First Steps Towards Unified Low-Power IoT Design: The "DYNAMIC" Framework

Jakub Lojda, Josef Strnadel, Pavel Smrz and Vaclav Simek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
Email: {ilojda, strnadel, smrz, simekv}@fit.vut.cz

***Abstract*—This paper presents the development of a framework named DYNAMIC (Dynamic Management Interface for Power Consumption), designed for the dynamic selection of alternative code implementations to minimize energy consumption and extend battery life. The proposed solution works effectively in situations where the device uses energy harvesting. In such cases, the method plans the device power consumption to match the available energy, minimizing the impact on device operation. It is important to note that each device utilizing our framework will be planning its energy budget in-field (and during run time), so each device is optimized within a few days based on its environment of usage. In the experimental section, we present simulation results of experimental power management algorithms for our framework. The results indicate that for our particular study and energy harvester, it was possible to dynamically adjust power consumption so that the user did not notice a significant difference in device performance while achieving a balanced energy budget – under simulated conditions and disregarding battery lifespan, potentially allowing for indefinite operation.**

***Keywords*—Internet of Things, Low Power, Energy Efficiency, DYNAMIC Framework, Electronic Design, Energy Harvesting.**

I. Introduction

During recent years, the word *Internet of Things* (IoT) gained a high popularity. The IoT means a new era of interconnected devices (often called "smart" devices) that cooperate fully autonomously on given tasks. In [1], the IoT is defined as *"An open and comprehensive network of intelligent objects that have the capacity to auto-organize, share information, data, and resources, reacting and acting in the face of situations and changes in the environment"*. A very common task for an IoT device is data acquisition and monitoring and also transmission of measured data into a safe data storage, usually a cloud computing platform. Wireless communication is, however, one of the very expensive activities when it comes to the energy consumption of a battery-powered device [2], [3]. This makes it an important part to be considered and managed when optimizing power consumption. The lack of energy can be partially solved by utilizing the so-called *Energy Harvesting* (EH) technologies [4], but still, in many cases, EH can provide only a very small amount of energy in a limited time.

The management of power consumption of low-power IoT devices involves various techniques to minimize power consumption while still maintaining the device's functionality. This can be achieved by making the device firmware aware of the available energy stored in a battery and the energy consumption required for the execution of different parts of the firmware. However, integration of the information of energy requirements into a code is a complex and challenging task, often resulting in a hard-to-maintain programming source code, possibly leading to inefficient power management and high maintenance efforts needed to maintain the programming code.

The work presented in this paper aims to present a novel framework to easily decouple the power management information and logic from the main logic of the firmware. This is because, generally, separation of code results in easier, maintainable, and more readable code [5]. In this case, this means easier optimization and adaptability of the code, allowing developers to implement power-saving strategies without modifying the firmware logic code. The framework is called *Dynamic Management Interface for Power Consumption* (DYNAMIC). The DYNAMIC framework focuses on C++ as the main implementation language. However, its key principles are transformable to other languages as well. It is important to note that the framework operates at the runtime and, as its name suggests, dynamically selects the optimal implementation based on the energy consumption constraints and the selected power management algorithm. In the experimental part of this paper, three different optimization strategies are shown that were simulated and are to be implemented into the DYNAMIC framework.

Even the language used for programming influences the energy efficiency of the resulting firmware [6]. In the literature, many achievements are shown in the field of various low-power and energy-efficient IoT devices. For example, in [7], a cooperative approach to minimizing the energy consumption of a complete network is shown. An approach to low-power *Over the Air* (OTA) firmware update was presented in [8]. Similarly to the previous approach, in [9], an OTA update is presented; however, this time, with a focus on updating TinyML models of machine learning, presented on an agricultural IoT scenario. A low-power management of power lines that are usually of-the-grid, using solar energy, is presented in [10]. Very energy-efficient implementation of an IoT gas meter is presented in [11]. An ultra-low power *Artificial Intelligence* (AI) platform is presented in [12]. A low-power AI platform for bird vocalization analysis is presented in [13]. A bus tracker IoT system is presented in [14]. All these undoubtedly very good achievements have only one common aspect: Each of the platforms solves its low-power approach in its own way, making it quite hard to update the power-management algorithms, not to mention a complete switch of the HW platform. A more generic approach is in [15], where a low-power multi-sensor system is presented.

Our research aims to address the lack of a standardized and unified approach to dynamic power management, particularly for IoT devices. The primary motivation for this research is to save significant resources in our environment, as [16] predicted that by 2025, 78 million batteries will be discarded **daily** only from IoT devices. Moreover, we want to make IoT devices more convenient and practical, as the need for battery replacement is a significant drawback in maintenance, especially in places with many such devices.

Our framework aims to address the following issues often present in the literature. To be fair, the cited literature does not primarily attempt to solve these problems. However, it serves as an example that current solutions for low-power IoT devices lack the following aspects, which justifies our efforts:

1) It is crucial not to combine power management algorithms with firmware application logic. This leads to unnecessary code duplication across different products and reduces the code's maintainability. Furthermore, it increases code complexity, making updates difficult.

⇒ In contrast to the methods commonly described in the literature, our approach explicitly separates power management logic, power management data (i.e., the data used for decision-making by the power management logic), and the firmware logic itself (i.e., device application algorithms). This separation, as implemented in our framework, allows for easier integration and modification of advanced power management algorithms without changing the core functionality of the device firmware. Our goal is to enhance maintainability and simplify power management integration into existing firmware algorithms.

2) Establishing a unified method for incorporating power management algorithms into existing firmware logic is very important.

⇒ A unified interface will allow portability of different algorithms across platforms, as only the hardware abstraction layer for switching power modes, etc., will need to be re-implemented when porting the framework to a new platform (microcontroller).

3) We strive to design and develop novel power management algorithms that achieve better results and also consider novel energy harvesting (EH) technologies.

⇒ Our focus is to reduce power consumption to extend battery life (i.e., improve long-term user experience) while maintaining the same level of functionality (i.e., short-term user experience). This can only be achieved by carefully optimizing device behavior (i.e., power management).

This paper presents the initial work on a framework that addresses the points mentioned above. This is our first publication on the topic. Our first step focuses on the design of the interface, which will be briefly introduced in this paper, along with the initial power management algorithms intended for use within the framework. Additionally, we will demonstrate a simulation of a thermometer utilizing these algorithms, followed by their evaluation. Naturally, we excluded algorithm ideas from our simulation experiments that did not yield satisfactory results, which highlights the importance of simulation when pre-evaluating initial ideas.

This paper is organized as follows. A simple case study in Section II presents the emerging framework approach. The experiments and achieved results are presented in Section III. The achievements are further analyzed in Section IV, with the intention to outline the next steps in the development of the framework. Section V concludes the paper and summarizes the possibilities of the following research in this field.

II. Power-Aware Design with DYNAMIC

The DYNAMIC framework is designed to assist both in the design phase and in the practical implementation of energy-efficient systems. Parameters for the framework settings can be fine-tuned using a simulation (more on that later in the experimental section of this paper). It is important to emphasize that the pseudocode provided in this paper is illustrative because, in the final version, the syntax may vary slightly however, but most of the framework's interfaces have already been implemented, so only small changes might be necessary.

The process of power-aware firmware design can be divided into three main phases: A) find the most power-hungry parts of the algorithm; B) measure energy demands for these most power-hungry parts; and C) alter the algorithm by inserting the measured data into the implementation.

A. *Locating Power-Hungry Parts of the Algorithm*

Before using the framework, it is essential to identify the energy-intensive parts of the algorithm. Typically, these include data sensing, data processing, and communication [3]. Each of these can be implemented in a less energy-intensive way, but this often results in a decreased quality of user experience. For example, less energy-demanding data sensing can be achieved by reduced data resolution; this can, however, lead to decreased accuracy of results. Similarly, for data processing, simplifying can reduce the informative value of the results. And lowering the frequency of communication can increase latency.

Code Snippet 1 shows a simple example of code for a temperature-monitoring IoT device. And its UML AD is shown in Figure 1.

```
#include "common.hpp"
uint8_t
    buf[COMMON::BUF_SIZE];
time_t slp =
    COMMON::DEFAULT_SLEEP;

int main() {
  for(;;) {
    measure(&buf);
    send(&buf);
    sleep(slp);
  }
}
```

Code Snippet 1. Example of code from a very simple temperature-monitoring device.

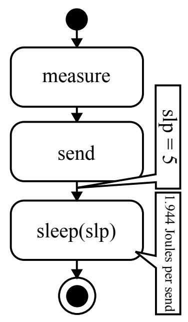

Figure 1. UML AD of one iteration of the example code.

If the UML AD of the algorithm is available, it can be extended to include different implementation options and configuration settings and their energy consumption; however, in our case, most of the energy can be conserved by increasing the length of sleep, in which the device is in a sleeping mode of a neglect consumption.

It is important that the HW itself is capable of measuring the current charge of the energy storage (battery or supercapacitor) to enable dynamic power consumption planning.

Additionally, the hardware must be able to measure time (e.g., time elapsed since startup). This is crucial for planning power consumption over time. When the hardware is in sleep mode, it cannot keep track of time, so a built-in *Real-Time Clock* (RTC) circuit is mandatory to ensure the framework can perceive time accurately.

B. Measuring Energy Demands

Subsequently, it is necessary to create implementation variants with lowered energy demands and measure the energy consumption of each implementation variant. This can be done experimentally [17] or estimated based on analysis. For tasks that are related to a special HW (e.g., communication using an external module), consumption can be measured directly (e.g., by connecting a precise power analyzer between the communication module and the power source). This allows for profiling energy consumption based on variables such as the number of packets sent, the distance from the base station, etc. The data obtained can also be incorporated into the UML AD as shown in Figure 1.

C. Modifying the Implementation

The next step involves integrating the collected data into the implementation. A simplified example can be seen in Figure 2, which demonstrates the previous implementation of a temperature monitoring device with possible modifications to the code.

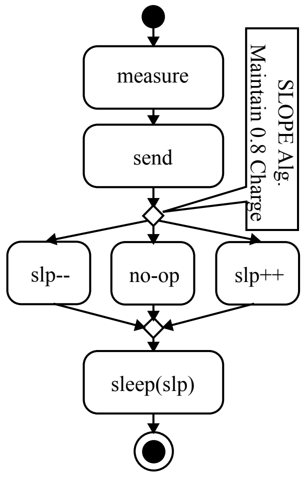

Figure 2. Part of the endless loop of the temperature-monitoring device UML AD modified to include data for power management planning.

To keep power demands data and the firmware implementation separate, our DYNAMIC framework employs advanced implementation techniques in C++, which extend the language with the necessary constructs.

The framework can select the power management style according to a chosen power-management algorithm. Currently, there are two experimental implementations, both of which are optimized for usage in combination with EH, although EH usage is not mandatory. Both of these target optimization of the power consumption while powering its impact on the user experience (precision, latency, etc.):

1) The “Amount” algorithm: This algorithm tracks the absolute amount of consumed energy and selects implementations based on that amount. A higher amount of consumed energy results in a less energy-demanding implementation executed and vice versa. These interval values for the decision must be set by the designer. This algorithm allows the setting of a border value of battery charge level, above which the energy saving is not enforced.
2) The “Slope” algorithm: This algorithm monitors the slope of the battery state development and selects energy-saving or non-saving variants based on this. When EH is implemented in the device, it can also select more energy-intensive implementations (when available) to potentially utilize the harvested energy that would be otherwise wasted (as limited by the battery capacity). This algorithm also allows the setting of a battery charge level, below which the planning is much more aggressive to conserve energy.

An example of code modified to use the “Slope” power-management algorithm be seen in Code Snippet 2.

```
#include "common.hpp"
#include "dynamic_lib/dynamic.hpp"
uint8_t buf[COMMON::BUF_SIZE];
time_t slp = COMMON::DEFAULT_SLEEP;
// capacity and pointers to measure functions
DYNAMIC_SOURCE(COMMON::BATT_CAPACITY_MAX,
    COMMON::curr_power, COMMON::curr_time);
int main() {
  for(;;) {
    measure(&buf);
    send(&buf);
    // dynamic selection of sending interval
    {
      // p. manag. alg., preferred batt. charge
      DYNAMIC_INIT(SLOPE, 0.8);
      PRICE_BLOCK(DYNA::INF, +0.15, {
          slp--;
      });
      // implicit no-op for uncovered interval
      PRICE_BLOCK(-0.15, DYNA::INF, {
          slp++;
      });
    } // point of the selected block execution
    sleep(slp);
  }
}
```

Code Snippet 2. Example of modified code using the DYNAMIC framework for power management.

As can be observed, at the beginning, the DYNAMIC library is included, which allows the use of the new constructs. Then, the battery capacity must be set, subsequently with methods used to obtain the current battery charge and the current in real-time in seconds (a value of seconds from the start of the system is sufficient). The following few lines of code remain unchanged. Then, the new part increasing the sleep interval is present within a block of alternative implementations. As can be seen, a variant of decreasing the sleep value is present as well. Using the Slope algorithm, it is distinguished which of these alternative blocks is executed. The actual execution of the selected block takes place exactly at the end of the block, meaning that before the ending bracket, the variable still holds its previous value. Within the block, there is also a definition of the management algorithm and the preferred charge level of a battery (0.8 equals 80% charge level in this case) to conserve enough energy for “worse times” when enough energy might not be provided by the EH.

III. Experiments and Results

To research appropriate types of algorithms for power management, we developed a simulation in Python3 language [18] with the help of the SimPy library [19], a process-based discrete-event simulation framework.

Again, we focus on modeling a simple device that monitors temperature. This device is equipped with a rechargeable battery (Li-Ion accumulator) of 1000 mAh, with a voltage range of 3.4 to 4.2 Volts. It also includes a communication module, where a single transmission consumes 1.944 Joules, which is approximately 0.15 mAh charge at 3.6 V. For simplicity, we neglect the sleep discharge current and the energy required for a single temperature measurement.

The device is initially set to transmit temperature data every 5 minutes. Additionally, we consider this same device with and without EH technology. We modeled EH technology capable of supplying 10 mA to the battery for a total of four hours within every 24-hour period, effectively increasing the battery charge by 40 mAh daily.

We had three use cases in total: One use case fully without any power management to serve as a reference and two additional use cases implementing the power management logic, one for each algorithm type.

No Power Management: In this use case, we created two models of the temperature monitor, one model without and one model with EH capability. Both are equipped with a battery but without using the DYNAMIC framework.

"Amount" Algorithm: In this use case, the device monitors the amount of energy consumed. When the battery charge decreases by more than 0.5 mAh, the transmission interval increases by one minute. The maximum interval is set at 60 minutes, beyond which no further increase occurs. In contrast, if the battery is recharged by more than 0.5 mAh, the transmission interval decreases, with a minimum value of one minute as the fastest interval. Consumption planning is dynamically adjusted with each execution of the scheduled code block. However, when measuring consumption, the fifth previous battery charge is considered (the framework implements a buffer of charge history), ensuring a slight smoothing of the process. The algorithm was tested on two simulation models with the same parameters: One for a device with and one without EH technology. The preferred battery charge level was set at 80% for both models.

"Slope" Algorithm: Here, the device monitors the slope of battery charge decrease or increase over time. For this algorithm to be applied, the device must be equipped with an RTC module, as the slope in degrees is calculated from the battery charge in time (similarly as on a chart). For devices without EH, if a decline is lower than -0.15 degrees, the sending interval increases by one minute per each sending interval to conserve energy. In contrast, to decrease the interval, an inclination of more than +0.15 degrees is required. The interval is adjusted until the slope of the charge is just between the interval of +0.15 and -0.15 degrees. It is important to note that a positive slope value indicates an increase in battery charge and vice versa; a zero-degree value means that the battery charge remains unchanged, and for instance, -90 degrees represents an ideal short circuit: Immediate battery discharge. For devices with EH, we adjusted these values to -1.75 and +1.75 degrees, respectively. In both cases, the preferred battery charge level was set at 80%.

The results of the battery charge simulations for each of the models for the first 60 days after the device was put into operation can be seen in Figure 3. The color of the line in each plot corresponds to the color palette on the right side of the chart and indicates user experience, specifically an increase (or decrease) in communication latency. This value is taken relative to the default setting, which was to transmit temperature data every five minutes. This means that a zero difference means the latency remains to be the expected one, being denoted by the green color. The red denotes a worse experience (i.e., higher latency), and the blue denotes an even better experience than expected (i.e., latency smaller than originally set).

As you can see from the simulation output, the worst performance was observed in the device without power management (top left column). It operated with expected latency for the first approximately 18 days, but then it fully discharged and shut down permanently. In contrast, the version with EH (bottom left column) discharged more slowly; however, it began to experience outages, which are barely visible as a red color in the bottom right corner of the corresponding plot. These outages are due to the device recharging during the EH activity, but the charge is insufficient, eventually leading to repeated shutdowns of the device.

In the middle column, you can see the device utilizing the "Amount" algorithm. For the version without EH (top center), there was a significant extension of lifespan (12 times longer than the version without planning, which corresponds to the upper limit of sending temperature data every 60 minutes). However, this also resulted in a considerable decrease in user experience. The device with EH (bottom center) performed notably better. The algorithm increases the transmission interval due to EH energy generation outages, with a condition ensuring that conservation measures are only implemented after the battery charge drops below 80%. As can be observed, this approach balanced energy intake and consumption, potentially allowing the device to operate indefinitely, of course, if we abstract from the battery and device wear (failure over time).

Devices utilizing the "Slope" algorithm are displayed in the right column. As observed for the device without EH (top right plot), the algorithm attempts to conserve energy because it detects continuous battery discharge. On the contrary, in the case of the device with EH (bottom right), there is a slight increase in latency, which remains under 10 minutes, stabilizing after a while and achieving a balanced battery charge at the desired 80%. This solution maintains a balanced intake and consumption of energy, implying that, provided the regular function of EH is maintained and battery degradation is ignored, the device could operate indefinitely. Additionally, acceptable latency is preserved so the user does not experience significant issues. When energy intake increases, the original sleep interval is restored, and when energy intake decreases, the interval is optimized. This behavior is observable in the plot, where during EH charging, the interval returns to its original state (i.e., the color changes from yellow-green to rich green in the plot sections when the battery charge rises).

IV. Analysis

The purpose of the DYNAMIC Framework, is to:

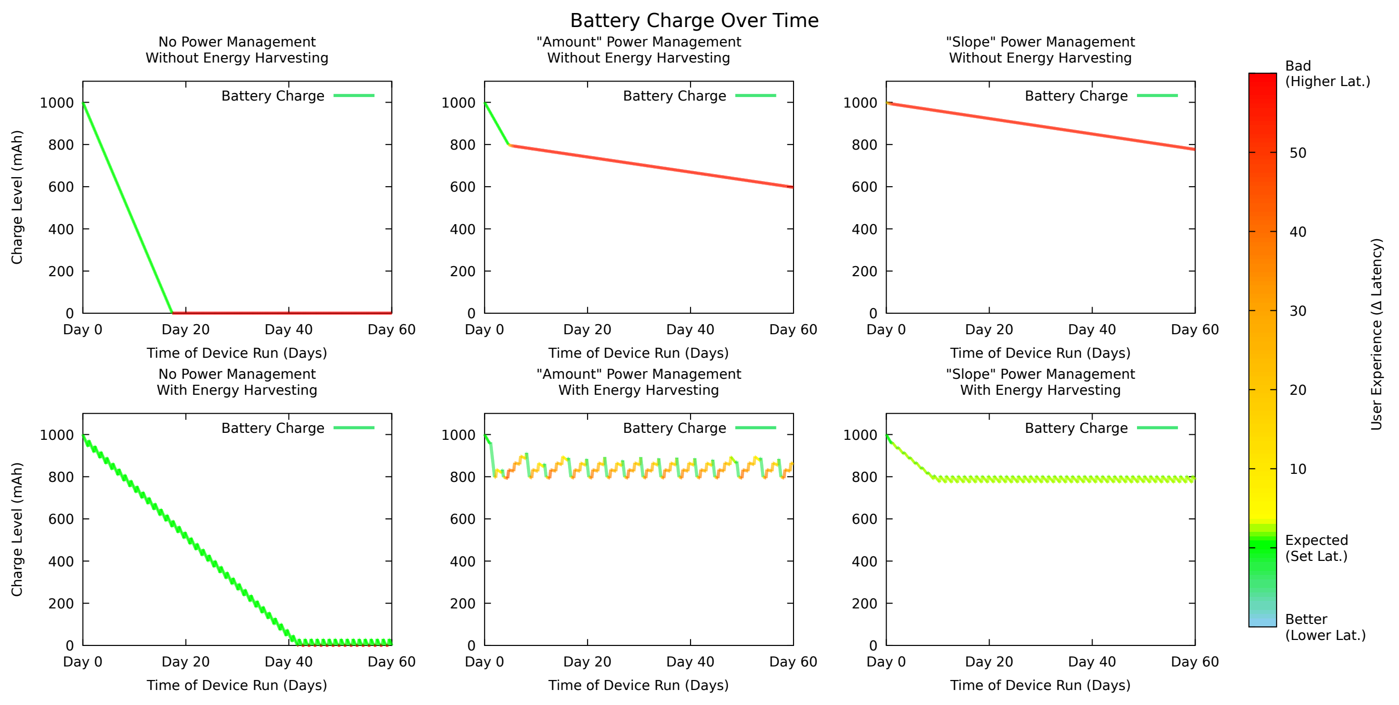

Figure 3. Results of simulations for the battery charge during runtime for the first 60 days; the color of the chart line indicates the user experience (change in latency from the expected setting).

1) simplify the integration of power management into existing firmware algorithms;
2) standardize the approach to implementing power management in new firmware algorithms;
3) separate the power management logic from the firmware logic in the code.

After presenting the interface, which was tested for feasibility in C++, we began designing algorithms for power management. For this, we employed a simulated approach, which allowed us to inspect a device's 60-day runtime in a significantly shorter period. During the implementation and analysis of the experimental results, we found out several fundamental properties that will potentially shape the future development of the framework:

1) Our solution stands out for its generality, offering a level of unification while still maintaining a natural separation between power management logic and the firmware algorithms (where algorithm logic allows for it).

2) Using relatively simple algorithms embedded into the simulated device, we were able to achieve a level of autonomy under certain conditions where the device could operate without battery replacement (of course, when abstracted from factors like battery aging or hardware degradation).

3) One limitation of our current solution is that it does not yet cover scenarios where the device itself provides additional data for power planning (e.g., a thermometer receiving input that temperature reporting is unnecessary, e.g., triggered by a button press or motion detection in a room). It would be interesting to incorporate additional data-driven responses into the framework. For example, a thermometer could only send temperature data when a change occurs while using the power management algorithm more intensively when the temperature fluctuates frequently. For some applications, it might also be helpful to plan based on external factors, such as whether a room is occupied, a building is in operation, or an off-season mode.

4) For more complex devices, it may be necessary to have multiple sections of code with power planning. In the future, we would like to test whether power management remains reliable if we apply multiple algorithms within a single device.

5) A current drawback is the need for specific hardware to inform the algorithms about the battery's charge level (such as a *Coulomb Counter* integrated circuit). While we would like to enable implementation in devices without this hardware, we have yet to find a way to maintain accuracy without such feedback.

6) Estimating device power consumption can sometimes be challenging. It might be interesting to explore the possibility of the algorithm learning the device's consumption pattern over the first few days of operation. This could be advantageous for specific applications, such as small-scale productions or devices where the final location cannot be predetermined. However, this somewhat contradicts our goal of keeping the framework as "lightweight" as possible.

7) With the previous point, exploring the possibility of automatically selecting the most suitable power management algorithms could be interesting. This would eliminate manual specification, allowing the framework to choose the best algorithm automatically. However, this level of automation appears to be a goal for distant future research.

V. Conclusions and Future Research

In this paper, we presented the initial steps towards designing a framework for power consumption planning, which allows for the abstraction of power management methods from the rest of the useful code (i.e., the application logic). The article further introduced two algorithms for this framework, which were simulated on a model of a simple wireless thermometer IoT device with and without EH technology.

The framework, once completed, will be usable to:

1) create devices with prolonged battery life easily and in a unified way, saving significant resources in our environment and
2) make IoT devices more convenient and usable, as the need for battery replacement is a significant drawback in maintenance, especially in places with many of such devices.

The results suggest that by choosing the appropriate algorithm and setting it correctly, it is possible to achieve a device that dynamically balances the energy budget while minimizing the negative impact on user experience (in our case, the latency of conducting and sending temperature measurements). An important aspect is that the device is planning its energy budget in-field, so each device is optimized within a few days based on its environment of usage.

Future research in this area could address the following questions:

1) Will the algorithms conflict if different ones are used within a single firmware?

2) Could the framework be improved by incorporating additional data into power planning, such as having the device send data only when a change in measured data occurs or when certain conditions are met? How could we integrate these external factors, like room occupancy, into the power management strategy?

3) Could the optimal algorithm be chosen automatically and dynamically during a firmware operation? Although we do not currently plan to implement this, it might be interesting because the computational cost of determining the best power management method is minimal compared to the potential energy savings from proper configuration.

4) Would it be feasible to make the energy demands measurements also on the device during run time? In such a case, the corresponding step during the design could be removed, potentially leading to easier development of power-efficient devices.

In conclusion, we would like to mention that the provided pseudocode of using the framework is still in development, and the usage might change slightly, but the main interface parts are already implemented, so any changes would be minor. The primary contribution of this paper is the simulation, which provides the foundation for implementing the algorithms into the framework. The simulation enabled us to design and optimize the initial set of energy management algorithms.

Acknowledgements

This work was supported by the Chips JU Project LoLiPoP-IoT (Long Life Power Platforms for Internet of Things), www.lolipop-iot.eu, grant agreement No. 101112286, which is jointly funded by the Chips Joint Undertaking and national public authorities.

References

[1] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 5, pp. 164–173, 2015.

[2] "IoT power budgets: IoT Connectivity Technologies and Batteries." [Online]. Available: https://caburntelecom.com/iot-connectivity-technologies-and-batteries/

[3] M. Enzinger, "Energy-efficient communication in wireless sensor networks," *Sensor Nodes–Operation, Network and Application (SN)*, vol. 25, no. 11, 2012.

[4] S. Priya and D. J. Inman, *Energy harvesting technologies*. Springer, 2009, vol. 21.

[5] J. K. Ousterhout, *A philosophy of software design*. Yaknyam Press Palo Alto, CA, USA, 2018, vol. 98.

[6] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: how do energy, time, and memory relate?" in *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*, 2017, pp. 256–267.

[7] S. R. Sarangi, S. Goel, and B. Singh, "Energy efficient scheduling in iot networks," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 733–740.

[8] O. Kachman and M. Balaz, "Optimized differencing algorithm for firmware updates of low-power devices," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2016, pp. 1–4.

[9] C. Nicolas, B. Naila, and R.-C. Amar, "Energy efficient firmware over the air update for tinyml models in lorawan agricultural networks," in *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*, 2022, pp. 21–27.

[10] X. Bu, C. Liu, S. Liu, Q. Wang, L. Lv, and W. Lu, "Low power consumption and intelligent design of power line iot edge nodes," in *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, vol. 5, 2021, pp. 1619–1624.

[11] Z. Wang, C. Hu, D. Zheng, and X. Chen, "Ultralow-power sensing framework for internet of things: A smart gas meter as a case," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7533–7544, 2022.

[12] Y. Meng, Z. Xudong, Z. Jianwen, X. Xinxin, W. Changling, and W. Fang, "A ultra-low power system design method of ai edge computation," in *2023 19th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2023, pp. 1–5.

[13] L. Schulthess, S. Marty, M. Dirodi, M. D. Rocha, L. Rüttimann, R. H. R. Hahnloser, and M. Magno, "Tinybird-ml: An ultra-low power smart sensor node for bird vocalization analysis and syllable classification," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.

[14] I. W. Mustika, A. Bejo, A. R. Fidiyanto, and D. W. Hapsari, "Development of campus bus tracker firmware using gnss module on the stm32 platform," in *2023 International Conference on Digital Business and Technology Management (ICONDBTM)*, 2023, pp. 1–6.

[15] M. Hayashikoshi, H. Noda, H. Kawai, K. Nii, and H. Kondo, "Low-power multi-sensor system with task scheduling and autonomous standby mode transition control for iot applications," in *2017 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2017, pp. 1–3.

[16] ENABLES Project, "Position paper," 2024, accessed: 2024-09-21. [Online]. Available: https://www.enables-project.eu/outputs/position-paper/

[17] M. Rashid, L. Ardito, and M. Torchiano, "Energy consumption analysis of algorithms implementations," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2015, pp. 1–4.

[18] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[19] Team SimPy, "SimPy: Discrete Event Simulation for Python," https://simpy.readthedocs.io/, 2023, accessed: 2024-07-10.