Sense Your Power: The ECO Approach to Energy Awareness for IoT Devices

MICHEL ROTTLEUTHNER and THOMAS C. SCHMIDT, Hamburg University for Applied Sciences, Germany
MATTHIAS WÄHLISCH, Freie Universität Berlin, Germany

Energy-constrained sensor nodes can adaptively optimize their energy consumption if a continuous measurement is provided. This is of particular importance in scenarios of high dynamics such as with energy harvesting. Still, self-measuring of power consumption at reasonable cost and complexity is unavailable as a generic system service.

In this article, we present ECO, a hardware-software co-design that adds autonomous energy management capabilities to a large class of low-end IoT devices. ECO consists of a highly portable hardware shield built from inexpensive commodity components and software integrated into the RIOT operating system. RIOT supports more than 200 popular microcontrollers. Leveraging this flexibility, we assembled a variety of sensor nodes to evaluate key performance properties for different device classes. An overview and comparison with related work shows how ECO fills the gap of in situ power attribution transparently for consumers and how it improves over existing solutions. We also report about two different real-world field trials, which validate our solution for long-term production use.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; • **Hardware** → **Energy metering**; • **General and reference** → *Measurement*;

Additional Key Words and Phrases: Energy harvesting, power measurement, energy management, IoT operating system

ACM Reference format:
Michel Rottleuthner, Thomas C. Schmidt, and Matthias Wählisch. 2021. Sense Your Power: The ECO Approach to Energy Awareness for IoT Devices. *ACM Trans. Embed. Comput. Syst.* 20, 3, Article 24 (March 2021), 25 pages.
https://doi.org/10.1145/3441643

1 INTRODUCTION

Sensors and actuators deployed in the emerging Internet of Things (IoT) are frequently unconnected to power and solely rely on local energy sources. Whenever energy is a limiting factor, detailed knowledge of its availability and demand is critical to sustain operation reliably. In dynamic scenarios online consumption measurement and autonomous energy management

Authors' addresses: M. Rottleuthner and T. C. Schmidt, Hamburg University for Applied Sciences, Berliner Tor 7, Hamburg, Germany, 20099; emails: {michel.rottleuthner, t.schmidt}@haw-hamburg.de; M. Wählisch, Freie Universität Berlin, Takustraße 9, Berlin, Germany, 14195; email: m.waehlisch@fu-berlin.de.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
1539-9087/2021/03-ART24 $15.00
https://doi.org/10.1145/3441643

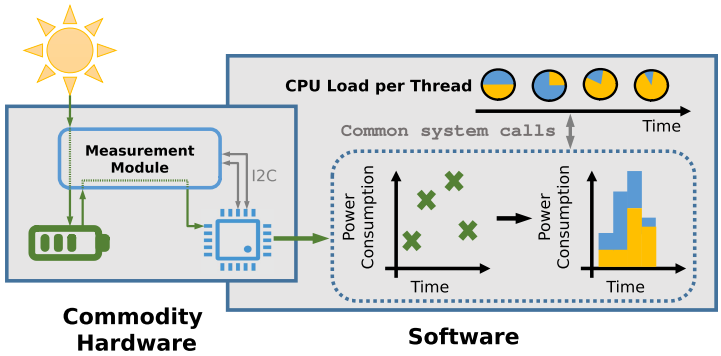

Fig. 1. ECO measures energy in situ by monitoring energy sources and consumers.

become important building blocks for that. Paired with energy-harvesting units, a successful resource control enables virtually unlimited deployment periods in the wild.

Tracking, controlling, and optimizing energy consumption depends on many aspects and quickly becomes complex. Many low-level details of the underlying hardware platform need consideration when building an energy-aware IoT device. This includes power consumption profiles of integrated sensors, actuators, communication peripherals, as well as power-saving primitives and configuration options of the employed microcontroller. Systems that are subject to varying operational or environmental conditions often cannot rely entirely on *a priori* derived predictions. Accurate information about actual deployment conditions are required instead. In addition, systems that rely on energy harvesting often require domain knowledge about the environmental conditions to proactively manage their consumption for sustainable operation. For those reasons, in situ measurement is a key strategy to obtain real-world information.

As hardware properties and conditions differ between use cases, developing energy-aware IoT devices often boils down to re-inventing (and thereby often re-implementing) the wheel. For avoiding this, a generic off-the-shelf solution is desired that is portable, reusable, and covers various settings of software and hardware components.

ECO is a hardware-software co-design that enables generic energy management down to commodity class-1 IoT devices [8]. The ECO approach is generally applicable and based on two key principles (see Figure 1). First, it connects a simple external measurement module via the standard I^2C interface, which is widely available on class-1 and class-2 microcontrollers. Second, it uses platform-independent software to expose energy metrics to the application while hiding low-level details from the developer. This software is integrated with the IoT operating system RIOT [5] and enables energy-aware applications to run on a large set of microcontroller boards without requiring any software adaptation when moving to another platform.

In previous work [42], we sketched the basic idea behind ECO. In this article, we introduce the complete design, detail its implementation, present an extensive evaluation, and report about two long-term field trials. In detail, we

(1) derive the key challenges of in situ energy management and give an overview on how IoT devices are powered and where generic, platform independent solutions for energy management are missing;
(2) present an energy measurement module that was designed as a reusable, external component and can be integrated with many common hardware platforms;

(3) explain the integration with the RIOT IoT operating system, by which ECO becomes a full energy management system that attributes power consumption to application-independent threads or application-specific traces;
(4) demonstrate the feasibility of ECO by thorough evaluations and performance measurements;
(5) validate our approach in two long-term field trials, one of which spans two months and the other runs continuously since more than one year.

The remaining article develops the problem space in Section 2, where the specific context and associated challenges are explained. A comprehensive discussion of related work is presented in Section 3. Section 4 works out our design principle including core hardware and software building blocks with their key implementation properties. A thorough evaluation of the ECO system follows in Section 5. We report about two extensive deployments in the wild in Section 6 and conclude with an outlook in Section 7.

2 ENERGY-AWARE SYSTEM DEVELOPMENT FOR THE IOT

We now discuss how typical IoT devices are powered, where energy management and in situ power measurement become important in this domain, and how energy-aware software development can benefit from a generic solution.

2.1 How the IoT Powers Its Things

IoT devices are powered in many ways. The most primitive example are permanently wired devices. Those have access to electricity all the time and only need very static power-saving mechanisms to not unnecessarily waste energy, but energy availability is not a critical factor for those devices to work.

Devices that are powered by disposable batteries throughout their entire lifetime share the concern of energy efficiency to archive the target runtime with the smallest possible battery size. Rechargeable batteries may loosen these constraints. Given the device can be charged up in time, very strict low-power operation is not always required. Also, depletion of rechargeable devices is often actively avoided by their users.

Energy-harvesting devices, however, rely on energy that can be collected from the environment. This does not require manual intervention and enables perpetual operation. Such devices usually harvest power from weak sources over a longer time for collecting enough energy to fulfill their task in a duty-cycled manner. Energy neutral operation (ENO) is achieved whenever a device is able to fulfill its given task while keeping consumption in balance with harvested energy [27]. The life cycle of an ENO system is thus mainly limited by the lifetime of its components—the most delicate part is often the batteries due to the memory effects of the chemical processes therein.

To overcome aging, chemical energy storage elements are often substituted with parts that store energy physically, such as capacitors or super capacitors. These are much more robust and provide longer service lifetime, but come with a much-reduced energy density. Battery-less, a.k.a. intermittent devices like RFID tags powered from RF energy [7], mark the lower end of embedded systems regarding energy availability. In this device class the energy availability is usually even smaller than for typical ENO devices and its intermittence often does not even allow to complete a task at once, but requires software check pointing mechanisms [9].

In this work, we focus on devices that fall into the ENO class, i.e., devices that are very constrained but still come with enough resources to include software abstractions and actively manage their energy. Similar to the work shown by Hester et al. [23], we want to foster easy-to-use energy-harvesting systems, but instead of building one very easy-to-use modular platform, we

explicitly focus on compatibility with various different devices and integrate it with a common software platform. This enables users of an already established operating system to benefit from our contribution.

2.2 Platform Independence to the Rescue

Software engineering best-practices have been pushing for building modular and reusable code that is platform-independent since a long time [26]. In the context of embedded IoT systems, this has not been well adopted [36]. Many manufacturers compete on the market by tailoring specialized solutions to their specific hardware. Portability and sustainability across platforms and different brands have not come to focus yet. More mature platforms such as PCs or smartphones have converged to these principles for a long time and introduced powerful abstractions using high-level languages and open ecosystems. For example, when developing an Android application, most of the time the developer neither knows, nor needs to care about, which device it will run on later.

Only recently, a similar trend emerged for the IoT with the growing popularity of embedded operating systems that aim to support reusable and modular software while managing compliance to the very tight resource constraints of IoT devices. The prospective use cases for an IoT OS are broad and range from toy gadgets to industrial-grade control systems with hard real-time requirements. The diverse set of peripheral interfaces for sensing, actuation, or other actions demand for sophisticated abstractions that expose low-level hardware features in a generic fashion.

RIOT [4] is a modular operating system for constrained IoT devices with a strong focus on open network standards, well-known programming interfaces, and vendor independence. It is built and maintained by an open source community spread all over the world. A major benefit of using RIOT comes with portability and comprehensive library support. By incorporating extensive hardware abstractions, it is available for a very broad range of devices with more than 200 different boards. The modular RIOT architecture also provides multiple network stacks [34], a large ensemble of communication protocols [20–22], and drivers for sensor and actuator peripherals. With its help, developers can implement interoperable, networked applications as reusable code for a huge set of different platforms. Such a unified software platform hides hardware complexity and makes writing reusable code viable—even across vendor boundaries. Developers benefit from generic software libraries, streamlined tooling, and unified development processes to focus on a fast implementation of application logic without being tied to specific hardware.

A versatile support for such large heterogeneity of hardware features and operational requirements, however, pose significant challenges to the design and implementation of generic IoT systems. Energy management can be seen as a poster child for such a system challenge. While smartphones and other mobiles have large batteries that are actively managed by users, most future IoT devices will autonomously run on low-power resources. In many deployment scenarios, including energy harvesting, the devices need a dynamic power management, which should preferably ship as a reusable generic component that just needs to be included and parameterized to fit the targeted application. This, however, needs to master a delicate interplay between the system software and the underlying hardware.

Building a dynamic energy management in form of a reusable system module demands for a platform independent way of tracing power consumption and charging metrics, while presenting this information to an abstract application-level interface. Attributing these values to software entities of the running application adds additional semantic contexts to the data and provides meaningful insights into which parts of the application are responsible for which portion of power consumption. Being aware of whether energy is available and how exactly it is used at the system level enables powerful feedback for a self-adaptive control of the energy management. Integrating

this functionality into a popular, widely deployed IoT operating system further allows to even use these features on already existing applications without any modification of existing source code.

3 THE RESEARCH LANDSCAPE: FROM SIMULATION TO IN SITU MEASUREMENTS OF POWER CONSUMPTION

Energy harvesting has been an active research area since more than two decades [48] and covers a wide range of topics [49]. Respectively, work related to this article overlaps with a multitude of different areas. Previous work focused on custom hardware layouts to improve cost, scale, efficiency, and many other aspects of harvesting, charging, and energy storage solutions. Software solutions cover implementations of energy management mechanisms and enable smarter use of energy by optimizing for specific quality of service metrics. For a brief, structured overview on related work regarding energy management, contributions can be coarsely categorized according to the abstract functionality they focus on.

Assessment provides information on how much power is charged or consumed, based for example on measurements or estimations.

Attribution correlates energy consumption with specific operations such as executing a function, running a communication protocol, or an application thread.

Allocation assigns available energy resources to activities such as data processing procedures or powering of peripherals.

Every category can be further split into sub-groups based on how the particular functionality is achieved. For example, assessing the consumption of an application can be done *online* (i.e., at runtime) or offline—with significant implications. Incorporating runtime information on the actual consumption as feedback for the energy management algorithm was already shown to improve application-level performance and robustness against uncertainties [19]. Similarly, assessment can be performed in situ (i.e., on a deployed node itself) or on an external system. Independent of *when* and *by whom* the assessment is executed, the underlying mechanisms can also be very different. Complex simulations, state models based on simplified static reference values, and physical measurements are notable examples.

With this context in mind, we will now dive into more details of the most relevant related work. This discussion of related work will guide from categorization to a qualitative overview that highlights the work most applicable in our context. As we focus on assessment and attribution aspects in this article, we will mainly discuss related work in these domains and keep the overview on allocation mechanisms short. At the end of this section, we summarize our discussion in a qualitative comparison.

3.1 Simulation and Estimation of Power Consumption

Software approaches range from simulation of large-scale sensor networks [45] to estimations based on offline reference measurements [13, 15]. The prediction accuracy of the power consumption and battery lifetime is significantly affected by its level of abstraction [33]. Neglecting low-level system events such as scheduling and timer-related interrupt handling can lead to substantial errors, because they often account for a relevant part of the power consumption.

Simulation of consumption can be performed from a generic high-level perspective [2] down to estimating CPU cycles [45] or even on a single instruction level [51]. The full control over relevant system parameters eases isolated analysis of individual aspects. Downsides relate to the inaccurate reproduction of reality due to environmental changes, varying hardware tolerances, and many other dynamics. Even if conditions are equal, different device instances of the exact

same model can exhibit significant variations in their consumption [35]. Also, simulations mainly focus on improving *a priori* adjustments, leaving runtime optimizations open to other solutions.

Estimating the energy consumption *online* allows incorporating more runtime specific criteria from actually tracking the system states [13]. As an example, energy usage caused by packet retransmissions can be accounted with higher accuracy if the exact number of transmissions is known and considered at runtime. While this varying size can be easily determined in software, other changes such as the efficiency of selected electric components are more challenging to estimate. Incorporating runtime information on the actual consumption as feedback for the energy management algorithm was already shown to improve application-level performance and robustness against uncertainties [19].

Dunkels et al. [13] developed Powertrace, a software solution to allow network-level profiling of applications for Contiki [14]. It feeds static values from offline measurements into a linear power model. This lightweight software-only solution has many benefits compared to hardware solutions. Unfortunately, it does not apply to dynamically powered systems such as energy-harvesting systems with varying supply voltage. It is also unsuitable whenever the individual consumption of the various components inter-depend non-linearly. Covering these effects with estimations and simulations significantly increases the complexity and may reduce general applicability.

3.2 Measuring Power Consumption

A typical way to keep track of the energy flow on an IoT node itself is to utilize coulomb counters or indirectly assessing the consumption by measuring voltage changes at the energy storage element [41]. While the temporal resolution is sufficiently high to assess the state of charge, it is insufficient to attribute energy usage to specific tasks or peripheral hardware. To overcome these problems, custom hardware for faster sampling is required.

A core problem of accurately measuring the power consumption of IoT nodes arises from the heavily varying power demands of microcontroller units (MCU) while switching between different power modes. These changes can span five orders of magnitude [24]. Additionally, the measurement itself should have little to no side effects on the observed system. Combining both demands introduces further complexity.

Many measurement systems were designed as external observers to record the detailed behavior of the sensor node. The underlying architectures range from Linux-capable systems based on single board computers [28, 37, 38, 46, 47], to small add-on boards that are equipped with an additional MCU [43, 53, 54], complex programmable logic devices (CPLD), or field programmable gate arrays (FPGA) [3, 43, 50]. While using powerful separate hardware has obvious benefits in terms of performance and flexibility, it is not feasible in production to deploy full Linux machines for power-metering purposes. Even using additional MCUs or FPGAs potentially doubles the system cost and significantly adds to the power requirements and system complexity.

With Rocketlogger, Sigrist et al. [46] introduced a portable device intended to provide a balance between top-notch lab equipment and mobile measurement. The platform provides four voltage and two current channels besides the option to interface digital sensors for additional environment metering. This custom mobile measurement platform allows the observation of multi-source harvesters but is not designed for long-term off-grid deployment because of high energy requirements. Using such sophisticated equipment for self-measurement on a per-IoT-node scale in production, as we focus on, is not suitable due to cost, resources, and complexity.

FlockLab by Lim et al. [37] targets distributed tracing and profiling. Only a single shunt is used, and linear regression is utilized for calibration. They also use hardware with relatively high computational power, disqualifying it for in situ usage. Similarly, Geissdoerfer et al. developed Shepherd [18] with special focus on recording and replay of harvesting conditions for very low-power

intermittent devices. Additional effort is put into precise time synchronization to cover distributed applications. Even though it is built as generally portable, its setup is also based on a powerful single board computer and thereby not targeted for per-node deployment at production scale.

Kazdaridis et al. [28] use classic shunt metering with two resistors in series. For dynamic switching, a load switch bypasses the measured current around the high resistance when the burden voltage becomes too high. The switch is controlled by an analog high-speed comparator. Continuous measurement without MCU interaction is not possible with the module, because no internal sampling buffer or averaging is available.

The power measurement based on voltage to frequency conversion introduced by Jiang et al. [24] is suited to be directly deployed with a sensor node. The system called SPOT leverages a voltage to frequency–based digitization process to overcome many of the previously described measurement challenges. While this solution keeps processing overhead low for infrequent reading, the overhead gets high for fast sampling. An implementation by Dutta et al. [16] simplifies SPOT by essentially eliminating the need for any hardware if the platform is powered by a switching regulator. Shortcomings of the solution are related to inherently high manufacturing tolerances for inductors (around ±10%), the frequency dependent power overhead, and that the voltage is assumed constant instead of being actually measured.

3.3 Attributing Energy to Software

The software running on a wireless sensor node defines how energy is spent in operations such as issuing sensing cycles or requesting the hardware to transmit data. So in addition to assessment mechanisms, energy awareness requires software components that semantically link actions performed by an application to the consumption induced by them. As resources such as CPU, RAM, or hardware peripherals are dynamically shared between a multitude of software components, attributing the exact amount of used energy to the correct software instance is challenging. Therefore, the main problem in this domain is correlating power consumption with the executed software.

A simple base metric for consumption correlation is *utilization*, derived from the time a thread occupies the CPU. The accuracy of this method can be significantly improved by monitoring CPU-internal performance counters [6]. With pTop [12], an implementation for desktop-scale devices was presented that uses this information to attribute power consumption to running processes.

Further accuracy improvements to online estimations can be achieved by tracking power states of individual components. For TinyOS, Kellner [30] uses a common model to estimate the overall power consumption, which is then attributed to different individual TinyDB queries with the help of resource containers. Fonseca et al. [17] augment the tracking of component power states with real power measurement and activity tracking to allow fine-grained offline analysis of energy usage. As TinyOS does not provide threads by default, both solutions introduce abstract entities (i.e., activities and resource containers) to which the energy use is attributed. This allows grouping of semantically related resources and thereby improves its high-level visibility to the developer. Still, it requires additional instrumentation of the target application, which we aim to avoid with our work.

3.4 Allocating Available Energy Resources

Once a system has gained understanding of energy availability and consumption, it needs to actively manage how the energy is used. Any approach to managing energy on constrained devices relies on accurate knowledge of the actual conditions. Overestimating the available energy may quickly put a node out of service, underrating energy may hinder its operational utility. Jiang et al. [25] formulate an abstract architecture for energy management that supports

Table 1. Qualitative Comparison of Related Work

Variant	In situ†	Physical Measurem	platform Agnostic	Assessment Mechanism	Attributed Entities	Feedback for Allocation
PowerTOSSIM [45]	✗	✗	✗	Simulation (State Tracking)	Selected HW Components	✗
Kazdaridis et al. [28]	✗	✓	✓	Sampling (Current)	*None*	✗
Rocket Logger [46]	✗	✓	✓	Sampling (Current, Voltage)	*None*	✗
FlockLab [37]	✗	✓	✓	Sampling (Current)	Selected Events *(ex post)*	✗
AVEKSHA [50]	✗	✓	✗	Sampling (Current)	Selected Events	✗
Shepherd [18]	✗	✓	✓	Sampling (Current, Voltage)	*None*	✗
Powertrace [13]	✓	✗	✓	Estimation (State Tracking)	Selected Activities	✓
Kellner [30]	✓	✗	✓	Estimation (State Tracking)	Selected Activities	✓
SPOT [24]	(✓)*	✓	✓	Sampling (Current)	*None*	✗
Nemo [54]	(✓)*	✓	✗	Sampling (Current)	*None*	✗
iCount [16]	✓	✓	✗	Sampling (Current)	*None*	✗
Quanto [17]	✓	✓	✗	Sampling (Current)	Sel. HW, Activities *(ex post)*	✗
ECO	✓	✓	✓	Sampling (Current, Voltage)	Selected Sections, Threads	✓

† Referring to usability in *off-grid* in situ deployments (i.e., some of the solutions marked as unsuitable may still be usable for *wired* in situ).
*depending on sufficient power supply (refer to Table 3 for details).

graceful degradation of performance instead of failures in case of decreasing energy availability. In addition to abstract concepts, many well-established algorithms exist that govern a spending of available energy at the system. Most approaches adapt the duty cycle of periodically executed tasks in a predictive or reactive fashion.

Exponentially weighted moving average (EWMA) filters can be used as simple and efficient adaption mechanism including basic forecasts for uncontrolled but predictable energy sources [27]. Vigorito et al. proposed ENO-Max [52], which also considers task performance and adds tunable stability for minimized duty-cycle variance. By not relying on a model for the energy source, it is generally applicable even without *a priori* information about the behavior of the source. To significantly improve the simplified EWMA approach under variable weather conditions, Piorno et al. [40] conditioned the calculation with current weather data. With LT-ENO [10], Buchli et al. focus on reliable longterm energy neutral operation that targets several years of deployment. This is achieved by preserving seasonal excess of available energy for longer periods when harvesting output is insufficient.

Recent work by Geissdoerfer et al. [19] incorporates user-defined utility into the energy management. Their solution effectively combines the benefits of predictive and reactive solutions to further improve the utilization of available energy resources, especially in cases where utility of the device operation and energy availability are not aligned.

3.5 Summary

Table 1 compiles a qualitative overview of the work under discussion. It shows which solutions are suitable for online in situ application and which make use of physical measurement or software-based approaches. Albeit many solutions can be used in situ as an external observer, our classification explicitly highlights whether an approach can be performed onboard by the IoT node itself. Contributions are marked as platform agnostic if their design principle, architecture, and implementation are considered to be portable to other hardware and software platforms. The assessment

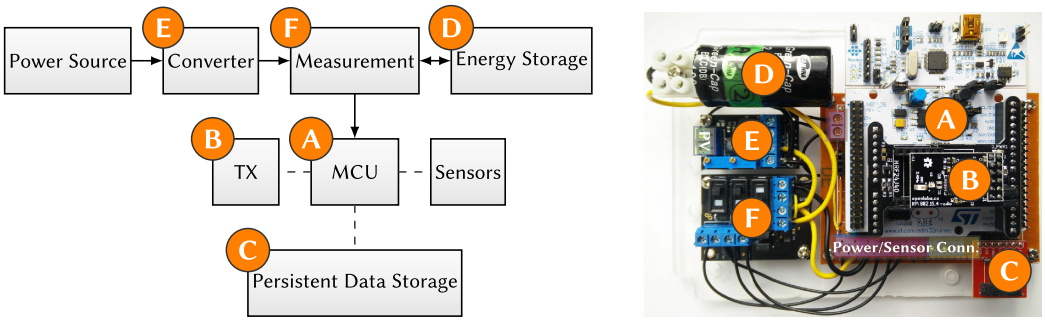

Fig. 2. Mapping of ECO building blocks to physical components of energy-harvesting wireless sensor node.

is grouped by the mechanisms in use and by the entities that are considered. All contributions are further classified according to how assessment, attribution, and allocation are performed. When methods for attribution are within scope, the consumption-correlated entities are listed. Cases where the attribution is considered but actually performed as *ex post* analysis are marked accordingly. Feedback provided by assessment and attribution mechanisms that is directly usable by an OS or energy management component to regulate allocation parameters are marked separately.

In the following section, we introduce a flexible self-measurement setup that uses readily available parts and is easy to integrate with common IoT devices. It spans a wide range of configurable measurements, supports different sampling rates, and is compatible with common IoT platforms. Using various configurations the accuracy is verified with precise reference measurements. Additionally, the overhead (i.e., invasiveness) induced by the measurement action itself and communication with the module is analyzed to show what cost is tied to more fine-grained, online energy profiling. The results can be used to choose an appropriate measurement configuration for specific use cases by weighing between tolerable overhead and additional granularity.

4 INTEGRATED ENERGY MANAGEMENT: DESIGN AND IMPLEMENTATION

We are now ready to describe the integrated hardware-software co-design of ECO. First, we introduce the flexible modular hardware platform. Figure 2 depicts our abstract model, consisting of a wireless sensor node and a power subsystem and its mapping to the physical hardware components. The power subsystem includes a power source, modules for charging and measurement, and an energy storage. The node itself is built as a fully modular integration of an MCU, a radio transceiver, sensors, and persistent data storage.

In the second part, we discuss the software integration of power-measurement primitives for the RIOT operating system. Two orthogonal concepts are implemented for the online measurement that provide different tradeoffs between integration complexity and accuracy. To add energy awareness to new applications, we provide an API to explicitly gauge the energy consumed on an individual task base, i.e., for predefined code-sections. Additionally, we extend the scheduler of the operating system to provide energy statistics on a per-thread level. With this implicit energy attribution to software entities, the system can even inform about energy consumption without requiring any changes to existing application code.

4.1 Modular Hardware Design

The hardware is composed of an off-the-shelf evaluation board, a super capacitor as energy buffer, and a measurement module that quantifies the charging and discharging rate. The entire setup is built as an orchestration of independent modules. All of them can be exchanged, which leaves

the design flexible regarding the selection of specific components and enables its use in various scenarios.

To represent typical IoT use cases, an IO-interface serves connectivity to external peripherals such as sensors for data acquisition. A network uplink is provided by either an IEEE 802.15.4 module based on the Atmel AT86RF233 chip or a Semtech SX1272 LoRa module. If needed, the system can also be used fully independent of external infrastructure, making the setup applicable for standalone settings (e.g., wildlife monitoring). In that case, a micro-SD card is utilized as cost effective persistent storage for long-term data logging.

A bare development board with the low-power STM32L476 MCU is used as the plain basis for a fully modular setup that can represent a wide range of typical IoT sensing devices. Using a bare development platform like this has the benefit to keep almost all individual components replaceable with other variants without much effort. The selected MCU offers fine-grained configuration for almost any low-power feature. Peripherals, ram-retention, power states, and clock frequencies can be configured with a high degree of freedom to adjust between performance and efficiency. Despite appreciating its inherent flexibility, it is worth noting that the setup is not tied to this specific development board, as we will show later.

Using different radio modules, voltage regulators, energy storage elements, or other sensors is simple, while keeping low-level control over all those components. The effects and impact of changing a single specific component can therefore be researched more easily.

The power-measurement module consists of a configurable, yet simple breakout board for the Texas Instruments INA226 shunt monitor. With a fixed maximum burden voltage of 81.92 mV, the measurement range can be varied by manually switching three different shunt resistors for simple adaption to other target platforms. It is interfaced via I^2C and allows changing the slave address to use multiple modules in a single setup. For increasing the dynamic range, an additional high resistance shunt together with a bypass switch can be used, as shown in [28].

The assembly of all interconnected parts for the deployment-ready energy-harvesting system is depicted in Figure 2. The bare development board (A) with the external radio module (B) on top is plugged into an interface board that in turn provides connection to persistent storage via a micro-SD slot (C), the power subsystem, and various external sensors. A 100 F (2.7 V) super capacitor (D) in combination with a flexible photovoltaic charging circuit (E) and the previously introduced measurement module (F) form the external modular power subsystem. All components such as the radio, micro-SD card, and external sensors can be powered down completely by individually switchable transistors. These are integrated on the interconnect-board below the MCU and are controlled by plain GPIO control via the platform independent API.

4.2 Integrated System Software

Integration of ECO into RIOT is split into different modules to form a generic attribution layer. A peripheral driver for access and control of the INA226 over I^2C is implemented on top of the respective I^2C and GPIO interfaces of the RIOT hardware abstraction. In addition to raw register access the driver provides functionality for conversion into physical units and helpers for measurement calibration. The calibration is performed by connecting a trusted reference device and the module to the same load. The calibration is then enabled by feeding the measured values back into compile time configurations.

We extend the simple command line interface of RIOT with an additional command named `es`. It builds on top of the existing `ps` command and adds information on power draw and energy consumed per thread, similar to the default statistics such as stack usage and context switching count. The logic for that is implemented with a separate background thread that controls the measurement, reads samples from the external module, and performs the required calculations.

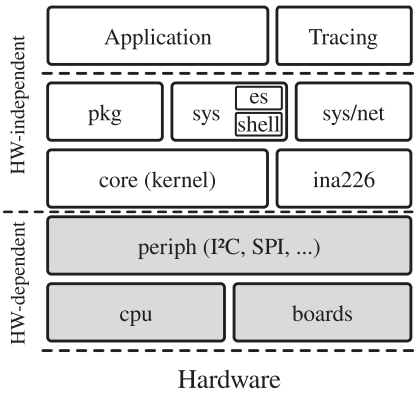

Fig. 3. RIOT modular architecture with integrated `es` command and application-level tracing.

By that, the thread priority control of the OS can be used to adjust between precise timeliness of the measurements and less invasiveness.

The automatic attribution schema splits the samples according to the time each thread was active and accounts resources to the different threads accordingly. This approach has the benefit that not a single line of code needs to be changed to give an overview of energy expenditure by different parts of the application. It is noteworthy that the accuracy of this method may depend on the application architecture. Energy consumed by threads that trigger a (peripheral) activity of high energy demand and then go back to sleep immediately cannot be evaluated very accurately by this approach.

To overcome this limitation, a tracing mechanism is implemented to let an application explicitly record energy traces of specific tasks defined by the developer. Tracing can be controlled by the interface calls `trace_start()` and `trace_stop()`. It may record a full series of power samples or return a single aggregated value. Using this tracing, an application can re-evaluate its energy use per task in varying conditions such as ambient temperature, state of charge, or degraded component health. When the measurement function is not in use, the module is put to a low-power mode and consumes less than 2 μA. For cases in which even this low consumption is critical, powering down the module can be combined with transistor-based power gating, as previously described.

Figure 3 visualizes a simplified RIOT software stack that integrates the aforementioned components. The dashed line in the middle separates the stack into hardware dependent parts on the bottom and hardware independent parts above. The lower dashed line illustrates the border between OS modules and the specific hardware it is running on, and the upper dashed line splits user- or application code from the OS. In particular, the driver of the INA226 is situated on top of the hardware abstraction and is thus available on every platform that supports I^2C. The `es` command is implemented as a `sys` component using the shell module. Next to the application the explicit tracing mechanism resides as an independent module.

5 OVERALL SYSTEM PERFORMANCE: EVALUATION OF IN SITU MEASUREMENT

We verify the ECO design by evaluating its key performance properties and question whether the generic self-measurement module is able to support the target use cases. More specifically, we want to answer the two questions:

(1) How accurate does the self-measurement perform?
(2) Which penalties does its generic implementation impose?

To illustrate how ECO performs, we focus on three different aspects: (*i*) measurement quality, (*ii*) cross-platform compatibility, and (*iii*) overhead. We investigate the overhead in two dimensions: the power consumption added by the self-measurement and the CPU utilization imposed by acquiring and processing the data. We validate the CPU utilization on four largely heterogeneous MCU platforms with computational power ranging from 100M instructions per second down to 16M instructions per second. Since the focus lies on the energy-constrained domain, we inspect the consumption of measurement operations and the communication with the external module individually.

A highly accurate Keithley DMM7510 bench multimeter is employed for reference measurements. With 7½-digit measurement capability, a sampling rate of up to 1 MHz, and integrated self calibration (<200 PPM), it provides a highly accurate baseline to compare against Reference [29]. This meter is controlled by automated scripts for multiple reproducible runs. Accordingly, the sensor node is controlled by a custom RIOT-shell interface that can start test runs and handle configuration of all runtime parameters. The application running on the MCU consists of a thread that queries the power consumption readings produced by ECO on top of the RIOT operating system core including modules that would typically also be part of a real-world application. This includes abstractions for inter-process communication (e.g., message passing, mutexes), timers, peripheral drivers for GPIO, and interrupt handling as well as the ECO-module. Additionally, the shell module with custom command handlers is part of the firmware to communicate with the device from a connected computer. In repeated experiments, errors of the measurement and digitization process are quantified by comparing results for the device under test with the reference values.

5.1 Measurement Quality

5.1.1 Accuracy. Testing the accuracy of the measurement is done using a fixed resistor as static load. The applied voltage is varied for different current values. Measurements of the device under test and the reference are run simultaneously and repeated 1,000 times. Figure 4 shows the distribution of the relative errors as a function of current measurements ranging from 200 μA to 2 mA. The successive increase towards the lower end of the scale is attributed to bigger relative impact of noise and the digitization resolution. For loads of 200 μA the median error stays very close to 1%, which is well suited for self measurement. Currents higher than 1.8 mA are much more stable around a median of 0.5‰ Extending this measurement up to the full range of 40 mA (for the selected 2 Ω shunt) confirmed that the pattern of negligible deviation remains valid for higher currents, as can be seen from the absolute measurement deviation in Figure 5.

5.1.2 Resolution. The voltage resolution has a fixed value of 1.25 mV by design. Current measurement resolution depends on the active shunt resistor value (R_{shunt}) and the fixed 2.5 μV LSB step size of the shunt voltage as indicated in Equation (1).

$$I_{LSB} = \frac{2.5\ \mu\text{V}}{R_{shunt}} \tag{1}$$

We select a 2 Ω shunt that allows currents of over 40 mA, thus covering most typical IoT devices including connected sensors, while still maintaining a reasonable base voltage around 80 mV. This shunt resistor value grants a resolution of 1.25 μA. For tracing of the active node consumption, this is considered sufficiently accurate.

5.1.3 Latency. The dominating share in read latency is added by I^2C communication, which depends on the wiring and clock speed configuration. Selecting an appropriate priority for the power-measurement thread is also important. Choosing the priority too low may starve the thread

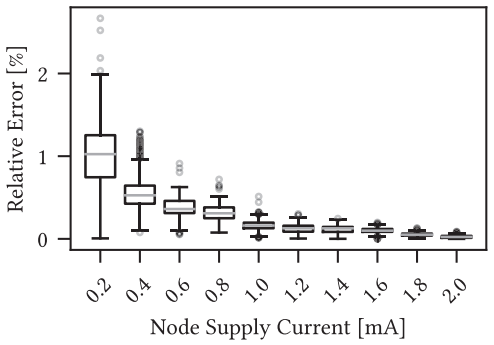

Fig. 4. Relative measurement errors at constant load with 332 μs bus/shunt conversion time and 16 avg. steps (IQR: 25th–75th percentile, whiskers: Q1-1.5*IQR and Q3+1.5*IQR).

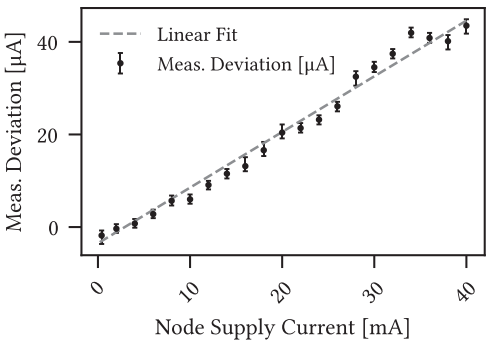

Fig. 5. Absolute measurement deviation in 40 mA range at constant load (2.5th to 97.5th percentile).

on high utilization and significantly increases latency and jitter. With a high priority it is important to select the sampling rate of the module low enough to avoid interference with the application. For deriving reasonable values, an application should be compiled for the target platform with the "ps" command functionality included. The overall computation overhead induced by the self-measurement can then be investigated and tweaked for the specific use-case and the available hardware resources.

5.1.4 Thread Attribution. The accuracy of attributing power consumption to a thread depends on several parameters, most notably the sampling rate of the measurement. Each power sample represents an average for the sampling duration due to the internal oversampling (@ 500 kHz) of the employed ΔΣ-ADC. This leads to errors whenever a context switch occurs in between and the subsequent thread has a different power consumption. Relative errors are thus a function of the ratio between consumption of a correctly and wrongly attributed thread running on the system. Relative errors also decrease with the active duration of the thread.

Figure 6 compares the relative attribution errors for different scenarios as a function of the sampling rate. P_{TA} refers to the power consumption of a thread that shall be attributed. Respectively, P_{TB} denotes the power consumption of the subsequent thread, which may affect the last sample. The ratio P_{TA}/P_{TB} therefore determines the relative difference of consumption between those threads. Accordingly, a hypothetical thread TB that has the same consumption as TA will cause no error and lead to a straight line of zero percent in the graph. Note that this ignores the negligible quantization error of calculating the timeshares. The accuracy clearly benefits from a higher sample rate, but some use cases may want to trade sampling overhead for attribution accuracy. Power measurements could be further improved by aligning the sampling with thread schedules or applying correcting factors to compensate for the overlap period.

5.2 Cross-platform Compatibility

A major goal of ECO is to enable energy measurements on the majority of IoT platforms, which requires a seamless way to integrate the module within existing platforms. We selected four largely heterogeneous boards to validate the cross-platform applicability of ECO. For a representative set of samples from different manufacturers and architectures, we explicitly chose two devices from a higher performance class (32-Bit Cortex M4 based nucleo-l476rg and slstk3402a), one midrange

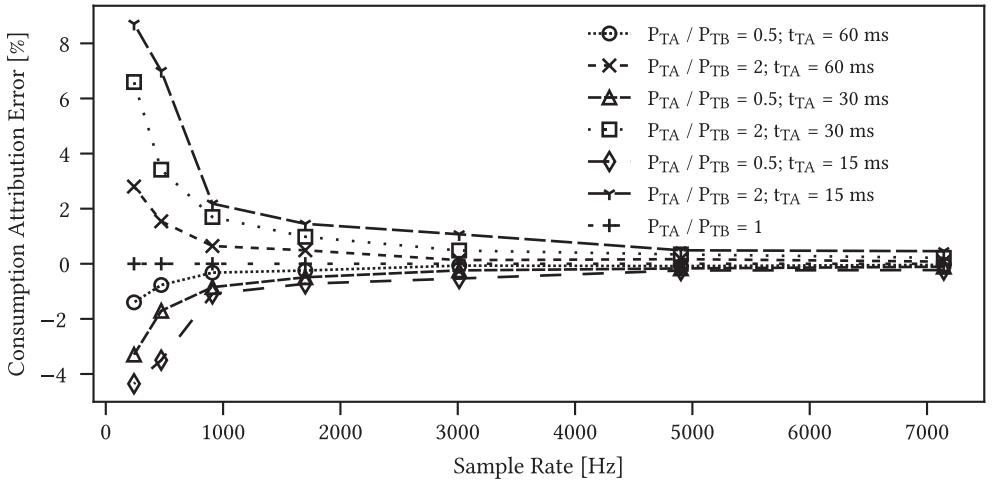

Fig. 6. Relative error of attributing power consumption to a thread as a function of sampling rate: P_{Tx} denotes the power consumption of thread x and t_{Tx} the runtime of thread x. Inaccuracy increases for large power changes of subsequent threads.

device (samr21-xpro running 32-Bit Cortex M0+), and one low-end device (8-Bit AVR8 arduino-mega2560).[1]

Notably, the cross-platform compatibility only depends on few key properties. The measurement module is a separate, external part. The accuracy of its physical measurement process hence only needs a single validation and evaluation. Calibration parameters are also tied to the module and not related to the specific MCU in use. Custom MCU-specific calibration procedures and accuracy problems of internal ADCs can also be circumvented by that. Thus, using ECO on another platform merely requires the availability of an I^2C interface and the ability to route the power supply lines through the measurement module before it connects to the MCU.

With the chosen I^2C connectivity, we already cover 100 of the boards currently supported by RIOT. Extending this list just requires provisioning of an I^2C peripheral driver. Apart from the low-level driver, enabling ECO for a new platform only requires to physically connect six pins from the microcontroller to the measurement module, which in turn is connected to the energy storage. The software support is enabled by declaring a dependency for the ECO-module to the project Makefile, which is the common way to enable features in RIOT. Custom pin mappings for the hardware, i.e., which I^2C-bus and pins of the MCU shall be used to communicate with the measurement module, can also be added directly to the Makefile.

While the general approach is not strictly tied to RIOT but can be applied to any modern IoT operating system, some specific capabilities of RIOT simplified the implementation. The tick-less scheduling keeps the system load low in the absence of events. With multi-threading, the consumed energy can be implicitly attributed to separate parts of the application. Also, prioritized scheduling provides direct control over the measurement invasiveness.

The general applicability of the proposed solution comes at the cost of a higher computational overhead compared to a highly specialized alternative. The following section, however, reveals that the approach is still usable on low-end 8-bit platforms. Considering that in the field of wireless sensor networks these architectures are even expected to be replaced with more capable 32-Bit

[1]The device names refer to the unique board names within RIOT.

Table 2. Maximum Values for Bus Capacitance (C_b), Signal Rise Time (t_r), and Pull-up Resistors (R_p) Based on I^2C Specification

I^2C mode	C_b	t_r	R_p
Fast	400 pF	300 ns	~885 Ω
Fast+	550 pF	120 ns	~258 Ω
High	400 pF	80 ns	~236 Ω

variants [31], we argue that the external module together with its tight operating system integration can be considered generically applicable for class-1 and class-2 devices.

5.3 Overhead of Self-measurement

We investigate the overhead introduced by bus communication in terms of time and power usage, the power usage of the measurement itself, and the computational overhead to process the samples.

5.3.1 Power. The power overhead is caused by three consumers: The measurement module itself consumes power during sampling, the serial bus spends energy on communication, and the shunt resistor introduces power loss. Because ECO uses an I^2C bus, the related communication overhead directly depends on the configured bus clock and the wiring. To ensure fast signal switching times and thus maximize bus speed, a low value pull-up resistor is needed. Decreasing the resistance, however, increases power consumption while the line is driven low because of the current flowing through the pull-up to ground. Following this observation, it is necessary to find an appropriate balance between high speed and low power consumption, i.e., a pull-up resistor introducing low overhead.

The main factors for determining the necessary pull-up resistance are the bus capacitance (C_b) and the signal rise time (t_r), each of which allows for a different level of adjustment. The value of C_b essentially represents physical properties of the bus connection, in contrast to t_r, which follows from the targeted clock and data signal speed. According to the I^2C bus specification [39], Equation (2) can be used to calculate the *maximum* pull-up resistance that is sufficient to comply with a specific bus capacitance. Table 2 lists the required resistance values for the given maximum ratings of C_b and t_r at different bus speeds as specified in the I^2C standard [39, Table 10, page 48]. It is worth noting that these characteristics are not tailored to ultra-constrained energy deployments, but reflect multi-purpose (worst-case) setups.

$$R_p(max) = \frac{t_r}{0.8473 \cdot C_b} \tag{2}$$

Instead of just selecting R_p based on these worst-case values, we are interested in the most energy-efficient setting for our specific implementation. In our setup, we measured the bus capacitance to be 158 pF, for which Equation (2) gives a calculated resistance value of around 600 Ω at *High* mode. This is already much more energy-efficient than the specified worst case of 236 Ω. Instead of simply adopting this value, we further examine the actual energy consumption via measurements that vary resistance values around the calculated optimum of 600 Ω.

Figure 7 depicts the overhead caused by reading values from the external measurement module in the time and energy domain for multiple bus configurations. Both, energy and time refer to a single register read. The energy values subsume the supply for the module itself as well as losses through pull-ups. Plotted time values include all delays introduced by layers up to the reading application. The varying parameters are the speed mode of the MCU I^2C peripheral and the pull-up configurations of the wiring. It can be deduced from the data that selecting the pull-up values

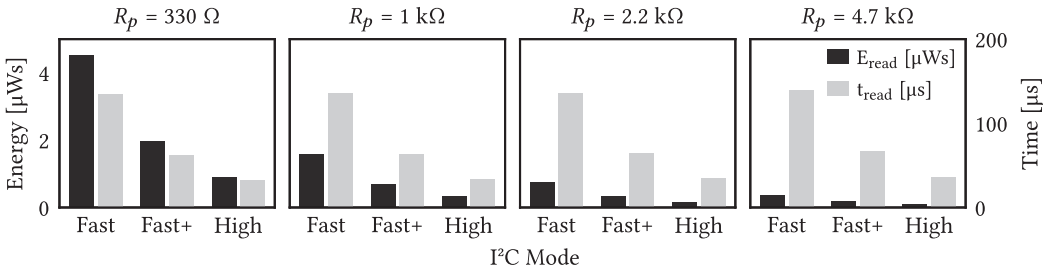

Fig. 7. Energy and time overhead for a single register read with different pull-up resistors (R_p) and I²C modes, given a targeted bus capacitance of 158 pF.

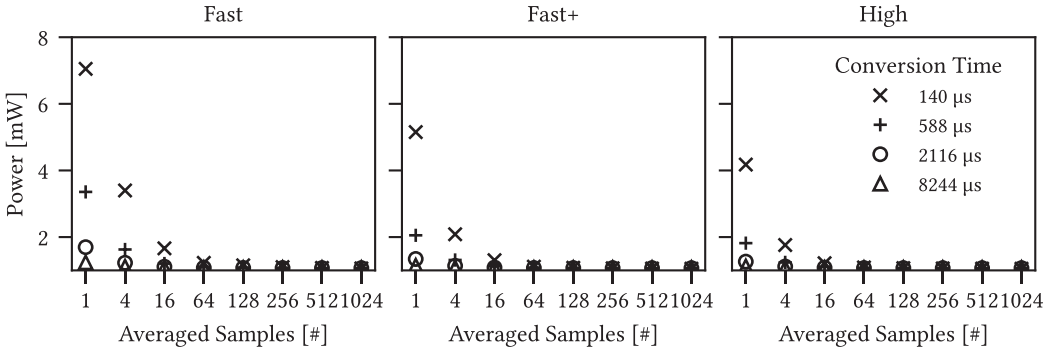

Fig. 8. Power consumption of the measurement module at different sampling configurations and I²C modes.

just according to the specification consumes significantly more energy, while the time spent for communication does not decrease analogously. The results can be used to select the most efficient energy configuration from an overall system view.

Our measurements show that the energy cost per read can vary from 4.5 μWs (330 Ω pull-up, fast mode) down to 100 nWs (4.7 kΩ pull-up, high mode). Because the read duration does not improve with the same ratio, the energetic optimum is between those two points and also depends on how much energy is spent by the MCU during this transaction. With Equation (3), we identify the high-speed configuration with a 2.2 kΩ pull-up as most energy-efficient for an average MCU consumption of 12 mA. This stays in contrast to the I²C specification recommendations and effectively leverages available safety margins by imposing stricter wiring requirements.

$$E_{read} = (P_{MCU} + P_{read}) \cdot t_{read} \tag{3}$$

Using the 2.2 kΩ configuration, we check whether different conversion times or averaging steps show a significant impact on the power draw. Respectively, Figure 8 shows the average power consumption during the sampling process.

Short conversion times together with low averaging (i.e., a high sampling frequency) shows significant impact compared to the overall measurement consumption. With 16 averaging steps, the overhead is only visible for the fastest sampling configuration, which becomes completely negligible when 64 averaging steps or more are used. Comparing the values with no averaging to the equivalent value with four averaging steps shows no noticeable dependence on how a targeted sampling interval is achieved.

Finally, we assess the power overhead based on the power loss introduced by the shunt resistor. This overhead exposes much less dynamics, because it just depends linearly on the current drawn

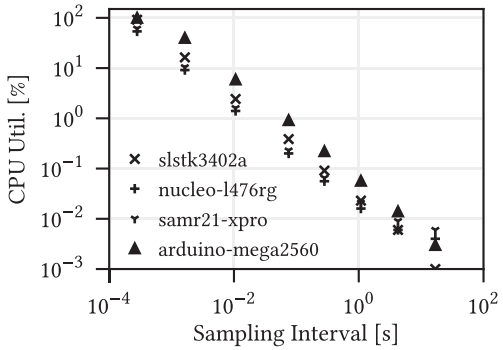

Fig. 9. CPU utilization for different sampling intervals.

from the sensor node and its supply voltage. A lower supply voltage implies that the fixed maximum drop across the shunt has higher relative impact. So for systems using voltages as low as 1 V, we recommend using a lower resistance shunt to trade efficiency for resolution.

Concluding the evaluation of the power overhead, we identify the static consumption of the sampling process itself as the dominant factor for most cases. The module communication becomes only relevant for very fast sampling and the shunt losses are only of concern when very low voltages and high resolution are required at the same time. The measurement consumption can be compensated by subtracting a fixed offset in the case of medium to high sampling rates. Maximum sampling rates require a linear correction that takes the communication overhead into account.

5.3.2 Computation. We quantify the CPU allocation while collecting the samples from the measurement module to select an appropriate sampling rate for the target domain. CPU utilization is measured by the `ps` command of the RIOT shell. The module is set to continuous sampling configuration with interrupt assertion enabled. A dedicated thread for the measurement is woken up by each interrupt and reads the values for voltage and current. This operating mode is left running for 10 minutes before the `ps` command is executed to list the percentage of active time for all threads. The values include every step from reading the device registers to physical unit conversion. The whole procedure is run for eight different sample interval configurations.

Figure 9 shows the results of this evaluation on four selected platforms ranging from a 32-Bit Cortex-M4 board running at 80 MHz (nucleo-l476rg) down to an 8-Bit AVR running at 16 MHz (arduino-mega2560). While the more powerful nucleo-l476rg board stays at 54 % utilization for a sampling time of 280 μs, the arduino-mega2560 is working on its limits and is not able to allocate any CPU resources to other application threads. All values drop almost linearly over the full range down to 0.16‰ and 0.57‰, respectively, at around 1 s. It is noteworthy that exhausted computational power at the highest sampling rate does not prohibit using ECO on that platform. Limited computational capabilities are also expected to decrease the required sampling rate due to the lower rate of events that need to be measured.

Comparing these values with the raw computational performance of the MCUs shows that the utilization on arduino-mega2560 only increases by a factor ≈ 4 compared to the nucleo-l476rg, whereas the performance differs by a factor of 6 (16M instructions per second vs. 100M instructions per second). The CPU utilization is thus dominated by the I^2C transfer time and not limited by the raw instruction performance. As the slowest sampling rate only generates 35 context switches to the measurement thread, its CPU values are subject to inaccuracies of the `ps` command.

Using the integrated power calculation of the module instead of reading current and voltage separately can about halve the time for I^2C reading. More room for improvement exists in the I^2C

Table 3. Performance Comparison of Selected Measurement Solutions

	Voltage	Relative Error	Power Overhead	CPU Overhead	Read Duration
SPOT [24]	✗	3%	~5 mW	*unpublished**	*unpublished**
iCount [16]	✗	±15% (5 μA - 50 mA)	0.3 μW - 30 μW	*unpublished*†	15 μs
Nemo [54]	✗	1.34% avg. 8% max.	0.4 μW - 12 mW	0.6% (10 s@0.5 Hz, 10 ms@8 kHz)	1 ms‡
ECO	✓	<1 % (>0.2 mA)	6 μW - 1.1 mW	0.16‰@1 Hz 1.5%@100 Hz	33 μs

*A longer reading time is indicated because multiple steps are required per sample and the I^2C is running with a slower clock (100 kHz).
†By using an internal hardware counter the overhead is assumed to be very low, but hardware-dependent.
‡Best case for stated bandwidth, ignoring protocol overhead, assuming 16 bit samples.

peripheral drivers of RIOT, which often do not leverage all the hardware features such as interrupt control or direct memory access. It is also worth to note that up to the sampling rate of 100 Hz, no dynamic compensation is required for the additional power consumption related to the I²C bus. At this rate even the very constrained 8-bit MCU running at 16 MHz stays below 6% utilization. This shows that ECO can be used on a huge set of platforms—even including very constrained variants—and it integrates seamlessly.

5.3.3 Memory. Memory overhead is considered from three perspectives. Static memory used by the module driver, the measurement thread, and a buffer for storing a history, if desired. The driver itself uses up to 560 B, depending on the actually used functionality. The application uses additional 36 B for the ISR callback, 32 B for a message queue, 268 B for measurement thread code, and additional 256 B of thread stack size. So the overall memory overhead introduced by ECO is close to 1 kB—roughly split in half between RAM and ROM. This will slightly differ, depending on the target architecture and the features enabled. Depending on whether separate voltage and current samples need to be stored or a single power value suffices, a trace history needs either additional two or four bytes per sample.

5.4 Discussion: Rating ECO

ECO was designed as a highly portable approach that can be deployed with over 100 different boards using a hardware shield carrying only commodity components and software tightly integrated with the RIOT OS. In contrast to many existing solutions, ECO neither exploits special aspects of individual boards, nor takes advantage of highly specialized or expensive components. It outperforms related work, though.

We compare the key performance indicators of ECO with those solutions that are roughly comparable and have published corresponding measurements in Table 3. The indicators are (*i*) availability of voltage readings, (*ii*) relative errors, (*iii*) overhead in power and CPU utilization, and (*iv*) reading performance. Overall, we find ECO in the vicinity of the best performers for each indicator. In particular, our system appears to be the most balanced with respect to the considered metrics, and it is the only solution that actually measures voltage.

SPOT [24], iCount [16], and Nemo [54] do not keep track of voltage and instead assume a constant value. In particular for super capacitor–based energy-harvesting systems, this assumption is not valid and leads to erroneous results. Accessing measurement values of SPOT is also done via I²C, but the related communication overhead (in terms of power and CPU utilization) is not evaluated. In contrast, ECO takes I²C cost carefully into consideration and can adapt to its different configurations.

By using an internal hardware counter and a signal of the switching regulator, iCount almost completely removes the need for additional hardware components and hence has low power

overhead. This comes at the cost of losing general applicability for different supply circuits and has the vital downside of requiring complex per-device calibration. It is also noteworthy that supply circuit properties and sampling rate affect the achievable resolution [54].

Nemo [54] brings an additional MCU-board, which guarantees good measurement accuracy, high sampling rates, and relatively low CPU overhead. The major downsides are its high complexity and power overhead. With 150 μA in sleep mode, it already consumes almost half the power of our setup during measurements. In active operation this increases to 4.6 mA, effectively requiring an additional power supply for long-term in situ deployments.

Nemo is focused on seamless integration into one specific platform and avoiding additional wiring by using modulation of the supply voltage to transmit its information to the measured node. The authors give an example of the overhead based on a use case of sampling at 0.5 Hz for 10 s and then sampling at 8 kHz for 10 ms while buffering all data on the measurement node. Transmitting the buffer to the observed node afterwards results in an average overhead of 0.6%. The communication bandwidth implies that the achievable continuous meter-to-node data rate is over 60 times slower than with the I^2C configuration we are using. For use cases, in which the node continuously requires timely information of its power consumption (e.g., when correlating power usage with software tasks), this slower data rate combined with buffering is inadequate.

From these observations, we conclude that specifically optimized custom solutions for specific use cases can slightly outperform our generic approach, but the commodity design of ECO shows to be significantly more versatile without sacrificing performance.

6 DEPLOYMENT SCENARIOS: EXPERIENCES WITH ECO IN THE WILD

After measurements and evaluations in our lab, we now validate ECO in real-world deployments, which target ubiquitous urban sensing scenarios. These experiments reflect common urban settings with direct access to local network infrastructure, but also rural and completely remote areas without any external infrastructure at all. To cover the requirements of both stationary deployment with dense local infrastructure and fully autarkic operation in the wild, we set up our system under very dissimilar conditions.

We implement two environmental monitoring applications under outdoor conditions, in which the ECO systems are deployed on a rooftop and on a public transport bus. Both scenarios aim for completely self-sustainable long-term operations. Solar harvesting enables perpetual, maintenance-free energy supply only limited by the lifetime of employed components. In both cases, tasks of environmental data collection have to meet weather-proof requirements. The rooftop scenario comprises a stationary deployment with steady local network coverage and a measurement task that shows little variation in execution consumption. In contrast to this, the bus deployment represents a mobile use case including high variations of the consumption of energy for measurements.

The power availability from the solar energy harvesting depends on temporary variations due to weather changes and slowly changing seasonal conditions. Influential factors for the mobile station such as the local placement and nearby structures like trees that block the sunlight create varying patterns over the days and need careful consideration. All of this imposes significant dynamics in power availability, which challenge the system.

6.1 Stationary Urban Sensing: Deployment on a Rooftop

For urban deployments with dense local network coverage (e.g., sensing at and around buildings), we deploy five individual ECO nodes for environmental sensing on a rooftop. All five nodes are equipped with different sensors to gather and report environmental data. Figure 10 shows three, (2), (3), and (4), of the five ECO sensors, together with the gateway (1).

Fig. 10. ECO outdoor deployment on a rooftop: ECO-boxes (2)–(4) connected via a LoWPAN gateway (1) to the Internet.

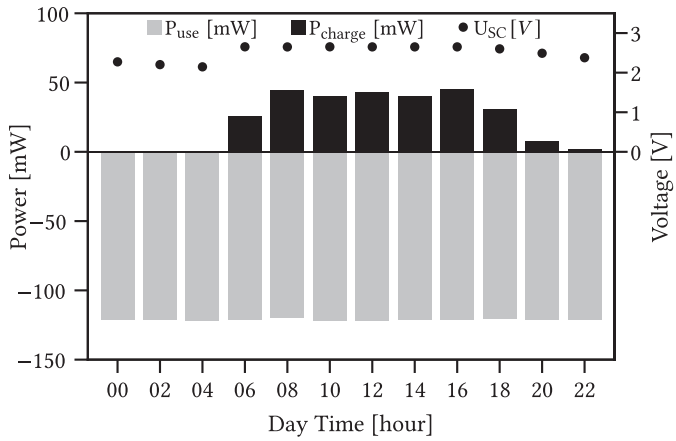

Fig. 11. Power consumption (P_{use}), charging (P_{charge}), and super capacitor voltage (U_{SC}) over one day observed on an ECO-enabled sensor node in static deployment.

We select sensors for temperature, humidity, pressure, and particulate matter air pollution. Power requirements and time taken to execute a measurement varies significantly between nodes because of the different nature of the sensor peripherals. An 802.15.4-based 6LoWPAN uplink is used for networking. The network is set up in a pure star topology, in which the gateway router is within range of every ECO node.

Even though the self-measurement and energy management are identical for all nodes, each ECO box needs to self-adapt to the different sensor requirements. Thanks to the self-measurement abilities, no prior power profiling of the individual components is required that parameterizes the power management of the nodes.

Figure 11 displays a typical course of the power availability during a day cycle. The gray bars correspond to power consumption of an active node, while the black bars represent the available charging power. The necessity for a duty cycling mechanism can be clearly seen from the much higher consumption compared to the charging rate. Due to the fixed position and static measurement tasks, the consumption variation is rather small. On a cloudy day, a prediction can adjust the expected energy available during the day as soon as a change in the charging rate compared

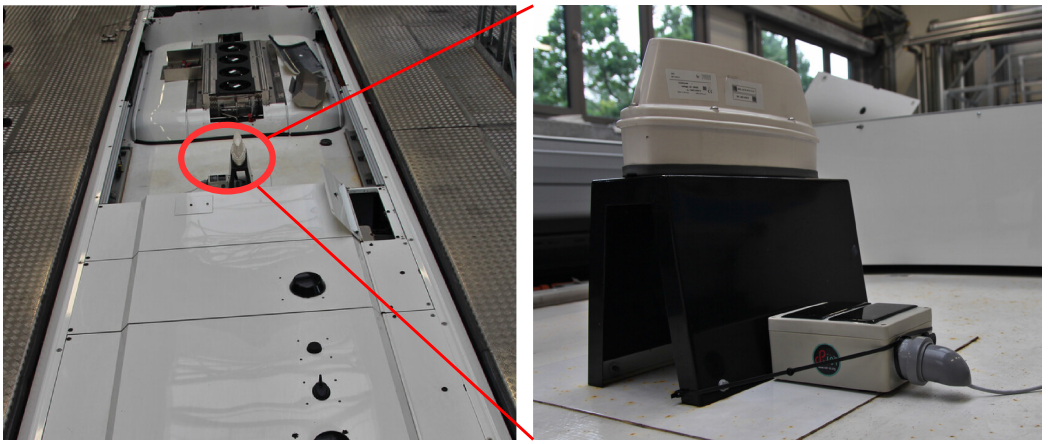

Fig. 12. ECO-box placement on an electric public transit bus.

to the previous day is detected. Additionally, public weather forecasts could be easily used in this static scenario to further improve prediction accuracy [32] but are ignored in our experiment.

In a mobile scenario, the power profile can look completely different. The node may move to a location where power is temporary not available. Sunlight may for example be blocked when moving indoors or when standing below a bridge. In such a setting, it becomes much more important to quickly react to changes of the charging rate.

6.2 Mobile Urban Sensing: Deployment on a Public Transport Bus

In our mobile setting, not only the network access becomes less reliable, but also the energy availability suddenly depends on local conditions as well as the movement in a constantly changing environment.

We face these challenges with a node sensing air pollution from within moving traffic. For this, we place an ECO sensebox on top of an electric public transit bus to measure the emissions of nearby vehicles. Figure 12 shows the placement on top of the bus. For this use case, it is a hard requirement that no infrastructure of the bus can be used (power, network, etc.). Hence, the sensor is plainly mounted to the roof without any connection to or modifications of the bus. Not having a separate system as external observer for monitoring, reconfiguring, or resetting the device demands very reliable operation.

The mobility also requires the sensor to identify its location with the help of a rather power-hungry GPS. Air measurements shall be linked to geographical coordinates. The GPS is also used to synchronize the nodal time with a global reference for time-stamping of the collected data.

Additional challenges of this deployment relate to the unpredictable operation of the bus, which serves different routes that change on daily demand. The bus is parked for charging regularly at a place, where measurements are not of major interest. For maintenance, it is parked in a garage where neither sunlight nor GPS reception or network connectivity are available. The times to readjust the GPS signal largely depend on movement and environmental factors, such as weather conditions and surrounding buildings. To achieve a reasonable range and coverage for connectivity in this setting, we use a LoRa-based LPWAN uplink via The Things Network (TTN) [11]. The node does not only transmit data of environmental measurements, but also statistics about the consumption of its individual activities to allow for collecting metrics related to the power management.

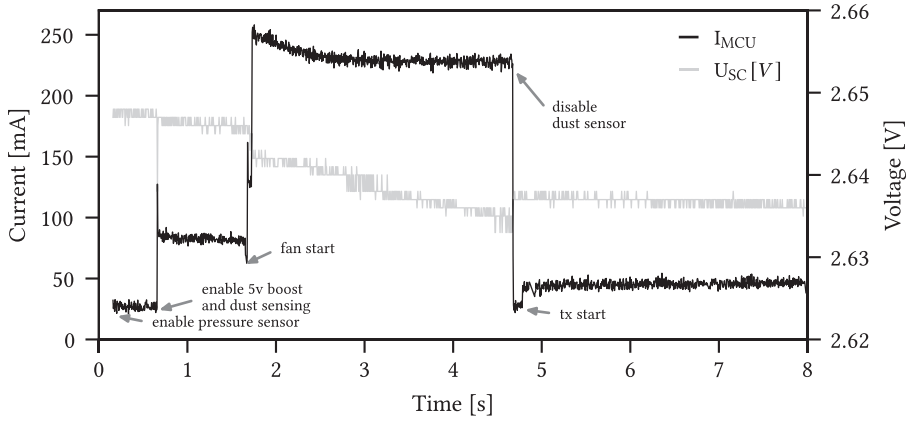

Fig. 13. Power trace of a dust sensor execution cycle.

In contrast to the coarse day course of power consumption shown in Figure 11, Figure 13 gives an example of a fine-grained trace of current and voltage for a node equipped with a particulate matter sensor. The trace shows how a pressure sensor is read after startup, followed by enabling a voltage converter to drive the power-demanding dust sensor and finishing with the data transmission. After reaching the highest consumption directly after the fan startup, it is clearly visible how the current slowly settles with the fan reaching its final speed. Thereafter, the dust sensor is disabled and the collected data is transmitted. Detailed traces like this include even tiny artifacts such as the visible short spikes that result from enabling the boost converter. With this precise self-measurement, the node can not only adjust itself to changing environmental conditions and peripheral consumption, but can also detect critical health conditions of components such as raising equivalent series resistance of the super cap or faulty behavior of mechanical components such as friction or blocking.

From the data collected, we see for example that the overall energy consumed for obtaining a GPS position varies largely with a relative standard deviation of above 70%. Yet, the self adaption of the system works very reliably for our deployment phase of more than a year. During this period, we encountered largely varying environmental conditions with temperatures ranging from −8 °C to 52 °C. Over 100,000 measurement cycles have been performed. Even with a huge battery in the order of a standard size laptop, this would not have been possible.

7 CONCLUSION AND OUTLOOK

In this article, we presented ECO, a hardware-software co-design that adds autonomous energy management to low-end IoT devices. We gave a detailed overview on challenges and related work in the specific domain of in situ self-measurement and energy attribution mechanisms that can be used in real production-scale deployments. ECO shows how the combination of a modular hardware setup and an OS-integrated, generic software can be used on a huge set of different platforms.

By measuring the different dimensions of overhead of the overall system, we were able to show the feasibility of our approach—even on very constrained 8-bit platforms. We tested our system against reality by deploying outdoor field trials of multiple energy-harvesting sensor nodes in two largely different application scenarios. The successful outcome of both field trials validates that ECO can be very valuable for building energy-aware sensor nodes. The self measurement in particular offers benefits in situations of high energy dynamics, or whenever the exact conditions

of the deployment are not known beforehand. When manufacturing tolerances or aging effects of components in the field lead to different consumption profiles across multiple devices, self measurement may also be preferable over tedious per-device profiling. Apart from the technical perspective, the generic system integration can be equally helpful for application designers that are not familiar with details of low-level MCU features for energy preservation.

In future work, we plan to add peripheral state tracking to the implicit thread measurement and layers for allocation and prediction that work hand-in-hand with our attribution layer. Further, we want to investigate how different application designs and architectures impact the applicability and accuracy of energy awareness provided by the operating system. An automated energy regression testing for the RIOT development community is on our schedule as well.

A Note on Reproducibility and Public Release

We explicitly support reproducible research [1, 44]. All of our work is intended for public release. We expect many of the system software to gradually enter the master branch of the RIOT open source operating system. The source code of our implementations (including HW design, scripts to setup the experiments, RIOT measurement apps, etc.) will be available on GitHub at https://github.com/inetrg/ECO.

REFERENCES

[1] Association for Computing Machinery. 2017. Result and Artifact Review and Badging. Retrieved from http://acm.org/publications/policies/artifact-review-badging.

[2] Muhammad Hamad Alizai, Qasim Raza, Yasra Chandio, Affan A. Syed, and Tariq M. Jadoon. 2016. Simulating intermittently powered embedded networks. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN'16)*. Junction Publishing, Canada, 35–40.

[3] Panagnos Anagnostou, Andres Gomez, Pascal A. Hager, Hamed Fatemi, José Pineda de Gyvez, Lothar Thiele, and Luca Benini. 2018. Torpor: A power-aware HW scheduler for energy harvesting IoT SoCs. In *Proceedings of the 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS'18)*. IEEE, New York, NY, 54–61.

[4] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet Things J.* 5, 6 (Dec. 2018), 4428–4440.

[5] Emmanuel Baccelli, Oliver Hahm, Mesut Günes, Matthias Wählisch, and Thomas C. Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. In *Proceedings of the 32nd IEEE INFOCOM. Poster*. IEEE Press, Piscataway, NJ, 79–80.

[6] Frank Bellosa. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop (EW'00)*. ACM, New York, NY, 37–42.

[7] Naveed Anwar Bhatti, Muhammad Hamad Alizai, Affan A. Syed, and Luca Mottola. 2016. Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. *ACM Trans. Sensor Netw.* 12, 3 (Aug. 2016), 24:1–24:40.

[8] C. Bormann, M. Ersue, and A. Keranen. 2014. *Terminology for Constrained-node Networks*. RFC 7228. IETF.

[9] Adriano Branco, Luca Mottola, Muhammad Hamad Alizai, and Junaid Haroon Siddiqui. 2019. Intermittent asynchronous peripheral operations. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys'19)*. ACM, New York, NY, 55–67.

[10] Bernhard Buchli, Felix Sutton, Jan Beutel, and Lothar Thiele. 2014. Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys'14)*. ACM, New York, NY, 31–45.

[11] TTN Community. 2020. The Things Network. Retrieved from https://www.thethingsnetwork.org/.

[12] Thanh Do, Suhib Rawshdeh, and Weisong Shi. 2009. pTop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power-aware Computing and Systems (HotPower'09)*. ACM, New York, NY.

[13] Adam Dunkels, Joakim Eriksson, Niclas Finne, and Nicolas Tsiftes. 2011. *Powertrace: Network-level Power Profiling for Low-power Wireless Networks*. Technical Report. Swedish Institute of Computer Science.

[14] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. 2004. Contiki—A lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the Conference on IEEE Local Computer Networks (LCN'04)*. IEEE Computer Society, Los Alamitos, CA, 455–462.

[15] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. 2007. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets'07)*. ACM, New York, NY, 28–32.

[16] Prabal Dutta, Mark Feldmeier, Joseph Paradiso, and David Culler. 2008. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN'08)*. IEEE Computer Society, Washington, DC, 283–294.

[17] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. 2008. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*. USENIX Association, Berkeley, CA, 323–338.

[18] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. 2019. Shepherd: A portable testbed for the batteryless IoT. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys'19)*. ACM, New York, NY, 83–95.

[19] Kai Geissdoerfer, Raja Jurdak, Brano Kusy, and Marco Zimmerling. 2019. Getting more out of energy-harvesting systems: Energy management under time-varying utility with PreAct. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks (IPSN'19)*. ACM, New York, NY, 109–120.

[20] Cenk Gündogan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2020. IoT content object security with OSCORE and NDN: A first experimental comparison. In *Proceedings of the 19th IFIP Networking Conference*. IEEE Press, Piscataway, NJ, 19–27.

[21] Cenk Gündogan, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2018. NDN, CoAP, and MQTT: A comparative measurement study in the IoT. In *Proceedings of 5th ACM Conference on Information-centric Networking (ICN'18)*. ACM, New York, NY, 159–171.

[22] Cenk Gündogan, Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. 2020. Designing a LoWPAN convergence layer for the information centric Internet of Things. *Comput. Commun.* 164, 1 (Dec. 2020), 114–123.

[23] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid prototyping for the batteryless Internet-of-Things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys'17)*. ACM, New York, NY.

[24] Xiaofan Jiang, Prabal Dutta, David Culler, and Ion Stoica. 2007. Micro power meter for energy monitoring of wireless sensor networks at scale. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*. ACM, New York, NY, 186–195.

[25] Xiaofan Jiang, Jay Taneja, Jorge Ortiz, Arsalan Tavakoli, Prabal Dutta, Jaein Jeong, David Culler, Philip Levis, and Scott Shenker. 2007. An architecture for energy management in wireless sensor networks. *SIGBED Rev.* 4, 3 (July 2007), 31–36.

[26] T. Capers Jones. 1984. Reusability in programming: A survey of the state of the art. *IEEE Trans. Softw. Eng.* 10, 5 (Sept. 1984), 488–494.

[27] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. 2007. Power management in energy harvesting sensor networks. *ACM Trans. Embed. Comput. Syst.* 6, 4 (Sept. 2007), 32–44.

[28] Giannis Kazdaridis, Ioannis Zographopoulos, Polychronis Symeonidis, Panagiotis Skrimponis, Thanasis Korakis, and Leandros Tassiulas. 2017. In-situ power consumption meter for sensor networks supporting extreme dynamic range. In *Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization (WiNTECH'17)*. ACM, New York, NY, 97–98.

[29] Keithley. 2016. Model DMM7510 7-1/2 Digit Graphical Sampling Multimeter Specifications. Retrieved from https://de.tek.com/sitewide-content/marketing-documents/m/o/d/model-dmm7510-7-1-2-digit-graphical-sampling-multimeter-specifications.

[30] Simon Kellner. 2010. Flexible online energy accounting in TinyOS. In *Proceedings of the Conference on Real-world Wireless Sensor Networks* (LNCS, Vol. 6511). Springer Berlin, 62–73.

[31] Hyung-Sin Kim, Michael P. Andersen, Kaifei Chen, Sam Kumar, William J. Zhao, Kevin Ma, and David E. Culler. 2018. System architecture directions for post-SoC/32-bit networked sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys'18)*. ACM, New York, NY, 264–277.

[32] Frank Alexander Kraemer, Doreid Ammar, Anders Eivind Braten, Nattachart Tamkittikhun, and David Palma. 2017. Solar energy prediction for constrained IoT nodes based on public weather forecasts. In *Proceedings of the 7th International Conference on the Internet of Things (IoT'17)*. ACM, New York, NY, 1–8.

[33] Olaf Landsiedel, Klaus Wehrle, and Stefan Gotz. 2005. Accurate prediction of power consumption in sensor networks. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNets'05)*. IEEE Computer Society, Washington, DC, 37–44.

[34] Martine Lenders, Peter Kietzmann, Oliver Hahm, Hauke Petersen, Cenk Gündogan, Emmanuel Baccelli, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. *Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things*. Technical Report arXiv:1801.02833. Open Archive: arXiv.org.

[35] Qiang Li, Marcelo Martins, Omprakash Gnawali, and Rodrigo Fonseca. 2013. On the effectiveness of energy metering on every node. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems (DCoSS'13)*. IEEE Computer Society, Washington, DC, 231–240.

[36] Peter Liggesmeyer and Mario Trapp. 2009. Trends in embedded software engineering. *IEEE Softw.* 26, 3 (Apr. 2009), 19–25.

[37] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. 2013. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN'13)*. ACM, New York, NY, 153–166.

[38] Roman Lim and Lothar Thiele. 2017. Testbed assisted control flow tracing for wireless embedded systems. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN'17)*. Junction Publishing, Canada, 180–191.

[39] NXP. 2014. *I^2C-bus Specification and User Manual*. Rev. 6. NXP Semiconductors.

[40] Joaquín Recas Piorno, Carlo Bergonzini, David Atienza, and Tajana Simunic Rosing. 2009. Prediction and management in energy harvested wireless sensor nodes. In *Proceedings of the 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology*. IEEE, New York, NY, 6–10.

[41] Christian Renner, Volker Turau, and Kay Römer. 2014. Online energy assessment with supercapacitors and energy harvesters. *Sustain. Comput.: Inform. Syst.* 4, 1 (Mar. 2014), 10–23.

[42] Michel Rottleuthner, Thomas C. Schmidt, and Matthias Wählisch. 2019. Eco: A hardware-software co-design for in situ power measurement on low-end IoT systems. In *Proceedings of the ACM SenSys, 7th International Workshop on Energy Harvesting & Energy-neutral Sensing Systems (ENSsys'19)*. ACM, New York, 22–28.

[43] Rinalds Ruskuls and Leo Selavo. 2010. EdiMote: A flexible sensor node prototyping and profiling tool. In *Proceedings of the Conference on Real-world Wireless Sensor Networks* (LNCS, Vol. 6511). Springer Berlin, 194–197.

[44] Quirin Scheitle, Matthias Wählisch, Oliver Gasser, Thomas C. Schmidt, and Georg Carle. 2017. Towards an ecosystem for reproducible research in computer networking. In *Proceedings of the ACM SIGCOMM Reproducibility Workshop*. ACM, New York, NY, 5–8.

[45] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. 2004. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, 188–200.

[46] Lukas Sigrist, Andres Gomez, Roman Lim, Stefan Lippuner, Matthias Leubin, and Lothar Thiele. 2017. Measurement and validation of energy harvesting IoT devices. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE'17)*. European Design and Automation Association, Leuven, Belgium, 1159–1164.

[47] Philipp Sommer and Branislav Kusy. 2013. Minerva: Distributed tracing and debugging in wireless sensor networks. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys'13)*. ACM, New York, NY, 12:1–12:14.

[48] Thad E. Starner. 1996. Human-powered wearable computing. *IBM Syst. J.* 35, 3.4 (1996), 618–629.

[49] Sujesha Sudevalayam and Purushottam Kulkarni. 2011. Energy harvesting sensor nodes: Survey and implications. *IEEE Commun. Surv. Tutor.* 13, 3 (Mar. 2011), 443–461.

[50] Matthew Tancreti, Mohammad Sajjad Hossain, Saurabh Bagchiand, and Vijay Raghunathan. 2011. AVEKSHA: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys'11)*. ACM, New York, NY, 288–301.

[51] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. IEEE Press, Piscataway, NJ, 477–482.

[52] Christopher M. Vigorito, Deepak Ganesan, and Andrew G. Barto. 2007. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *Proceedings of the 4th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'07)*. IEEE, New York, NY, 21–30.

[53] Hong Zhang, Mastooreh Salajegheh, Kevin Fu, and Jacob Sorber. 2011. Ekho: Bridging the gap between simulation and reality in tiny energy-harvesting sensors. In *Proceedings of the 4th Workshop on Power-aware Computing and Systems (HotPower'11)*. ACM, New York, NY, 9:1–9:5.

[54] Ruogu Zhou and Guoliang Xing. 2013. Nemo: A high-fidelity noninvasive power meter system for wireless sensor networks. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN'13)*. ACM, New York, NY, 141–152.

Received April 2020; revised September 2020; accepted December 2020