presented in [19]:

$$V_0 = \frac{V_{min} - I_h \rho(I_s)(1 - e^{(\frac{-t_s}{\rho(I_s)C})})}{e^{(\frac{-t_s}{\rho(I_s)C})}} \tag{1}$$

where V_L is replaced with V_{min}, the minimum voltage on which the task ends, and I_h is the estimated harvesting current. Based on how I_h is estimated, we can obtain different estimates for V_0, referred to as V_0':

i) Worst case estimate (I_h = 0) -

$$V_0' = \frac{V_{min}}{e^{(\frac{-t_s}{\rho(I_s)C})}}, V_0' > V_0 \tag{2}$$

As a baseline for comparison, the harvesting current can simply be estimated as 0, resulting in the highest possible voltage threshold, but the lowest chance of a power failure occurring. However, this approach shows some disadvantages, especially with higher harvesting currents when the required amount of energy for task execution can be collected much faster. In this case, the device will waste time waiting to reach the worst-case threshold, even if a lower value can ensure the successful task execution, affecting the data freshness and missing deadlines.

ii) Perfect prediction (I_h = known) -

$$V_0' = V_0 \tag{3}$$

Considering the perfect prediction, the harvesting current is constant and defined before the experiment starts, which makes the calculation of required voltage thresholds much easier. However, it is not realistic to assume the future short-term harvesting current is known, as it fluctuates rapidly over time.

iii) Predicted estimate (I_h') - energy harvesting predictions can help determine after how long the energy-aware scheduler will have the required energy to execute a task. However, there are still two possible scenarios: (i) if the predicted harvesting current I_h' is higher than the real one, the calculated voltage threshold V_0' will be lower than required, and a power failure will occur, and (ii) when the predicted harvesting current I_h' is lower than a real one, the calculated voltage threshold V_0' will be higher than required, which is not optimal.

In this work, we have performed experiments both considering a perfectly predicted I_h, and the worst-case estimate approach. We did not consider a predictive approach, due to the difficulty in accurately predicted harvested energy. We consider this a separate topic, and subject of future work.

The electrical circuit model of a battery-less IoT device using a current source energy harvester, which was presented in our previous work [19], is a simplified model of the EE, and with it we are able to get a match in terms of the capacitor behavior. On the other side, there are some differences between the Intelligent Power Management Unit based on the AEM10941 chip and EE. When the Intelligent Power Management unit based on the AEM10941 chip is used, (i) the supply voltage to the DUT is constant, and (ii) there is different charging/discharging behavior of the capacitor depending on if the voltage is supplied or not, or in which mode the board operates (Section IV-B2). Based on that, we are not able to get a perfect match comparing with our mathematical model, but we can still use it to determine the approximate value that will be used as the required voltage threshold.

V. Results and Discussion

In this section, we present results and validation of our energy-aware task scheduling approach described in Section III based on real experiments and the task voltage thresholds derived in Section IV. In our scenarios, we have considered two devices, a battery-less low-power node (nRF52840 DK) on which the application along with the energy-aware task scheduler is implemented, and a constantly powered server/receiver node (nRF52840 DK) to receive the sent average temperature values. In order to enable the communication, both devices are equipped with a Bluetooth Low Energy (BLE) radio supporting the Bluetooth mesh features.

Before they can participate in a Bluetooth mesh network, both nodes must be provisioned. The provisioning process is used for adding devices to the mesh network, and it is performed by the provisioner, which can be the same board as we mentioned above running the provisioning code, or a smartphone on which the required nRF Mesh application [27] is installed. In our case, the second option was used,

TABLE III: Experimental setup with the EE

Parameter	Symbol	Value
Turn-off Voltage	$V_{turnoff}$	1.8 V
Min Voltage	V_{min}	2 V
Max Voltage	E	3.5 V
Capacitance	C	(4.7, 10, 20) mF
Harvesting Current	I_h	17.09 - 85.47 μA
TX Power	TP	8 dBm
Duration	T_{exp}	160 - 250 s
Sensing periodicity	t_{temp}	1 s
Voltage check periodicity	t_{vol}	1 s
Required num. of temp. samples	N_{sample}	3

TABLE IV: Application task priorities

Task	Priority
Temperature measurement	3
Store	5
Compute	8
Transmit	10
Blinking LED	4

where devices have been provisioned before the program started. During the provisioning task, loading and parsing of the metadata (JSON files) can be done.

A. Non-Intelligent PMU (Environment Emulator)

1) Experimental setup: Based on the defined IoT sensor application and manually set configuration of the environment emulator, we defined and performed different experiments to validate our energy-aware task scheduling approach. Table III lists the general parameters used in our EE experiments. The minimum operating voltage V_{min} is equal to 2V, and the maximum allowed voltage based on EE constraints is 3.5V. The device collects a new temperature value every 1 second, and after 3 measurements the next task can be executed. Using the mathematical model described in our previous work [19], the required voltage threshold for each defined task can be calculated exactly considering the set capacitor size, harvesting current, as well as measured current consumption and execution time. The device needs to wait until the voltage threshold is satisfied to avoid a power failure from occurring. In case that the measured voltage is below the threshold, the device sleeps for a predefined time, which is in our case set to 1 second, and checks the voltage again. Our energy-aware task scheduler was configured to stay above 2V, with a safety margin, as the turn off voltage is equal to 1.8V. This can be done by calculating the voltage threshold for each task with V_{min} set to 2V. As mentioned before, all tasks are selected based on their priorities, which are summarized in Table IV. The priority has been set based on the type of application task; measuring the temperature is not as critical as computing the average value when the required number of samples is collected. After each sensing task, the temperature value must be stored in the flash, which makes this task higher prioritized than the periodic measurement. The transmit task is the task with the highest priority and will be executed first whenever an average temperature is available. In the end, the application cycle is finished by blinking the LED.

For the EE experiments, we have considered two main strategies: (i) the constant harvesting current during the full time of the experiment, and (ii) we defined different harvesting currents and switched between them every 35 seconds during the experiment in order to see if our scheduler is able to dynamically react to environmental changes. Also, it must be noted that using the EE, experiments start at the maximum allowed voltage with a fully charged capacitor, and the harvester starts to provide energy after 5 seconds, which can affect the number of executed cycles at the start of the experiment before the voltage drops to the steady state (around 2.05V) in all considered cases.

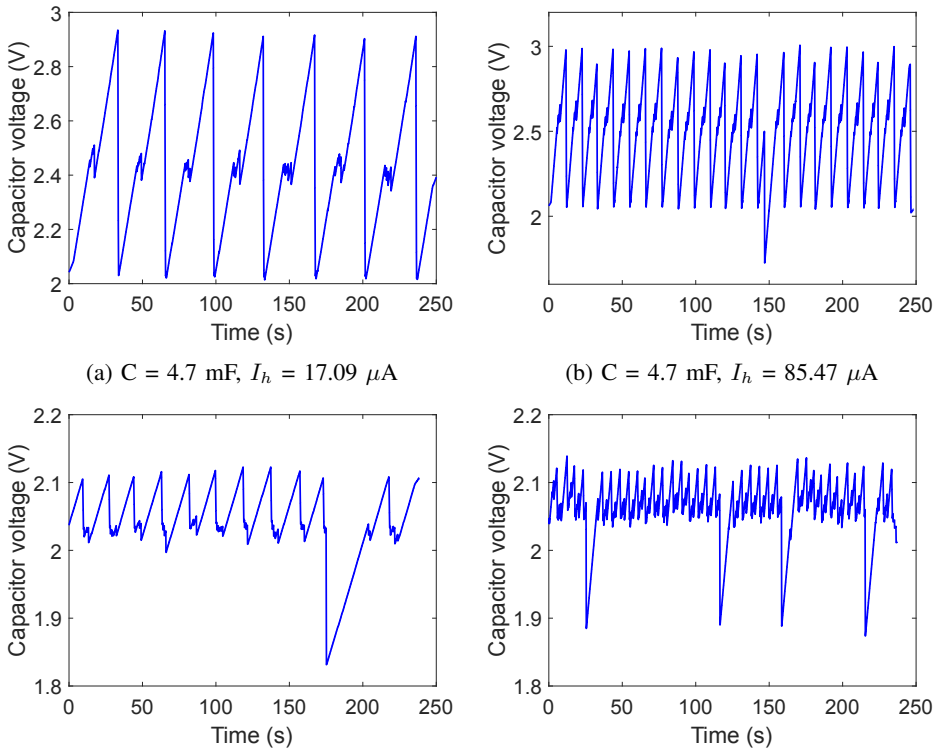
2) Constant harvesting current: In this approach, the known harvesting current for calculating the required voltage thresholds of the tasks is considered, which means that the perfect prediction case is used. This value is constant and defined before the experiment starts. Figure 4 shows the capacitor voltage changes when the device executes different tasks described in Section IV, considering different capacitor sizes and harvesting currents. The most power-hungry task is the LED task, which consumes the highest amount of energy and takes much more time to complete compared to other tasks that we considered in this work. As the capacitor size increases, lower harvesting currents can be used. It is expected that using a smaller capacitor will lead to a reduction charging time, resulting in more application cycles to be completed. However, this is not the case, as can be clearly seen when comparing Figure 4a and 4c or 4b and 4d. This is because a smaller capacitor requires a much higher voltage threshold which in turn results in longer charging times.

When using a smaller capacitor (cf. Figure 4a) with a lower harvesting current, such as 17.09 μA, the device needs to sleep for a long time in order to reach the required threshold (above 2.9V) to perform the LED task, which is the highest energy consumer. This task will deplete almost all the available energy, and force our device to sleep again in order to reach the threshold of the sensing task and start a new cycle. The impact of this behavior can be seen in a total number of successful cycles (7 cycles in 250 seconds) at the end of the experiment.

As the harvesting current increases (cf. Figure 4b), the required thresholds are still high (all above 2.4V), but the number of successful cycles increases (22 cycles in 250 seconds). In this case, the device needs less time to collect enough energy for performing the LED task, and meet the next task constraints after finishing it.

As the capacitor size increases (cf. Figure 4c, 4d), the required voltage threshold for each task will decrease. It can still be observed that the LED task consumes the most energy compared to the other tasks, but now the threshold is much lower (below 2.15V) and the device can reach it faster. Even when the voltage drops below 2V, the capacitor charges fast, and the device needs less time to reach the next threshold. Increasing the capacitor size and harvesting current, the device will show better performance, until it reaches the point when there is enough energy to perform all tasks without voltage variations.

Figure 4b, 4c, and 4d show some unexpected behavior where huge voltage drops occurred. This can be explained considering the added scheduler features which are mostly related to the flash erase and write functions. In our case, the



(a) C = 4.7 mF, I_h = 17.09 μA

(b) C = 4.7 mF, I_h = 85.47 μA

(c) C = 20 mF, I_h = 17.09 μA

(d) C = 20 mF, I_h = 85.47 μA

Fig. 4: Capacitor voltage behavior when executing different tasks considering different capacitor sizes and constant harvesting currents

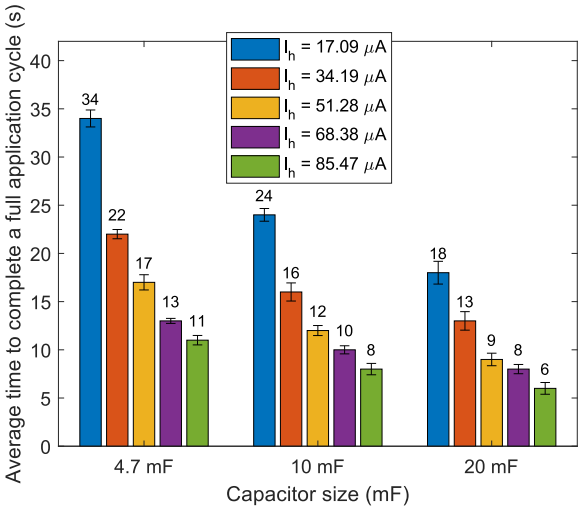


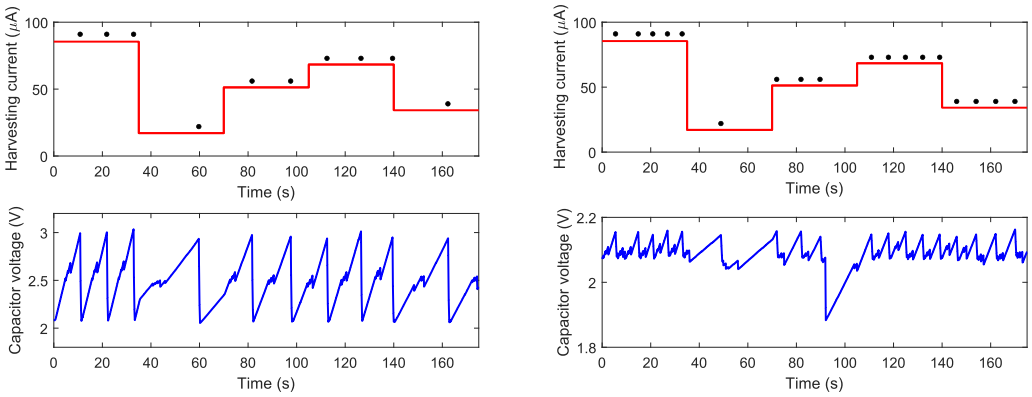
Fig. 5: Average time to complete a full application cycle considering different capacitor sizes and harvesting currents

device needs to execute the flash operations more frequently, before and after each task execution in order to prevent data loss, which can occasionally take more time than expected preventing the chip from going to sleep mode and increasing the total current consumption. This behavior causes delays in flash operations and holds the device in the ON state discharging the capacitor very fast. The voltage drop depends on the provided harvesting current, capacitor size, and the starting voltage at which this behaviour occurs affecting the total number of executed application cycles. To ensure that the device will not turn off when this behavior occurs, the higher current consumption and execution time are considered for calculating the required voltage threshold, taking into account this worst case energy consumption which occurs sporadically.

The average time needed to complete a full application cycle (cf. Figure 5), which starts with temperature measurements and ends with the LED task if all constraints are satisfied, as described in Section IV (cf. Figure 3), depends on the capacitor size and harvesting current, as well as the calculated required voltage for every considered application task. We considered different capacitor sizes, and tested their behavior with different harvesting currents.

Considering smaller capacitors such as 4.7mF, the needed average time to complete a full application cycle is longer compared to the other two. The reason for that is because the required voltage thresholds that the device needs to reach in order to execute every task are higher, which takes more time and affects the final number of performed cycles. The same behavior as with the other two capacitor sizes is observed, increasing the harvesting current the average needed time for one cycle decreases and the device shows better performance.

3) Non-constant harvesting current: In the second experiment, the harvesting current was varied throughout the experiment. We switched the harvesting currents every 35 seconds, considering the values presented in Table III, to check if our energy-aware task scheduler can react to environmental



(a) C = 4.7 mF (b) C = 20 mF

Fig. 6: Capacitor voltage behavior when executing different tasks considering different harvesting currents during the experimental time

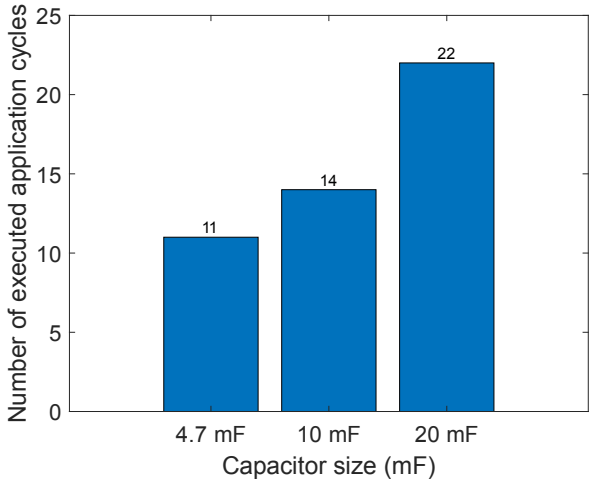


Fig. 7: Number of executed application cycles for different capacitor sizes considering variable harvesting current over time

changes. Figure 6 shows the capacitor voltage changes during the task executions for different capacitor sizes and harvesting currents, which were changed during the experimental time, starting with the highest value of 85.47μA.

In both shown examples (cf. Figure 6a, 6b) it can be seen that our scheduler is able to react to environmental changes, and adapt the execution to the new situation. The capacitor acts based on the provided harvesting current and set voltage threshold, and as the harvesting current increases more cycles will be executed. For the larger capacitor (cf. Figure 6b), the device shows better performance executing more cycles, which are presented as black dots on the graph, in almost all cases except when the harvesting current is the lowest. In this case, the device executed only one task cycle for both capacitor sizes due to the higher required voltage threshold which requires almost all available time to be reached. In all other cases, the device with the larger capacitor performs better and executes more application cycles, due to the lower required threshold that can be reached faster compared to when a smaller capacitor is used (cf. Figure 6a).

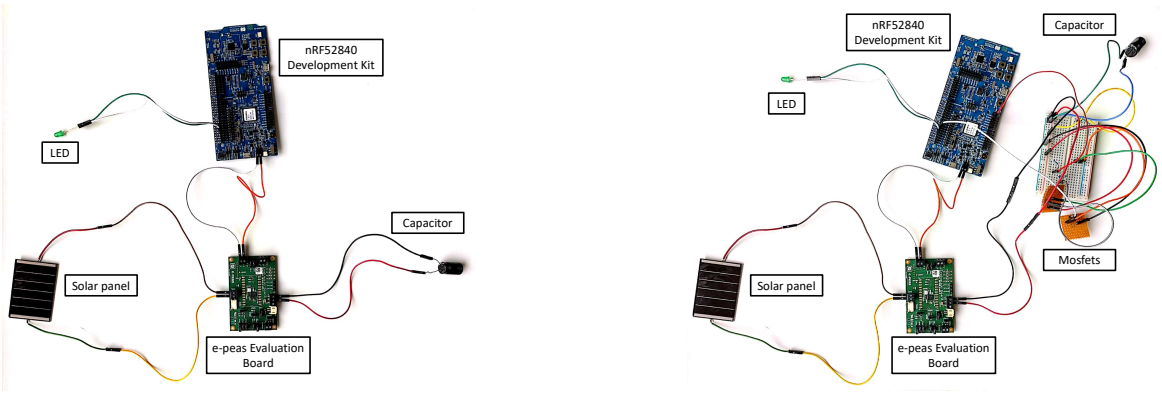
The final comparison between different capacitor sizes related to the total number of a full application cycles is shown in Figure 7. Again, the larger capacitor of 20mF shows the best performance allowing our device to execute the highest number of application cycles due to the behavior and reasons mentioned above. As the capacitor size decreases, the required voltage threshold for each task will increase taking more time to charge and allowing our device to execute the tasks less frequently.

TABLE V: Experimental setup with the Intelligent PMU based on the AEM10941 chip

Parameter	Symbol	Value
Turn-off Voltage	$V_{turnoff}$	2.8 V
Turn-on Voltage	V_{turnon}	3.67 V
Max Voltage	V_{max}	4.5 V
Capacitance	C	(4.7, 10) mF
Harvesting Current	I_h	(20, 40) μA
TX Power	TP	8 dBm
Duration	T_{exp}	5400 s
Sensing periodicity	t_{temp}	1 s
Voltage check periodicity	t_{vol}	1, 10 s
Required num. of temp. samples	N_{sample}	3

B. Intelligent PMU based on the AEM10941 chip

1) Experimental Setup: Based on the defined IoT sensor application (cf. Figure 3) and manually set configuration of the e-peas evaluation board, we performed different experiments to validate our energy-aware task scheduler and compare it with different scheduling approaches. Table V lists the general parameters defined in our experimental setup with the e-peas board. The turn-off voltage, below which the device cannot operate, is set to 2.8V, and the maximum allowed voltage, V_{max}, is equal to 4.5V. The battery-less IoT device will turn on when the voltage threshold of 3.67V is reached. These voltage thresholds can be set in different operating modes from a range that covers most application requirements through three configuration pins, without any dedicated component [24].



(a) Energy-unaware (b) Energy-aware

Fig. 8: Experimental setup using the e-peas evaluation board

Compared to the approach where the environment emulator is used, for these experiments we have considered the real energy harvesting environment with solar panels. The Panasonic AM-5608 [28] outdoor solar panel consisting of 6 amorphous silicon solar cells was used. As the sunlight intensity is unpredictable, and the harvesting current can change at any moment, the controllable setup with artificial light which is placed at some distance above the solar panel was designed, in order to fairly compare different scheduling approaches. We have used the Philips 5.5W White LED lamp (470lm) as a light source. The experiments are performed in a completely dark room where the light is only produced by the bulb. All tasks are selected based on their priorities, which were summarized in Table IV.

For the Intelligent PMU based on the AEM10941 chip experiments, we have considered four different scheduling strategies presented in Section IV-C. In both energy-unaware scheduling approaches, the device tries to execute the next task immediately after the previous one, which does not guarantee that the task will be finished before a power failure occurs, as the required voltage thresholds on which the task executions should start are not taken into account. The hardware setup for the energy-unaware scheduling approaches is shown in Figure 8a. In contrast, using the energy-aware scheduling approach, the battery-less IoT device needs to be able to obtain the current voltage on the capacitor, in order to compare it to the calculated voltage threshold of each task. In Figure 8b, the energy-aware experimental setup with the e-peas board is shown, where an additional voltage divider is added to enable our device to read the voltage on the capacitor, and based on the permitted range of Nordic GPIO act accordingly. As the voltage measurement circuit contains additional resistors, the current consumption will increase. To reduce this, MOSFETS are used as a circuit switch. In this way, the harvested energy can be used better by determining the usable energy capacity stored in the capacitor, and the IoT application can be modified to act in an energy-aware fashion.

In both cases, we have used three power pins on the e-peas evaluation board:

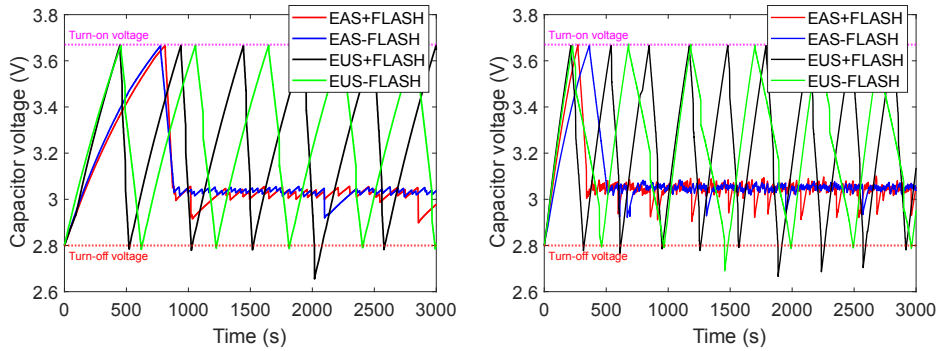
i) BATT pin that is the connection to the energy storage element, which is in our case the capacitor, and cannot be left floating.
ii) SRC pin that is the connection to the harvested energy source, which is in our case the solar panel.
iii) LVOUT pin that presents the output of the low voltage LDO regulator. This pin is used as the connection to our nRF52840 DK board.

2) Comparison between Approaches: We compare our energy-aware task scheduling approach with the energy-unaware scheduling strategies using the defined IoT application, and the e-peas evaluation board setup that considers the real energy harvesting environment. In our experiments, we have followed the behavior of the device under different implemented scheduling strategies (cf. Figure 9) in terms of the time the device is turned on and the number of power failures (cf. Figure 10), and the total number of full application cycles (cf. Figure 11).

Considering the energy-aware task schedulers, with (EAS+FLASH) and without additional flash operations (EAS-FLASH), when a smaller capacitor such as 4.7mF (cf. Figure 9a and 9b) is used, power failures can be avoided completely. In both cases, defining the required voltage thresholds for each application task, the device knows when the needed amount of energy is stored and the task can be successfully executed. If the required voltage is not reached, the device goes in sleep mode, consuming a very low power (around 10μA), and checks the voltage again after the predefined time t_{vol} (1 and 10 seconds).

For a larger capacitor such as 10mF, considering our energy-aware task scheduling approaches (cf. Figure 9c and 9d), we have noticed the same behavior as in the case the capacitor of 4.7mF is used. Using both of our energy-aware task schedulers, the device stays on for the entire experiment duration (cf. Figure 10a and Figure 10b), avoiding power failures in all considered cases (cf. Figure 10c and Figure 10d). However, this is possible only if we perfectly determine the value of the harvesting current, which is the case in our controllable setup. In reality if this knowledge is not perfect, power failures could occur if the harvesting current (I_h) is overestimated (Section IV-E).

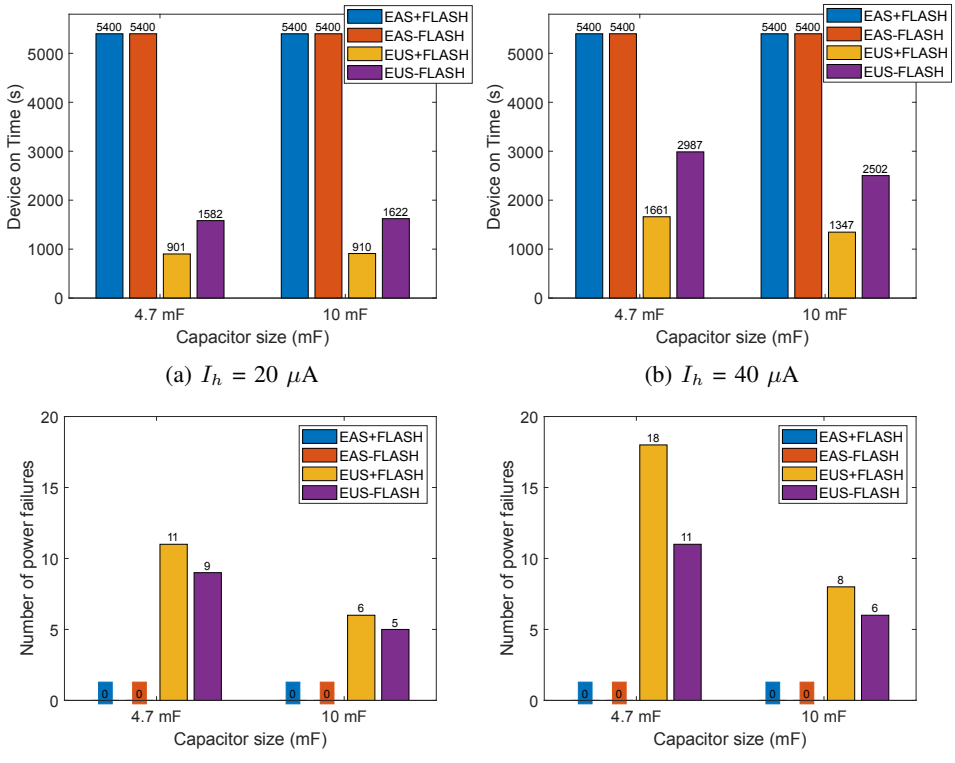
In contrast, both energy-unaware approaches (EUS+FLASH and EUS-FLASH) will try to execute tasks every time there



(a) C = 4.7 mF, I_h = 20 μA (b) C = 4.7 mF, I_h = 40 μA

(c) C = 20 mF, I_h = 20 μA (d) C = 20 mF, I_h = 40 μA

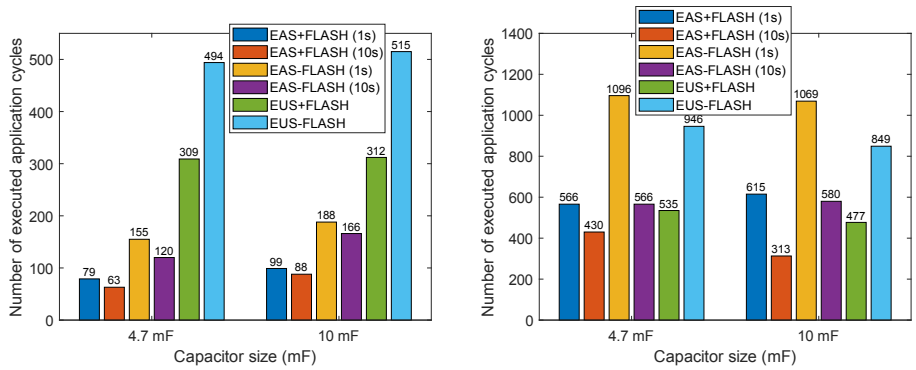
Fig. 9: Capacitor voltage behavior when executing different tasks considering different capacitor sizes and harvesting currents



(a) I_h = 20 μA (b) I_h = 40 μA

(c) I_h = 20 μA (d) I_h = 40 μA

Fig. 10: Total time the device is on and number of power failures for different task scheduling approaches considering different capacitor sizes and harvesting currents



(a) I_h = 20 μA (b) I_h = 40 μA

Fig. 11: Number of executed application cycles considering different capacitor sizes, harvesting currents, and voltage check periods

is energy available, without checking if that amount of energy is enough for the successful execution. After some time, the energy in the capacitor will drain and the device will turn off. After a power failure occurs, the device needs to wait until the turn-on voltage threshold is reached to repeat the task execution or reset the full application cycle depending on the selected energy-unaware approach. This can affect the data freshness and cause missed deadlines, or even complete data loss if the solution without flash is used. The total number of power failures depends on the capacitor size and harvesting current. As the capacitor grows smaller, it is able to perform more charge/discharge cycles, especially when the higher harvesting current is used (cf. Figure 9b), which increases the number of power failures (cf. Figure 10d).

A larger capacitor such as 10mF will take longer to charge and reach the turn-on voltage threshold, especially when the lower harvesting current is considered (cf. Figure 9c). However, since both energy-unaware solutions do not worry about the task energy cost, power failures are not avoided in this case either. The energy-unaware solution where the additional flash operations are added consumes energy very fast, due to the flash's overhead, causing the highest number of power failures (cf. Figure 10c and 10d). Even if the energy-unaware solution without the flash storing reduces this overhead and enables the device to be awake for a longer period (cf. Figure 10a and 10b), it is still not enough to manage and avoid power failures.

Finally, Figure 11 shows the number of full application cycles executed during the experiments, considering different capacitor sizes, harvesting currents, and voltage check periods (1 and 10 seconds), which are used in the energy-aware task scheduling approaches. Based on the obtained results, we have concluded that the capacitor size, harvesting current, as well as a period between two voltage checks, will determine the number of performed application cycles. If the period between two voltage checks is too long, the device will reach the required voltage threshold before the next measurement, wasting the rest of the time in sleep mode. Otherwise, if this period is too short, the device will check the voltage too often, consuming extra power, which can cause the task execution to be postponed. Therefore, we tested two different voltage check periods. The device measures the voltage every 1 or 10 seconds to check if the required threshold is reached in case that it does not have enough energy to perform the specific task. Between these two checks the device goes in sleep mode consuming around 10μA. This parameter is only relevant for the energy-aware solutions, as they consider the energy aspect when selecting and scheduling tasks, and different voltage check periodicity will not affect the total time the device is on or the number of power failures. Therefore, we do not show results for different voltage check periods in Figure 10.

For a lower harvesting current such as 20μA (cf. Figure 11a), the energy-unaware approaches, especially the solution without the additional flash operations, show the better performance compared to the energy-aware approaches. Using the EUS-FLASH solution the device consumes the lowest energy, as there are no flash operations, and voltage checking, which enables faster task execution until the turn-off voltage is reached and a power failure occurs. Since the implemented application is not too power-hungry, starting at turn-on voltage, which is in our case equals 3.67V, until it reaches the turn-off voltage threshold the device can execute a high number of cycles. This behavior leads the energy-unaware solutions to show the better results with this configuration compared to our energy-aware task schedulers. In contrast, considering our energy-aware approaches, power failures can be avoided, but the device operates around lower voltages (cf. Figure 9a, 9c). It needs to wait until the calculated voltage threshold is reached for each task in the cycle, which reduces the frequency of task execution, especially in case when lower harvesting currents are considered. When more frequently voltage checks are implemented (i.e., every 1 instead of 10 seconds), both energy-aware approaches show better performance, especially the EAS-FLASH solution where the additional flash operations overhead is reduced.

As the harvesting current increases (cf. Figure 11b), our energy-aware task schedulers start to show better performance compared to the energy-unaware solutions. The needed time for reaching the voltage thresholds decreases, which results in more frequent task executions. In contrast, the number of full application cycles using the energy-unaware solutions also increases, but as they are not capable of avoiding power failures, the device still needs to wait for charging periods.

The best results for both cases (4.7mF and 10mF) are shown by the EAS-FLASH solution, in case when the voltage check period is set to 1 second. As it reduces the flash's overhead, the current consumption is lowered, power failures are avoided, and the device does not waste time in sleep mode if the required threshold is already reached. When the voltage check period is set to 10 seconds, the device sleeps more missing the opportunity to execute tasks earlier, which results in a smaller number of performed cycles. Similar behavior can be observed with the EAS+FLASH approach, where the number of performed cycles is reduced due to additional flash's overhead. With the voltage check period of 1 second, the EAS+FLASH solution shows better performance compared to the similar energy-unaware implementation (i.e., EUS+FLASH).

VI. Conclusion

In this article, we presented an energy-aware task scheduler for battery-less IoT devices based on dependencies and priorities, which intelligently schedules the application tasks avoiding power failures. All tasks are characterized by an order, which is implemented as a parent/child relationship, and the task with the highest priority will be executed first, starting with the initial (parent) task. Each task can have multiple dependent (child) tasks, which will be selected only if their execution constraint is satisfied. As we considered energy awareness, calculating the required voltage thresholds for every specific application task will ensure safe task execution without power failures.

First, using an environment emulator and a Nordic nRF52840 board, we validated our scheduling approach, considering two main strategies. Our results showed that with properly defined task voltage thresholds, power failures can be avoided and more tasks can be successfully executed. Also, we have shown that our energy-aware task scheduler is able to react to changes in the harvested current.

Second, based on the defined IoT sensor application and an e-peas power management board, we performed different experiments considering a real energy harvesting environment with solar panels. As the sunlight intensity is unpredictable, and the harvesting current can change at any moment, a controllable setup with artificial light was used. We validated our energy-aware task scheduling approach against an energy-unaware scheduling strategy in terms of the total number of full application cycles, the time the device is awake, and number of power failures. Using the energy-aware task scheduler, the device is awake for the entire experiment duration, completely avoiding power failures. We have shown that with the properly defined voltage check period, our energy-aware scheduling approaches were able to execute more application cycles compared to their energy-unaware counterparts, when a relatively small capacitor of 10mF or less is used at harvesting current of 40μA.

Acknowledgment

Part of this research was funded by the Flemish FWO SBO S001521N IoBaLeT (Sustainable Internet of batteryless Things) project, the University of Antwerp IOF funded project COMBAT (Time-Sensitive Computing on Battery-Less IoT Devices) and the CERCA Programme, by the Generalitat de Catalunya.

References

[1] C. Delgado and J. Famaey, "Optimal energy-aware task scheduling for batteryless iot devices," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2021.

[2] S. Bose, B. Shen, and M. L. Johnston, "A batteryless motion-adaptive heartbeat detection system-on-chip powered by human body heat," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 11, pp. 2902–2913, 2020.

[3] P. Escobedo, M. Bhattacharjee, F. Nikbakhtnasrabadi, and R. Dahiya, "Smart bandage with wireless strain and temperature sensors and batteryless nfc tag," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 5093–5100, 2021.

[4] S. Sachdev, J. Macwan, C. Patel, and N. Doshi, "Voice-controlled autonomous vehicle using iot," *Procedia Computer Science*, vol. 160, pp. 712–717, 2019, the 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919317223

[5] A. Coates, M. Hammoudeh, and K. G. Holmes, "Internet of things for buildings monitoring: Experiences and challenges," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3102304.3102342

[6] J. Hester and J. Sorber, "The future of sensing is batteryless, intermittent, and awesome," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3131672.3131699

[7] B. Islam and S. Nirjon, "Scheduling computational and energy harvesting tasks in deadline-aware intermittent systems," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 95–109.

[8] A. Y. Majid, C. D. Donne, K. Maeng, A. Colin, K. S. Yildirim, B. Lucia, and P. Pawełczak, "Dynamic task-based intermittent execution for energy-harvesting devices," *ACM Trans. Sen. Netw.*, vol. 16, no. 1, February 2020. [Online]. Available: https://doi.org/10.1145/3360285

[9] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, October 2017. [Online]. Available: https://doi.org/10.1145/3133920

[10] K. S. Yildrim, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester, "Ink: Reactive kernel for tiny batteryless sensors," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–53. [Online]. Available: https://doi.org/10.1145/3274783.3274837

[11] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3131672.3131673

[12] A. Colin and B. Lucia, "Chain: Tasks and channels for reliable intermittent programs," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 514–530. [Online]. Available: https://doi.org/10.1145/2983990.2983995

[13] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: Association for Computing Machinery, 2011, p. 159–170. [Online]. Available: https://doi.org/10.1145/1950365.1950386

[14] M. Hicks, "Clank: Architectural support for intermittent computation," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 228–240. [Online]. Available: https://doi.org/10.1145/3079856.3080238

[15] J. Van Der Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 17–32.
[16] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 575–585. [Online]. Available: https://doi.org/10.1145/2737924.2737978
[17] F. Yang, A. S. Thangarajan, W. Joosen, C. Huygens, D. Hughes, G. S. Ramachandran, and B. Krishnamachari, "Astar: Sustainable battery free energy harvesting for heterogeneous platforms and dynamic environments," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '19. USA: Junction Publishing, 2019, p. 71–82.
[18] K. Maeng and B. Lucia, "Adaptive low-overhead scheduling for periodic and reactive intermittent execution," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1005–1021. [Online]. Available: https://doi.org/10.1145/3385412.3385998
[19] A. Sabovic, C. Delgado, D. Subotic, B. Jooris, E. De Poorter, and J. Famaey, "Energy-aware sensing on battery-less lorawan devices with energy harvesting," *Electronics*, vol. 9, no. 6, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/6/904
[20] E. Çürük, K. S. Yıldırım, P. Pawelczak, and J. Hester, "On the accuracy of network synchronization using persistent hourglass clocks," in *Proceedings of the 7th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ser. ENSsys'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 35–41. [Online]. Available: https://doi.org/10.1145/3362053.3363497
[21] J. de Winkel, C. Delle Donne, K. S. Yildirim, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 53–67. [Online]. Available: https://doi.org/10.1145/3373376.3378464
[22] P. De Mil, B. Jooris, L. Tytgat, R. Catteeuw, I. Moerman, P. Demeester, and A. Kamerman, "Design and implementation of a generic energy-harvesting framework applied to the evaluation of a large-scale electronic shelf-labeling wireless sensor network," *EURASIP J. Wirel. Commun. Netw.*, vol. 2010, 2010. [Online]. Available: https://doi.org/10.1155/2010/343690
[23] e-peas semiconductors, "Energy harvesting — making devices energy autonomous — e-peas," 2021. [Online]. Available: https://e-peas.com/
[24] ——, "Aem10941 solar harvesting — photovoltaic energy harvesting — e-peas," 2021. [Online]. Available: https://e-peas.com/product/aem10941/
[25] N. Semiconductor, "nrf52840 dk - nordicsemi.com," 2021. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk
[26] ——, "Power profiler kit ii - nordicsemi.com," 2021. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2
[27] ——, "nrf mesh - nordicsemi.com," 2021. [Online]. Available: https://www.nordicsemi.com/Products/Development-tools/nrf-mesh
[28] Panasonic, "Amorphous silicon solar cells amorphous photosensors," 2021. [Online]. Available: https://panasonic.co.jp/ls/psam/en/products/pdf/Catalog_Amorton_ENG.pdf

Adnan Sabovic received the M.Sc. degree in telecommunications engineering at Faculty of Traffic and Communications, University of Sarajevo, Bosnia & Herzegovina in 2018 and B.Sc. degree in telecommunications engineering at Faculty of Traffic and Communications, University of Sarajevo, Bosnia & Herzegovina in 2016. He is currently a PhD researcher with the University of Antwerp and imec, Belgium. His research interests lie in field of sustainable Internet of Things, energy harvesting, low power and battery-less communications, and energy optimization of IoT devices and their networks.

Ashish Kumar Sultania received the M.Sc. degree in Computer Science from the University of Tartu, Estonia, and Norges teknisk-naturvitenskapelige universitet, Norway in 2017 and B.E. in Information Technology from the University of Delhi, India in 2011. He is currently a PhD researcher with the University of Antwerp and imec, Belgium. His research focuses on optimizing energy consumption of IoT devices and their networks. Prior to starting his masters, he worked as a Senior Software Engineer at NXP Semiconductor, India (2011-2015).

Carmen Delgado received the M.Sc. degree in telecommunications engineering, the M.Sc. degree in biomedical engineering and a Ph.D. (cum laude) in Mobile Network Information and Communication Technologies from the University of Zaragoza, Spain, in 2013, 2014, and 2018 respectively. She joined the Internet Technology and Data Science Lab (IDLab) of the University of Antwerp, associated with imec, Belgium as a post-doctoral researcher in 2018. She is currently working in the i2CAT Foundation as senior researcher. Her research interests lie in the field of Internet of Things, resource allocation, energy harvesting, low power communications, energy modeling and performance evaluation of wireless sensor networks.

Jeroen Famaey is an assistant professor associated with imec and the University of Antwerp, Belgium. He received his M.Sc. degree in Computer Science from Ghent University, Belgium in 2007 and a Ph.D. in Computer Science Engineering from the same university in 2012. He is co-author of over 120 articles published in international peer-reviewed journals and conference proceedings, and 10 submitted patent applications. His research focuses on performance modeling and optimization of wireless networks, with a specific interest in low-power, dense and heterogeneous networks.

Lander De Roeck received the B.Sc. degree in Computer Science from the University of Antwerp, Belgium in 2021. He is currently pursuing the M.Sc. degree in Computer Science: Data Science and Artificial Intelligence at the University of Antwerp, Belgium.