Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things*

Kai Geissdoerfer†
Nessie Circuits
kai.geissdoerfer@nessie-circuits.de

Marco Zimmerling
TU Darmstadt
marco.zimmerling@tu-darmstadt.de

ABSTRACT

The rapidly growing Internet of Things (IoT) can avoid the high cost and environmental burden of replacing trillions of batteries by using sustainable battery-free devices that operate maintenance-free for decades. To develop battery-free IoT systems, researchers and makers require a common platform that is versatile, affordable, and easy to use. However, limited availability and lack of support have prevented widespread adoption of previous battery-free platforms. We introduce Riotee, an open-source and commercially available battery-free platform that includes multiple boards, extensive software, and comprehensive documentation. We demonstrate Riotee's capabilities through a machine-learning application and present results from a user study involving students and customers, who rated its usefulness and usability highly.

CCS CONCEPTS

• **Hardware** → **Sensor devices and platforms**; • **Computer systems organization** → **Embedded software**.

KEYWORDS

Battery-free systems, intermittent computing, sustainability, hardware and software platform, open source, reusable, accessible

ACM Reference Format:
Kai Geissdoerfer and Marco Zimmerling. 2024. Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things. In *The 22nd ACM Conference on Embedded Networked Sensor Systems (SenSys '24), November 4–7, 2024, Hangzhou, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3666025.3699332

1 INTRODUCTION

The rapid proliferation of embedded devices has led to significant challenges, particularly in terms of limited battery lifespan, high installation and maintenance costs, and increasing electronic waste. Addressing these issues is critical as the number of connected devices is projected to reach one trillion by 2035 [39]. These challenges hinder the scalability of IoT systems and pose serious environmental and economic concerns. Therefore, sustainable alternatives are needed to support the growing ecosystem of connected devices.

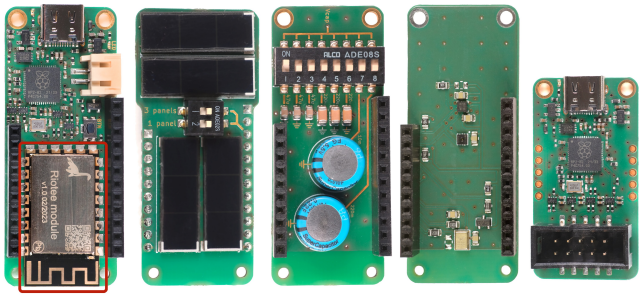

Figure 1: Riotee hardware. *A stamp-sized reflow-solderable module, a user-friendly development board, three expansion shields, and a probe for in-circuit programming and debugging.*

Problem. In this context, the community has explored the potential of battery-free devices, which buffer energy harvested from ambient sources such as sunlight, motion, and thermal gradients in small capacitors to power their operation. Over the past decade, researchers have developed various methods for battery-free devices, yet an accessible, general-purpose platform is still lacking [2].

Validation on real hardware is an integral part of experimental research and education. Similarly, makers and hobbyists rely on well-defined, easy-to-acquire platforms to develop new applications. For battery-powered devices, numerous options exist: Arduino, Raspberry Pi, NVIDIA Jetson, etc. These platforms are readily accessible and cater to the versatile needs of developers and researchers alike.

Unfortunately, none of the existing battery-free platforms meet these requirements. Commercial battery-free platforms like EnOcean STM 400J [32], Wiliot Pixel [33], and Everactive Eversensor [15] are unsuitable for research and education due to undisclosed hardware designs, proprietary software, and licensing limitations. Conversely, most battery-free platforms used in academia are custom-made for one specific deployment [1] or method [11, 17, 20]. Because of their narrow focus and lack of documentation and software support, these platforms are not easily reusable and accessible. Our literature review (Section 2) reveals that, despite open-source hardware designs, the few efforts toward a general-purpose battery-free platform [3, 16, 21] have not been used beyond the original paper. The key distinction is that these platforms require assembly from design files, whereas platforms like Arduino or Raspberry Pi are available for online purchase.

This shows that while open-sourcing hardware is an important step, it alone does not make a platform easily usable by others. Even with well-documented schematics, layouts, and bills of materials, reproducing a multi-layer printed circuit board (PCB) design

*See https://riotee.nessie-circuits.de for hardware designs, code, and documentation.
†Work performed while at Networked Embedded Systems Lab, cfaed, TU Dresden.

SenSys '24, November 4–7, 2024, Hangzhou, China

ACM ISBN 979-8-4007-0697-4/24/11.
https://doi.org/10.1145/3666025.3699332

Table 1: State-of-the-art battery-free platforms used in academia. *While most battery-free platforms are available as open source, they are often tailored to a specific application (i.e., deployment scenario or proposed method) and not commercially available. A lack of documentation and software support further hinders the widespread adoption of existing battery-free platforms.*

Name	Year	General purpose	Purchasable	Open source	Software	Documentation
WISP [35]	2008	✗	✓	✓	✓	✗
DEBS [20]	2016	✗	✗	✗	✗	✗
Flicker [21]	2017	✓	✗	✓	✗	✗
Capybara [8]	2018	✗	✗	✓	✓	✗
PiBLE [16]	2018	✓	✗	✓	✗	✗
BFree [26]	2020	✗	✗	✓	✓	✗
Botoks [11]	2020	✗	✗	✓	✓	✓
Empire [1]	2020	✗	✗	✗	✗	✗
Flync [17]	2021	✗	✗	✓	✓	✗
SuperSensor [3]	2023	✓	✗	✓	✓	✗
This work: Riotee	2024	✓	✓	✓	✓	✓

with numerous unique components and layout-sensitive radio frequency (RF) circuitry is challenging. Variations in design requirements between manufacturers and shortages of specific electronic components often necessitate partial redesigns, which are time-consuming and require deep technical expertise. Along with long lead times and high costs for PCB assembly in small quantities, this makes building a platform from design files nearly as expensive and effort-intensive as designing a new platform from scratch. Consequently, numerous research groups worldwide invest countless hours and substantial portions of their budgets in hardware development—a significant investment that could otherwise be directed toward advancing science and education. Moreover, many researchers lack the resources for such endeavors, preventing them from implementing and evaluating their ideas on real hardware.

Contribution. To tackle this issue and establish a common battery-free platform for researchers and makers alike, we present Riotee. Riotee is open-source *and* commercially available, enabling experimental evaluation of ideas, replication and comparison of results, and deployment setup without the need for hardware assembly. By providing this missing piece of foundational infrastructure, Riotee aims to foster community growth and accelerate progress toward a sustainable IoT. Overall, we make the following key contributions:

- We design and build the Riotee hardware (Figure 1), consisting of six units that seamlessly enable battery-free development from first prototypes to real deployments.
- We design and implement the Riotee software, featuring an efficient battery-free runtime, various peripheral drivers, two network protocols, an easy-to-use Arduino API, and extensive application examples.
- We demonstrate Riotee's utility through an intermittent deep neural network (DNN) inference application.
- A systematic user study reveals that both customers and students highly rate the usefulness and usability of Riotee.
- We provide comprehensive online documentation of the Riotee hardware and software components, including pinouts, API reference, in-depth technical descriptions, and getting started and troubleshooting guides.

2 CURRENT BATTERY-FREE PLATFORMS

Table 1 provides an overview of state-of-the-art battery-free platforms used in academia. It shows that new battery-free platforms are proposed frequently. But why has none of them been widely adopted by the research community? One reason is that most platforms have been introduced to validate one specific new method, compromising on other aspects essential for general-purpose usage. For instance, Botoks [11] incorporates a cascaded resistor-capacitor (RC) timekeeper that occupies 1 square inch of PCB area, allowing users to trade off measurement range for resolution. Flync [17] is a platform custom-designed for battery-free device-to-device communication, featuring a dedicated clock extraction circuit drawing 5 µW of power. WISP [35] is a radio-frequency identification (RFID) sensing platform. While available for online purchase and used in numerous earlier works, it only functions with a high-powered RFID reader nearby.

A few serious efforts toward a general-purpose battery-free platform have been made in recent years [3, 16, 21]. Unfortunately, none of these platforms has been used beyond the original paper or managed to attract a significant user base. We maintain that this is largely due to insufficient availability of the hardware and lack of software and documentation, as indicated in Table 1. For example, Flicker [21] is an ecosystem of plug-in boards based around the dated MSP430. It caters to various scenarios yet suffers from limited availability, software support, and documentation. SuperSensor [3], an advancement of Flicker, facilitates easy component exchange, including the main microcontroller unit (MCU). However, the SuperSensor hardware is not readily available, and reconstructing it from open-source design files is challenging due to reasons described in Section 1.

Our review of the state of the art indicates that a general-purpose, readily accessible platform for research and education in battery-free, intermittent systems is still absent. This lack of foundational infrastructure hinders rigorous evaluations and community growth, ultimately impeding progress toward a more sustainable future of computing [2].

3 RIOTEE OVERVIEW

To fill this gap and establish a common battery-free platform, we design Riotee with the following key goals:

- *Availability.* To foster widespread adoption, the platform must be commercially available at an affordable price comparable to existing battery-powered platforms. For example, Arduino boards currently sell for between USD 20 and USD 114 [40]. Additionally, hardware design and software should be open-source, enabling others to study, verify, and modify them.
- *Easy of use.* To establish a large user community, the platform must be user-friendly. Thorough documentation of hardware and software and a development environment based on standard cross-platform tools are crucial. The time and effort required to get a simple application up and running must be minimal to attract new users. Moreover, the platform should support as much basic functionality as possible, allowing users to focus on their projects. For example, a developer working on a new time-keeping solution should not have to write a radio driver.
- *Versatility.* Battery-free technology is not one-size-fits-all. Different applications require different harvesters, peripherals, and energy storage capacities. A general-purpose platform should integrate core components while allowing users to easily adjust application-specific parameters and hardware components (e.g., new types of solar cells and energy storage capacitors).
  Moreover, while it is impossible to foresee all use cases, the platform's design should consider the needs and requirements of various sub-fields in battery-free research. For example, many works on intermittent computing require fast, long-lasting non-volatile memory [4, 31, 43]. Reactive checkpointing techniques rely on external notifications of an impending power loss [4, 10, 24]. The emerging field of battery-free networking requires low-level access to a radio [14, 18, 28]. Researchers working on energy harvesting-based sensing need access to the voltage and current of the harvester [25, 37]. Future applications and deployments of battery-free devices likely require a combination of all these features [1, 9].
- *Deployability.* A successful platform must offer a convenient development board equipped with an on-board debugger, buttons, light emitting diodes (LEDs), pin headers, and expansion boards. While this results in a relatively large platform with a higher price tag, it is essential for development and testing. On the other hand, the transition from a prototype to a real deployment scenario—where size, weight, and cost constraints are often critical—should be as seamless as possible.

Guided by these design goals, we present Riotee. It is both open-source and commercially available, aiming to empower researchers and makers to experimentally evaluate ideas, replicate results, and set up deployments without the need to build battery-free hardware first. Riotee consists of multiple boards accompanied by software and documentation.

Hardware. The Riotee Board (Section 4.1) is an easy-to-use development board that combines the key components of a battery-free device with an on-board programmer, a push button, LEDs, and a USB-C port. Using the board's expansion headers, various harvesters, capacitors, and sensors can be added by stacking multiple add-on shields we provide (Section 4.4). The core components of the Riotee Board—such as the harvesting circuitry, microcontroller, non-volatile memory, wireless radio, and antenna—are all encapsulated in a tiny module known as the Riotee Module (Section 4.2). This module can be reflow-soldered onto a custom PCB and programmed with the Riotee Probe (Section 4.3) to meet the size, weight, and cost constraints of real deployments.

Software. The Riotee hardware is accompanied by a well-documented, open-source software development kit (SDK) that includes a battery-free runtime (Section 5.1), various peripheral drivers (Section 5.2), and several application examples. Furthermore, an Arduino package (Section 5.3) provides a simplified application programming interface (API) and cross-platform installation with just a few mouse clicks.

Documentation. Riotee comes with extensive, well-prepared documentation. Doxygen-style code comments are combined with hardware descriptions, pinouts, and examples. These are rendered together with getting started and troubleshooting guides as a publicly hosted documentation website.

The next two sections detail Riotee's hardware and software design, and evaluate key performance characteristics. Section 6 demonstrates the capabilities and utility of Riotee through a real-world use case, while Section 7 presents a user study that evaluates the usability of Riotee among customers and students. The results of this study indicate that both user groups highly rate the usefulness and usability of Riotee.

4 HARDWARE

This section describes the six boards comprising the Riotee hardware shown in Figure 1, starting with the Riotee Board.

4.1 Riotee Board

The Riotee Board is a user-friendly development board integrating key battery-free device components with an on-board programmer, push button, LEDs, USB-C port, and expansion headers. This setup facilitates rapid prototype creation without soldering or breadboard wiring. The core of the Riotee Board is the Riotee Module, which is combined with the same circuitry found on the Riotee Probe for uploading and debugging code running on the module. The board's two 0.1-inch pin headers expose signals for supply voltage, capacitor voltage, and 11 general-purpose input/output (GPIO) pins supporting I2C, SPI, and analog sensor applications. These headers allow connecting custom harvesters, storage capacitors, or peripherals and are also used to connect the Riotee Shields. Below, we describe the Riotee Module, Probe, and Shields in detail.

4.2 Riotee Module

The Riotee Module is a stamp-sized unit (27.2 mm by 15.2 mm) that can be soldered onto a user's PCB with additional circuitry like sensors or harvesters to create deployment-ready battery-free devices quickly. Figure 2 shows the architecture of the Riotee Module. It integrates harvesting circuitry, two MCUs, a real-time clock (RTC), a PCB antenna, and an LED on a compact 4-layer PCB enclosed in a metal RF shield. The module features breadboard-compatible castellated holes exposing essential signals, including 11 GPIOs (D0-D11), two of which can also function as analog inputs (A0, A1).

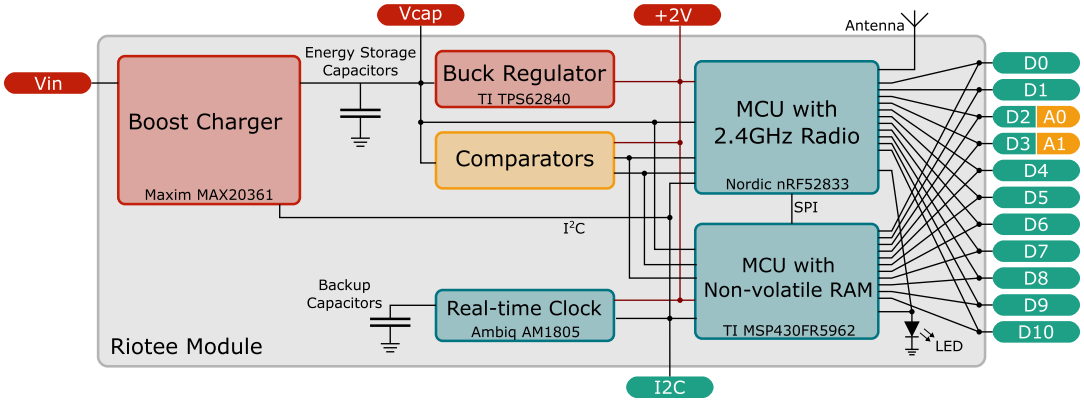

Figure 2: Architecture of Riotee Module. *The Riotee Module integrates energy harvesting, energy storage, power management, non-volatile memory, a powerful Cortex-M4 processor, and a* 2.4 GHz *BLE-compatible radio into a stamp-sized hardware unit.*

Dual microcontroller. The module hosts two MCUs. The first one, a Nordic Semiconductor nRF52833, has a 64 MHz Cortex-M4 with a floating point unit and a low-power 2.4 GHz radio with low-level register access. The Cortex-M4 is significantly faster than the MSP430 found on most previous battery-free platforms, enabling computationally demanding applications like machine learning inference with minimal current draw. The second MCU, a TI MSP430FR5962, offers 128 kB of embedded non-volatile FRAM with virtually unlimited write cycles. Its popularity in the intermittent computing community supports leveraging existing knowledge and experience, reusing software components, and ensuring reproducibility and comparability with prior research.

Both MCUs are user-programmable and share access to all module components. For example, the capacitor voltage is available on the on-board analog-to-digital converter (ADC) inputs of both MCUs. The two MCUs are interconnected via a 4-wire SPI bus and a handshake line, supporting various configurations. For instance, the Riotee SDK runs application and networking code on the nRF52, using the MSP430 as SPI-based non-volatile RAM to retain state across power outages. Alternatively, users can run application code on the MSP430 and use the nRF52 as an SPI-controlled radio.

Harvesting circuitry. An Analog Devices MAX20361 boost charger tracks the maximum power point (MPP) of the harvester from 225 mV to 2.5 V and charges three parallel 22 µF 0402-sized multilayer ceramic capacitors (MLCCs) up to 4.7 V. Due to the DC-bias effect on MLCCs, the three 22 µF capacitors have an effective total capacitance of about 11.1 µF at 4.7 V, which is sufficient to continuously power the nRF52 radio for approximately 1.5 ms from a full charge.

This harvesting interface supports any direct current (DC) voltage source within the specified input range, including solar panels, thermoelectric harvesters, DC wind generators, and, after rectification, piezoelectric and RF harvesters. The I2C interface of the MAX20361 allows users to dynamically control the harvesting voltage in software. This unique feature enables users to either dynamically adjust the parameters of the built-in open-circuit voltage maximum power point tracking (MPPT) algorithm or implement and experiment with their own custom on-device MPPT algorithms.

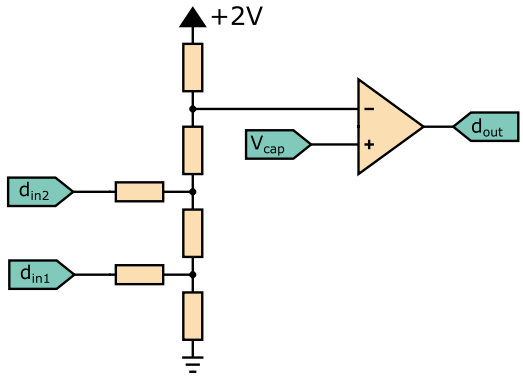

Figure 3: Diagram of resistor network. *Using two tri-state GPIO signals, users can dynamically select the suspend and resume thresholds,* V_{sus} *and* V_{res}*, from nine possible values.*

A TI TPS62840 buck regulator steps down the capacitor voltage to a constant 2 V supply that powers the two MCUs and the peripherals. The regulator activates when the capacitor voltage reaches the fixed 3.7 V wake-up threshold, V_{on}, of the MAX20361 and deactivates when the voltage drops below the fixed 1.75 V low-voltage lockout, V_{off}.

Comparators. Riotee implements a *soft intermittency* approach [10, 17], where the device avoids energy-intensive resets by gracefully suspending execution and entering a low-power sleep mode before the capacitor voltage drops below V_{off}. The energy required for each uninterrupted execution phase, or *energy burst*, depends on the application and may vary dynamically. Larger energy bursts allow longer continuous execution but require longer charging times.

To enable dynamic control of the burst size, a dual comparator monitors the capacitor voltage and outputs two digital signals indicating whether the voltage is below, between, or above the suspend threshold, V_{sus}, and the resume threshold, V_{res}. Figure 6 shows these thresholds along with the fixed V_{on} and V_{off} thresholds. A resistor network, inspired by traditional R2-R digital-to-analog converters (DACs), provides reference voltages to the comparators.

This network, shown in Figure 3, consists of six 0201-sized resistors, which are compact, significantly cheaper, and draw an order of magnitude less current than the rheostats used in other designs [3].

The two tri-state GPIOs signals, d_{in1} and d_{in2}, controlling the network can be set to High, Low, or High-Z, resulting in nine input states. We define target thresholds from 2.5 V to 4.1 V in 200 mV steps for V_{sus} and from 3.0 V to 4.6 V for V_{res}, based on the maximum capacitor voltage and V_{off}. Using mesh network analysis and brute-force optimization, we select resistor values from the E24 series that minimize the mean squared error (MSE) of the thresholds.

Energy harvesting-based sensing. Energy harvesting-based sensing [25, 37] can reduce the size, weight, cost, and energy consumption of a device by eliminating dedicated sensors and instead extracting information about a physical process from the harvester's parameters. Riotee supports such applications by providing access to the voltage and current of the attached harvester. The *harvesting counter* aboard the MAX20361 counts the switching cycles of the boost converter within a configurable time period, directly proportional to the harvesting current for any voltage. The MAX20361 also allows reading the open-circuit voltage and the currently set harvesting voltage, enabling sampling of the energy-harvesting signal at up to 20 Hz. Additionally, the capacitor voltage is available on the ADC/comparator inputs of both MCUs through a voltage divider and op-amp buffer.

Current measurement. Measuring current draw during application development is crucial for optimizing the use of harvested energy. To simplify this process for battery-free devices, the Riotee Module allows current measurement on the 2 V power supply using an external ammeter. By pulling the bypass signal high and connecting an ammeter between the 2 V (A_{on}) and V_{shunt} pads on the back of the module, developers can measure dynamic current consumption without affecting the device's real-world operation.

Timekeeping. Keeping track of time is essential. Traditional battery-powered devices use a counter driven by a low-power oscillator that runs continuously and may be periodically synchronized to an external reference clock. On battery-free devices, this is challenging because power outages stop the oscillator, preventing the counter from incrementing. After waking up, the device cannot determine the elapsed time.

To address this, Riotee features an ultra-low-power Ambiq AM1805 RTC with backup capacitors. In case of a power outage, the RTC switches to the backup supply, supported by two parallel 22 µF capacitors that sustain the 14 nA current draw for over 3 min. This solution occupies just 30 mm^2 and costs less than USD 1.50. The soft intermittency approach mitigates the disadvantage of long startup times, as Riotee suspends execution gracefully before a power outage, requiring the RTC to reinitialize only after extended periods without energy input. The RTC also includes a software-controlled switch to cut power to all parts except itself, minimizing current while waiting for a timer to expire. When the timer expires, the switch closes, and the system restarts.

4.3 Riotee Probe

Uploading and debugging firmware on battery-free devices is challenging. Harvester power is generally insufficient for flashing new firmware, and the additional current consumption and leakage through programming pins of off-the-shelf dongles can interfere with device behavior. However, connecting the device to a constant power supply prevents observing its behavior under *real* harvested energy conditions.

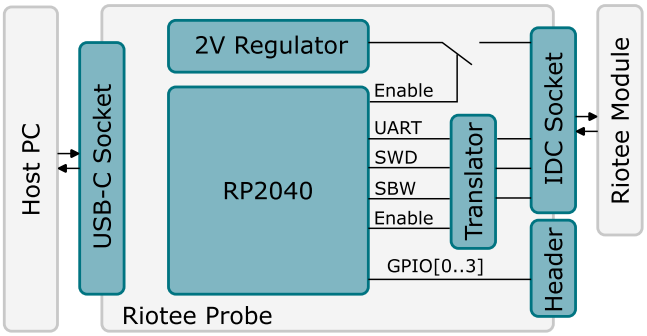

Figure 4: Architecture of Riotee Probe. *An RP2040 translates requests received via USB into SWD and SBW sequences on pins that control the MCUs on the Riotee Module. A 2 V regulator supplies the Riotee Module during programming and debugging. Four GPIO lines facilitate additional functionality.*

The Riotee Probe supports programming both the MSP430 and nRF52 on the Riotee Module while switching between constant supply mode and untethered battery-free operation. Figure 4 shows the architecture of the Riotee Probe. On the host side, the Raspberry Pi RP2040 MCU provides an ARM CMSIS-DAP compatible interface via USB-C, compatible with standard software tools like pyOCD and OpenOCD. On the target side, it exposes a standard 10-pin, 0.1-inch connector compatible with Tag-Connect cables for in-circuit debugging with minimal footprint. Pin-enabled level shifters translate between the 3.3 V supply of the RP2040 and the 2 V supply of the Riotee Module.

We develop a ZephyrOS-based firmware for the RP2040 that translates USB requests into corresponding transfers on the programming pins. At the beginning of each transfer, the constant power supply is enabled, and the programming signals are connected to the device via analog switches. After programming, the power supply and programming pins are disconnected, allowing the device to return to harvesting operation and enabling immediate observation of the new firmware's operation without interference. Users can also enable the constant power supply with a dedicated command.

We extend the default ARM CMSIS-DAP commands for programming and debugging the nRF52 with custom vendor commands for resetting and halting the MSP430 and for reading and writing its memory via TI's Spy-Bi-Wire (SBW) protocol. This allows programming both chips through a unified Python interface running on the host PC.

The Riotee Probe also acts as a USB-UART bridge, forwarding serial data between the host and a Riotee device in both untethered battery-free and constant power supply modes. Additionally, it has four GPIOs controllable via USB from the command-line interface or API. This allows building test jigs to automatically program and test Riotee-based devices or implementing specialized debugging mechanisms [7, 12].

Table 2: Performance specification of Riotee Module. *Values with a * are from datasheets, all others are measured.*

Parameter	Value
Idle current	4.8 µA
Deep sleep current	72 nA
RTC holding time	210 s
Threshold accuracy	0.097 V
Comparator circuit current	1.45 µA
MSP430 CPU active current*	3 mA
nRF52 CPU active current*	5.6 mA
Radio TX 1 Mbps 0 dBm current*	11.0 mA
Radio RX 1 Mbps current*	10.5 mA
Harvesting voltage range*	0.225 V to 2.5 V
Harvesting power range*	0.015 mW to 300 mW
Output current 2 V*	750 mA
Capacitor voltage range*	0 V to 4.7 V

4.4 Riotee Shields

The Riotee Board integrates all necessary components required to build a fully functional device with the exception of the harvester. Adopting the extensibility approach from highly successful platforms like Arduino and Raspberry Pi, we provide three shields that plug into the Riotee Board's headers, facilitating rapid battery-free device prototyping and serving as a reference for users designing their own custom shields.

Solar Shield. This shield features four AnySolar KXOB25x03F solar panels and a two-position sliding switch that connects none, one, three, or all four panels in parallel. This configuration allows easy experimentation with different harvesting currents under given ambient light conditions.

Capacitor Shield. This shield includes eight capacitors of different values and dielectrics: 3× 47 µF MLCCs, 3× 220 µF MLCCs, 1× 10 mF electric double-layer capacitors (EDLC), and 1× 220 mF EDLC. An eight-position sliding switch enables selection from 256 possible capacitor combinations to meet application needs. It also helps users (e.g., students) observe the impact of energy storage size on system behavior.

Sensor Shield. This shield integrates a Bosch BMA400 accelerometer, a Sensirion SHTC3 digital temperature and humidity sensor, and a Vesper VM1010 analog microphone with a power switch. The Sensor Shield enables quick assembly of real sensing applications and is used in some software examples and our hot-word detection case study in Section 6.

4.5 Performance Specification

Table 2 lists key performance metrics of the Riotee Module.

Input and output ranges. Riotee's harvesting voltage and power ranges are ideal for single- or multi-cell solar harvesting. Additionally, the boost converter interface is compatible with RF [41], thermal [38], and kinetic energy harvesting [36]. The buck regulator's output current supports power-hungry peripherals, such as satellite modems and small motors.

Table 3: Current selling prices of Riotee hardware.

Product	Price (USD)
Riotee Module	59
Riotee Board	99
Riotee Probe	56
Riotee Solar Shield	39
Riotee Capacitor Shield	39
Riotee Sensor Shield	39

Capacitor voltage range. The capacitor voltage range supports various storage elements. By reducing the maximum storage voltage through a command to the MAX20361 boost charger, Riotee is also compatible with lower voltage supercapacitors and rechargeable batteries.

Current draw. We utilize the current measurement feature of the Riotee Module to measure its current draw with a Keithley SMU2604B. Since the current is measured at the output of the buck regulator, it includes the consumption of both MCUs, the RTC, and the comparator circuit but excludes the quiescent currents of the boost converter and buck regulator. Idle current is measured with both MCUs in low-power mode, the RTC aboard the nRF52 running, and the comparator circuit active. Deep sleep current is measured while the AM1805 RTC is running and disconnecting the remaining components of the system (see Section 4.2).

The measured idle current is 4.8 µA, significantly lower than previous platforms, which draw at least 6.25 µA [21] and 10 µA [3]. The 72 nA power-down current is comparable to the leakage current of typical MLCC and EDLC.

RTC holding time. To measure the duration that the RTC can operate during a power outage, we set the RTC to an arbitrary date, disconnect the harvester for a specified duration, reconnect the harvester, and check if the RTC value matches the initial value plus the outage duration. This experiment is repeated with increasing power outage times until the RTC returns an incorrect time due to power failure. Table 2 lists the longest power outage after which the RTC still returns the correct time. The measured 210 s significantly exceed the useful range of state-of-the-art remanence-based timekeeping solutions [11, 13].

Threshold accuracy. To evaluate the accuracy of the comparator thresholds generated with the resistor network in Figure 3, we power a Riotee Module from a constant harvesting source, iterate through all possible V_{res} and V_{sus} thresholds, and measure the capacitor voltage when the corresponding interrupt triggers using the internal ADC. We then compute the absolute difference between the measured value and the target threshold. The median deviation measured is 0.013 V, with a maximum deviation of 0.097 V.

4.6 Pricing

Riotee is a first-of-its-kind platform offering high versatility, performance, and low-power operation in a compact package suitable for deployments. Integrating two MCUs, an RTC, a boost converter, a buck charger, an op-amp, a dual comparator, three transistors, an LED, and 77 passive components on the tiny Riotee Module necessitates the use of small-pitch components (e.g., nRF52833 in a

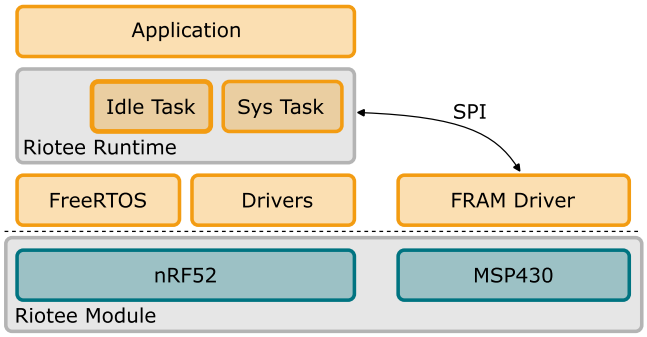

Figure 5: Riotee software architecture. *The application executes on the nRF52 as a FreeRTOS task. When energy gets low, the runtime activates a high-priority system task that checkpoints application state to non-volatile FRAM on the MSP430.*

350 µm WLCSP package) and high-density PCB design. Advanced manufacturing techniques, such as laser-drilled blind and buried vias, are prohibitively expensive in low volumes. However, manufacturing the hardware in larger batch sizes allows us to distribute these costs across many PCBs, making Riotee affordable. Table 3 lists the current selling price of the Riotee hardware. The price of USD 99 for the Riotee Board is comparable to typical battery-powered devices like Arduino (USD 20 to USD 114), TelosB (USD 82), or OpenMote (USD 128).

5 SOFTWARE

In addition to the Riotee hardware, we provide a well-documented, open-source SDK that includes a battery-free runtime, various peripheral drivers, two network protocols, and extensive application examples. The SDK, implemented in C with basic C++ support, features a public API with doxygen-style comments, contextual documentation, and several examples, all rendered on a publicly hosted website. An Arduino package provides a simplified SDK and cross-platform installation with just a few mouse clicks. The SDK is hosted on GitHub, where users are encouraged to submit issues and commits. Continuous integration (CI) and continuous deployment (CD) pipelines automatically test the compilation of the examples and handle the deployment of semantically versioned releases to a web server after new commits. Additionally, we offer a GitHub template repository for developing applications with the SDK in Visual Studio Code. This section details the Riotee software starting with the runtime.

5.1 Runtime

The battery-free runtime is at the core of the Riotee SDK, providing an efficient and flexible foundation for developing battery-free applications. The runtime's primary responsibility is to retain state across inevitable power failures, including all variables and register values representing the state of the application, network stack, and peripheral drivers. This process, known as *checkpointing*, involves copying data between volatile memory/registers and non-volatile memory.

Table 4: Callbacks of Riotee runtime. *The user interacts with the runtime mainly through a number of callbacks.*

Callback	Called …
earlyinit()	right after every reset
lateinit()	later after every reset
bootstrap()	once after programming the device
suspend()	before the application is suspended
resume()	before the application is resumed

Figure 5 provides an overview of the software architecture. In our current implementation, the application and the Riotee runtime execute on the more powerful nRF52, while the MSP430 serves as a SPI-controlled non-volatile memory.

To write an application, the user implements a *main()* function and several callback functions listed in Table 4. Using a typical capacitor voltage profile, Figure 6 illustrates the execution sequence of these functions as described below:

1. The 2 V power supply turns on as the capacitor voltage reaches the wake-up threshold, V_{on}. Immediately after the MCU starts up, the optional *earlyinit()* callback allows attached peripherals to be put into a low-power mode to avoid excessive current draw, which may prevent the application from ever starting up. Next, the MCU enters a low-power mode until the capacitor voltage reaches the resume threshold, V_{res}.
2. Execution continues with the *lateinit()* callback. This is the place to perform basic peripheral configuration that applies throughout the entire application execution, such as setting up serial output or initializing the ADC. Next, the *main()* function starts executing.
3. When the capacitor voltage drops below the suspend threshold, V_{sus}, the runtime suspends code execution and puts the device into low-power mode. This soft intermittency approach reduces the energy overhead associated with each store, reset, and restore cycle of traditional intermittent systems. An optional *suspend()* callback can turn off any power-hungry peripherals before entering low-power sleep mode.
4. If the harvesting current exceeds the sleep current, the capacitor voltage rises again. When it reaches the resume threshold, V_{res}, the *resume()* callback is called, and execution of the *main()* function resumes.
5. If the capacitor voltage does not recover above V_{sus} within the time interval T_{rec}, all volatile variables and registers are copied to non-volatile memory. T_{rec} is fixed at 100 ms, allowing the comparator to recover above the internal hysteresis when there is sufficient energy input, while preventing the capacitor voltage from dropping too low to checkpoint the application state using the remaining energy.
6. After a power failure, variables and registers are restored from non-volatile memory, and execution resumes from the last executed instruction.

In addition to this automatic suspend and resume cycle, an application may synchronize execution with ambient energy availability by calling *wait_for_cap_charged()*, which halts execution until the V_{res} threshold is triggered.

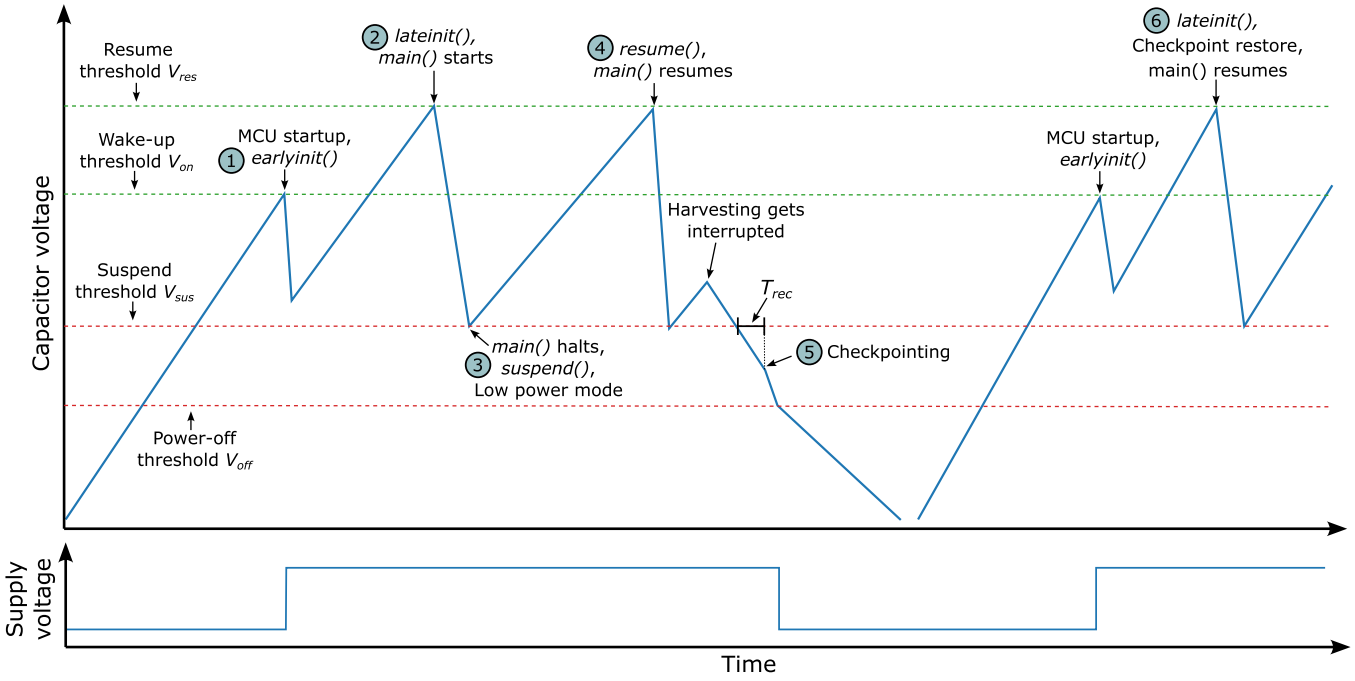

Figure 6: Illustration of Riotee's runtime operation. *The application is implemented in a main() function and several callback functions. The execution of these functions depends on the capacitor voltage and the voltage thresholds. The main() function executes continuously even after harvesting gets interrupted, for example, when a shadow temporarily covers the solar panel.*

FreeRTOS-based implementation. We implement the Riotee runtime on top of the widely used, well-audited FreeRTOS operating system. We select FreeRTOS because of its minimal runtime and code overhead and its seamless integration with our toolchain. Specifically, the runtime is implemented as three FreeRTOS tasks: a low-priority *idle* task, a medium-priority *user* task, and a high-priority *system* task. The idle task keeps the MCU in low-power sleep mode. The user task executes the application code, while the system task controls execution and handles checkpointing.

After the *lateinit()* callback, the user task is suspended, and the system task waits until the V_{res} threshold is detected. The idle task, being the only active task, keeps the system in low-power mode. Upon detecting the V_{res} threshold, the system task resumes the user task and blocks until the V_{sus} threshold is detected. The user task becomes the highest-priority task ready to execute and is swapped in by the scheduler.

When the capacitor voltage drops and remains below V_{sus} for T_{rec}, the system task becomes ready and is swapped in by the scheduler. At this point, the scheduler pushes all registers to the stack of the user task. Once the system task runs, it looks up the start and end of the user stack in the task control block and copies these data together with the global/static variables to non-volatile memory. After a power failure, these data are restored to the respective memory area, and once the task is swapped in again, execution continues exactly where it was suspended

Retained memory. Checkpointing consumes lots of energy and occurs when the energy stored in the capacitors is already low. To guarantee completion, the amount of data to be retained must be small enough to ensure the energy required for copying is less than the remaining usable energy in the capacitors. To satisfy this condition, the amount of retained RAM available to the user is limited by a constant define.

The default value of this define was determined to fit the on-board capacitance and default suspend threshold of the Riotee Module. Compilation fails with an error message if the user exceeds the memory limit. Users can either place variables outside the retained memory area using provided macros or increase the maximum amount of retained RAM. In the latter case, checkpointing may take longer and consume more energy than stored in the built-in capacitors, requiring an increase in either capacitance or the V_{sus} threshold.

5.2 Peripheral Operation

Peripheral state is generally volatile and cannot be easily saved and restored like other application state. For example, an application may enable an oscillator during startup by setting a bit in a write-only register. The oscillator stops when power fails and does not restart when power returns, although the application state is restored. This inconsistency can be critical if the application relies on the oscillator for operations such as transmitting a packet. To enable the safe use of peripherals with Riotee, we design an effective peripheral management scheme based on the three key mechanisms described below. Using this scheme, we implement drivers for the radio, ADC, RTC, GPIOs, SPI, and I2C.

Encapsulation. All configuration required for peripheral operation occurs in the *earlyinit()* or *lateinit()* callbacks (see Table 4)

and/or within a critical section. This ensures that the application is never interrupted by a power failure *during* peripheral configuration, which could lead to unsafe or incomplete configuration when the application resumes from a checkpoint. For all our implemented drivers and protocols, the configuration takes only a few microseconds, and the critical sections do not significantly delay system suspension.

Notification. After configuring an asynchronous peripheral and exiting the critical section, the application task blocks, waiting for a FreeRTOS direct-to-task notification from the peripheral's interrupt routine signaling the completion of the operation. While the application task blocks, the idle task keeps the MCU in sleep mode. In case of a power failure, the runtime sends a distinct notification value to unblock the application task after power is restored. When the application receives this notification, it can decide to retry the operation once sufficient energy becomes available.

Teardown. One critical assumption for successfully implementing a soft intermittency runtime is that power consumption must be reduced immediately upon detecting a low energy condition to avoid depleting the capacitor below the minimum operating voltage. While the CPU can be easily put into a low-power sleep mode, an active peripheral may still draw significant current, depleting the capacitor before a checkpoint can be taken. To prevent this, each driver can implement a *teardown()* function and store a pointer to it in a table. When the runtime detects a low-energy condition, it iterates through the table and executes every function with a valid pointer. Within the *teardown()* function, the driver can abort ongoing operations, put the peripheral into low-power mode, and notify the application.

5.3 Arduino Support

Installing the toolchain and working with the rich SDK API can be a barrier for programming novices wanting to start with battery-free devices. To address this, we implement the Arduino API on top of the standard SDK API and offer a convenient Arduino package that can be installed with a few mouse clicks. The Arduino ecosystem has enabled a generation of hobbyists, makers, and students to build exciting applications with a low entry barrier while maintaining the efficiency of programming close to the hardware in C/C++.

The Riotee Arduino package supports serial output, digital I/O, analog read, SPI, and I2C. The Arduino *setup()* function maps to the runtime's *lateinit()* callback, and the Arduino *loop()* function is continuously executed inside the runtime's *main()* function. Behind the scenes, most Arduino API calls translate into standard SDK API functions, incurring a negligible performance penalty. Thus, the compiled code is significantly more efficient than solutions relying on interpreted code, as demonstrated in the evaluation of a Python-based programming framework for battery-free devices [26].

When the Arduino API does not meet the developer's requirements for flexibility or performance, they can seamlessly use standard SDK API calls within their Arduino sketch. For instance, while the Arduino *analog_read()* function only allows reading a single sample from the ADC, the ADC driver in the SDK enables the acquisition of large numbers of samples with configurable sampling rates, resolutions, and ranges without CPU interaction.

5.4 Networking

The nRF52 on the Riotee Module comes with a powerful and flexible 2.4 GHz radio. As part of the Riotee SDK, we provide two network protocol implementations for reference and as a starting point for networked battery-free applications.

BLE advertising. Riotee's Bluetooth Low Energy (BLE) implementation allows sending standard-compliant undirected, non-connectable advertising packets on the three BLE advertising channels. The SDK API enables embedding arbitrary user data in the advertisements, making it easy to send application data, for example, to a smartphone.

Stella. We also provide a custom network protocol called *Stella* for bi-directional data exchange between Riotee devices and an always-on base station. The SDK includes the Stella firmware for the Riotee Module and a reference implementation for the base station. The base station, consisting of an nRF52840 dongle USB transceiver and a computer, can communicate with up to 16 connected Riotee devices and forward packets to a remote client via a RESTful API.

To communicate, a device transmits a packet to the base station containing its device ID, a packet ID, an optional acknowledgment of a previously received packet, and a payload of up to 247 bytes. After transmitting the packet, the device transitions into receive mode and listens for a response from the base station. The base station continuously listens for incoming packets. Upon receiving a packet from a device, the base station responds with an acknowledgment packet containing the device's ID, the packet ID of the received packet being acknowledged, the packet ID of the current packet, and a payload of up to 247 bytes.

6 RIOTEE IN ACTION

This section showcases the practical utility of Riotee through an example battery-free application: hot-word detection based on intermittent DNN inference.

Scenario. A battery-free device equipped with a microphone listens in low-power mode. Upon detecting a sound, the device wakes up and captures a 1-second audio sample at a 16 kHz sampling rate. After extracting features from the audio sample, the data are fed into a machine learning model designed to recognize and differentiate between two predefined keywords, "yes" and "no." The outcome of the on-device classification is then sent to a smartphone via BLE.

We build the battery-free device with a Riotee Board, Sensor Shield, Solar Shield, and Capacitor Shield. The device is placed next to a window on a sunny day, with all four solar panels on the Riotee Solar Shield activated. The Riotee SDK supports TensorFlow Lite, and we use the pre-trained *microspeech* model from TensorFlow Lite examples. The model consists of 18.8 kB of topological information and neuron weights stored in non-volatile flash. At runtime, feature extraction and inference use a shared 9464 B buffer in RAM.

While the Riotee runtime suspends and resumes feature extraction and inference, the 1-second microphone sampling must not be interrupted. To determine the required capacitance, we progressively add individual capacitors on the Capacitor Shield until sampling completes. Successful sampling is achieved when all six MLCCs on the Capacitor Shield are activated, corresponding to an effective capacitance of about 210 µF when fully charged to 4.7 V.

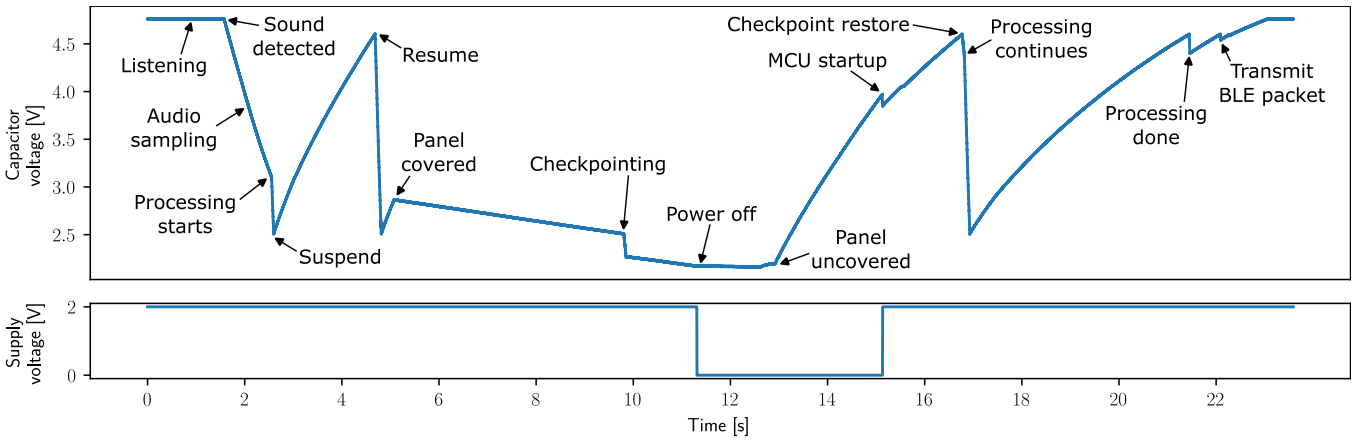

Figure 7: Real-world trace of a Riotee device performing battery-free hot-word detection using TensorFlow Lite deep neural network inference. *The solar panels are covered with a hand from time* 5 s *until time* 13 s *to showcase checkpointing.*

Example trace. Figure 7 shows a representative trace of the capacitor and supply voltages of our battery-free hot-word detection device, recorded with a Saleae Logic 8 logic analyzer. While waiting for a sound, the device consumes little energy, keeping the capacitor fully charged. Upon sound detection, sampling starts with medium power draw. Immediately after sampling, processing (i.e., feature extraction and inference) starts with higher power draw, quickly draining the remaining energy on the capacitor before the runtime suspends execution until the capacitor voltage recovers and execution continues. After covering the solar panels with a hand, the capacitor discharges slowly to around 2.5 V. At this point, a 45 kB checkpoint, including the 32 kB audio sample, the 9464 B processing buffer, and the application stack, is stored in non-volatile memory. Shortly after, the buck regulator switches off the supply voltage. Upon uncovering the solar panels, the capacitor quickly charges up, the supply voltage is turned on, and the MCU starts up. Once fully charged, the checkpoint is restored, and processing continues through another suspend-resume cycle. When processing is done, the application waits until the capacitor is fully charged and sends the classification result in a BLE packet.

Performance. The battery-free device reliably distinguishes between "yes," "no," and silence/noise. Preprocessing and feature extraction take 251 ms, while the actual inference takes 41.2 ms. Under the given light conditions, a complete cycle from sound detection to result transmission takes 5.8 s.

Summary. The Riotee hardware and software ecosystem facilitates the development of complex battery-free applications involving energy harvesting, sensing, machine learning, and wireless communication. The adaptable hardware simplifies the creation of custom prototypes without soldering, while the efficient runtime manages intermittent execution and checkpointing seamlessly, alleviating the burden on application developers. The code of this battery-free application is available as an example in the Riotee SDK.

7 USER STUDY

We conducted two user studies using questionnaires among real Riotee users. All participants of these user studies provided informed consent prior to their involvement, and their anonymity was protected throughout the research process. Our results show that both customers and students rate Riotee's usefulness and usability highly.

7.1 Customers

Riotee has been commercially available for one year and has been sold to 27 customers across eight countries in Europe, North America, and Australia. A total of 172 units have been sold across the six products combined, providing a unique opportunity to collect feedback from real users. Since our distributor does not disclose customer data, we contacted all six users for whom we had contact information because they had previously contacted us through email or GitHub. We asked participants to complete a questionnaire with ten standardized questions from the system usability scale (SUS) [6] and four custom open-ended questions. We received anonymized feedback from five respondents and summarize the results below (see Appendix A for the questions and complete responses).

Projects. Customers use Riotee for a variety of projects, from machine learning to intermittent networking. In addition to these anticipated use cases, we also received some unexpected replies. For example, one customer is coupling Riotee with a miniature wind generator, while another plans to replace a proprietary battery-free device with Riotee to benefit from open network protocols and modern programming.

Usability. We asked participants to express their agreement with the ten statements of the SUS [6] on a five-point Likert scale ranging from *strongly disagree* to *strongly agree*. If unsure, participants marked the center of the scale. Four out of five respondents would like to use Riotee more frequently, while one is undecided. None of the respondents found Riotee unnecessarily complex. Three respondents found Riotee easy to use, while two were undecided.

Table 5: Comparison of battery-free platforms. *Flicker is outdated and SuperSensor focuses on high-performance machine learning applications.*

Platform	Clock	RAM	NVM	Cap.	Idle Current
Flicker [21]	16 MHz	2 kB	128 kB	22 µF	6.25 µA
SuperSensor [3]	100 MHz	128 kB	2 MB	80 µF	>10 µA
Riotee	64 MHz	128 kB	128 kB	66 µF	4.8 µA

Two respondents believed that most people would learn to use Riotee quickly, while one disagreed. Overall, Riotee achieved a robust system usability score of 68 % across the five participants.

7.2 Students

One of the primary goals of Riotee is to introduce students to battery-free systems. We used Riotee for a lab course accompanying a lecture offered to undergraduate students at our university during the winter term 2023/2024. The lab consists of six assignments. The first five assignments use Riotee as a general-purpose development platform in constant power supply mode, gradually introducing students to basic microcontroller concepts. The final assignment focuses on battery-free aspects and intermittent computing. After the last lab, we asked the 24 students who completed the lab to fill out a six-question questionnaire. The first four questions asked students to express their agreement with a statement on the five-point Likert scale mentioned above, while the remaining two were open-ended questions. We received anonymized feedback from 14 respondents and summarize the results below (see [19] for the complete data).

Results. 11 out of 14 (78.6 %) respondents agreed that Riotee makes it easy to follow the course, while only one disagreed. 12 out of 14 (85.7 %) respondents believed that Riotee is well-suited for introducing students to battery-free systems, with only one disagreement. 11 out of 14 (78.6 %) respondents agreed that using the same platform throughout the course made for a seamless transition from continuously powered to battery-free operation, while one disagreed. In summary, students appreciate the usefulness of Riotee for the course.

8 RELATED WORK

Riotee does not introduce fundamentally new concepts. Instead, it synthesizes a decade of research on battery-free and intermittent systems by integrating proven ideas and solutions into a well-engineered, accessible platform covering a broad range of use cases. As such, Riotee's core contribution lies in providing a solid foundation that aims to facilitate future scientific advancements and community growth, comparable to how the TelosB has enabled research and development of battery-powered sensor networks.

Platforms. Table 5 compares key performance metrics of Riotee and some of its predecessors. Flicker [21] is built on the outdated MSP430FR5989, offering significantly lower capabilities and performance compared to Riotee while drawing higher current in idle mode. SuperSensor [3], a newly developed multi-core platform tailored for machine-learning applications, requires a proprietary toolchain and substantial programming effort. It also draws considerably more power than Riotee. Neither platform has a form factor suitable for practical deployments. Consequently, instead of merely improving software, documentation, and commercializing these platforms, we design Riotee from the ground up to meet the design goals outlined in Section 3.

Runtimes. Static checkpointing runtimes use a modified compiler to insert checkpoints at loop iterations and function calls [34] or before idempotent sections [42]. Task-based runtimes [22, 29, 43] reduce overhead by only checkpointing between tasks, but require to express the application as atomic, idempotent tasks in a runtime-specific syntax.

To reduce the overhead of unnecessary checkpoints while allowing execution of arbitrary application code, reactive checkpointing [4, 24] needs an interrupt that signals an impending power failure to checkpoint volatile state and put the device into sleep mode, avoiding depletion of the remaining charge. PowerNap [10] further reduces overhead by only checkpointing when the capacitor voltage does not recover after putting the device to sleep. Riotee adopts this approach, providing low energy overhead without burdening users with new syntax and detailed energy analysis of their code.

Debugger. The Energy-interference-free Debugger [7] for MSP430-based battery-free devices allows restoring the previous energy state after debugging a system halted on a breakpoint without interfering with the system's energy-driven behavior. DIPS [12] extends this concept for ARM Cortex-M based devices, enabling debugging while emulating energy input from harvesting traces.

These adapters can be used to debug applications running on the MSP430 or the nRF52 on the Riotee Module. Instead, the Riotee Probe supports programming both MCUs on the Riotee Module using a *single* adapter and a *unified* interface. Thanks to its programmable GPIO pins and power supply, energy-aware debugging tools can be implemented easily.

Accessible programming. BFree [26] aims to simplify the development of battery-free applications for hobbyists by implementing an intermittency-safe Python interpreter. A user study confirms that BFree achieves this goal, but a performance benchmark reveals that the processing time and associated energy consumption can be three orders of magnitude higher than the corresponding C implementation. Similarly, battery-free MakeCode [27] transforms programs written in novice-friendly but inefficient JavaScript, Python, or graphical Blocks into intermittency-safe code.

In contrast, Riotee offers seamless transitioning between novice-friendly Arduino code and low-level programming via a powerful C/C++ API. The efficient resulting code makes Riotee suitable for applications with tight energy budgets.

Peripheral support. Karma [5] prevents inconsistencies between peripheral state and application logic by tracking state and calls, rolling back execution to the last consistent state after a power failure. EaseIO [44] avoids redundant code execution, idempotency bugs, and unsafe code execution in task-based runtimes on mixed-volatility MCUs by letting programmers mark and categorize I/O

operations. Samoyed [30] profiles peripheral consumption at runtime and dynamically divides long-running operations into atomic sections for safe operation with reactive checkpointing.

Riotee also encapsulates peripheral configuration in atomic sections but allows long-running operations. To avoid inconsistencies, user code execution is halted until the operation terminates or aborts due to insufficient energy. The user may then repeat the operation when sufficient energy is available.

Dynamic energy burst scaling. Capybara [8] adjusts the amount of energy available per execution cycle through switchable capacitor banks that change the capacitance between fixed voltage thresholds. Another work [20] uses a non-volatile resistor network to control the turn-on threshold of the power supply dynamically. SuperSensor [3] uses a digital potentiometer for fine-grained turn-on thresholds.

Riotee adopts the approach from [3] but employs a custom-designed resistor network to provide nine different turn-on *and* turn-off thresholds while drawing an order of magnitude less current than the potentiometer-based solution.

Timekeeping. Maintaining accurate time on intermittently powered devices has received considerable attention. Common solutions leverage the decay of charge in physical components [23], such as volatile memory and capacitors, to gauge the duration of power interruptions. However, these methods often suffer from decreasing timing resolution as the length of the outage increases. For instance, despite efforts to balance resolution and range using multiple capacitor stages, solutions like CHRT [11] and HARC [13] offer only 1-second resolution after a 100-second outage. Another lightweight approach, based on modeling energy input, exhibits limitations under fluctuating energy harvesting conditions [45].

In contrast, Riotee uses an ultra-low power Ambiq AM1805 RTC with 44 µF backup capacitance to support its 14 nA current draw through power outages of over 3 min, while maintaining constant 10 ms timing resolution. This solution excels in size, cost, and accuracy, particularly for soft intermittency systems where long RTC startup times are not a concern.

9 CONCLUSIONS

Many battery-free platforms have been proposed but are rarely reused due to limited availability, insufficient software support, and lack of documentation. To address this, we have developed Riotee, an open-source, user-friendly hardware and software platform that is commercially available. Our user study shows that Riotee is highly rated for usefulness and usability by students, makers, and researchers, highlighting its potential as a valuable tool for research, education, and development of battery-free systems.

AVAILABILITY

For reproducibility, we provide the version of the open-source hardware design, software, and documentation available at the time of publication at https://doi.org/10.5281/zenodo.13902777.

ACKNOWLEDGMENTS

Thanks to Ingmar Splitt for his valuable input on the hardware design and support with the experiments. This work has been funded in part by the German Research Foundation (DFG) within the Emmy Noether project NextIoT (ZI 1635/2-1), the Saxon State Ministry for Science, Culture and Tourism, and the LOEWE initiative (Hesse, Germany) within the emergenCITY center (LOEWE/1/12/519/03/05.001-(0016)/72).

REFERENCES

[1] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the MithræUm of Circus Maximus. In *Proceedings of the 18th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[2] Saad Ahmed, Bashima Islam, Kasim Sinan Yildirim, Marco Zimmerling, Przemysław Pawełczak, Muhammad Hamad Alizai, Brandon Lucia, Luca Mottola, Jacob Sorber, and Josiah Hester. 2024. The Internet of Batteryless Things. *Communications of the ACM* 67, 3 (2024).

[3] Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. 2023. Protean: An Energy-Efficient and Heterogeneous Platform for Adaptive and Hardware-Accelerated Battery-Free Computing. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[4] Domenico Balsamo, Alex S. Weddell, Geoff V. Merrett, Bashir M. Al-Hashimi, Davide Brunelli, and Luca Benini. 2015. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters* 7, 1 (2015).

[5] Adriano Branco, Luca Mottola, Muhammad Hamad Alizai, and Junaid Haroon Siddiqui. 2019. Intermittent Asynchronous Peripheral Operations. In *Proceedings of the 17th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[6] John Brooke. 1996. *SUS – a quick and dirty usability scale.* 189–194.

[7] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson P. Sample. 2016. An Energy-interference-free Hardware-Software Debugger for Intermittent Energy-harvesting Systems. *ACM SIGARCH Computer Architecture News* 44, 2 (2016).

[8] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proceedings of the 23rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[9] Tuan Dang, Trung Tran, Khang Nguyen, Tien Pham, Nhat Pham, Tam Vu, and Phuc Nguyen. 2022. IoTree: a battery-free wearable system with biocompatible sensors for continuous tree health monitoring. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom)*.

[10] Tim Daulby, Anand Savanth, Geoff V. Merrett, and Alex S. Weddell. 2021. Improving the Forward Progress of Transient Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 3 (2021).

[11] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[12] Jasper de Winkel, Tom Hoefnagel, Boris Blokland, and Przemysław Pawełczak. 2023. DIPS: Debug Intermittently-Powered Systems Like Any Embedded System. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[13] Vishal Deep, Vishak Narayanan, Mathew Wymore, Daji Qiao, and Henry Duwe. 2020. HARC: A Heterogeneous Array of Redundant Persistent Clocks for Batteryless, Intermittently-Powered Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

[14] Vishal Deep, Mathew L. Wymore, Alexis A. Aurandt, Vishak Narayanan, Shen Fu, Henry Duwe, and Daji Qiao. 2021. Experimental Study of Lifecycle Management Protocols for Batteryless Intermittent Communication. In *Proceedings of the 18th IEEE International Conference on Mobile Ad Hoc and Smart Systems (MASS)*.

[15] Everactive Eversensor. https://support.everactive.com/hc/en-us/articles/9501962433303-Evernet-2-0-Basics. Accessed: 2024-04-13.

[16] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh Gupta. 2018. Pible: battery-free mote for perpetual indoor BLE applications. In *Proceedings of the 5th ACM Conference on Systems for Built Environments (BuildSys)*.

[17] Kai Geissdoerfer and Marco Zimmerling. 2021. Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[18] Kai Geissdoerfer and Marco Zimmerling. 2022. Learning to Communicate Effectively Between Battery-free Devices. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[19] Kai Geissdoerfer and Marco Zimmerling. 2024. *Riotee: An Open-source Hardware and Software Platform for the Battery-free Internet of Things.* Technical Report. https://nes-lab.org/wordpress/wp-content/uploads/2024/10/geissdoerfer24riotee_tr.pdf

[20] Andres Gomez, Lukas Sigrist, Michele Magno, Luca Benini, and Lothar Thiele. 2016. Dynamic energy burst scaling for transiently powered systems. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE).*

[21] 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).*

[22] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys).*

[23] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. 2016. Persistent Clocks for Batteryless Sensing Devices. *ACM Transactions on Embedded Computing Systems* 15, 4 (2016).

[24] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers. In *Proceedings of the 27th ACM International Conference on VLSI Design and 13th ACM International Conference on Embedded Systems.*

[25] Sara Khalifa, Mahbub Hassan, Aruna Seneviratne, and Sajal K. Das. 2015. Energy-Harvesting Wearables for Activity-Aware Services. *IEEE Internet Computing* 19, 5 (2015).

[26] Vito Kortbeek, Abu Bakar, Stefany Cruz, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. BFree: Enabling Battery-free Sensor Prototyping with Python. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020).

[27] Christopher Kraemer, Amy Guo, Saad Ahmed, and Josiah Hester. 2022. Battery-free MakeCode: Accessible Programming for Intermittent Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6 (2022).

[28] Gaosheng Liu and Lin Wang. 2021. Self-Sustainable Cyber-Physical Systems with Collaborative Intermittent Computing. In *Proceedings of the 12th ACM International Conference on Future Energy Systems.*

[29] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1 (2017).

[30] Kiwan Maeng and Brandon Lucia. 2019. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation.*

[31] Andrea Maioli and Luca Mottola. 2021. ALFRED: Virtual Memory for Intermittent Computing. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys).*

[32] EnOcean STM 400J Product Page. https://www.enocean.com/en/product/stm-400j/. Accessed: 2024-04-13.

[33] Wiliot Pixel Product Page. https://www.wiliot.com/product/iot-pixels. Accessed: 2024-04-13.

[34] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: system support for long-running computation on RFID-scale devices. In *Proceedings of the 16th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).*

[35] Alanson Sample, Daniel Yeager, Pauline Powledge, Alexander Mamishev, and Joshua Smith. 2008. Design of an RFID-Based Battery-Free Programmable Sensing Platform. *IEEE Transactions on Instrumentation and Measurement* 57, 11 (2008).

[36] Muhammad Moid Sandhu, Kai Geissdoerfer, Sara Khalifa, Raja Jurdak, Marius Portmann, and Brano Kusy. 2020. Towards Energy Positive Sensing using Kinetic Energy Harvesters. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom).*

[37] Muhammad Moid Sandhu, Sara Khalifa, Kai Geissdoerfer, Raja Jurdak, and Marius Portmann. 2021. SolAR: Energy positive human activity recognition using solar cells. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom).*

[38] Lukas Sigrist, Naomi Stricker, Dominic Bernath, Jan Beutel, and Lothar Thiele. 2020. Thermoelectric Energy Harvesting From Gradients in the Earth Surface. *IEEE Transactions on Industrial Electronics* 67, 11 (2020).

[39] Philip Sparks. 2017. *The route to a trillion devices: The outlook for IoT investment to 2035.* White Paper. ARM Limited.

[40] Arduino Store. https://store-usa.arduino.cc/. Accessed: 2024-04-18.

[41] Vamsi Talla, Bryce Kellogg, Benjamin Ransford, Saman Naderiparizi, Shyamnath Gollakota, and Joshua R. Smith. 2015. Powering the next billion devices with wi-fi. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNext).*

[42] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent Computation without Hardware Support or Programmer Intervention. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI).*

[43] Kasim Sinan Yildirim, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. InK: Reactive Kernel for Tiny Batteryless Sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys).*

[44] Eren Yildiz, Saad Ahmed, Bashima Islam, Josiah Hester, and Kasim Sinan Yildirim. 2023. Efficient and Safe I/O Operations for Intermittent Systems. In *Proceedings of the 18th European Conference on Computer Systems (EuroSys).*

[45] Eren Yildiz and Kasim Sinan Yildirim. 2021. Persistent Timekeeping Using Harvested Power Measurements. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys).*