Shepherd: A Portable Testbed for the Batteryless IoT

Kai Geissdoerfer
Networked Embedded Systems Lab
TU Dresden, Germany
kai.geissdoerfer@tu-dresden.de

Mikołaj Chwalisz
Telecommunication Networks Group
TU Berlin, Germany
chwalisz@tkn.tu-berlin.de

Marco Zimmerling
Networked Embedded Systems Lab
TU Dresden, Germany
marco.zimmerling@tu-dresden.de

ABSTRACT

Collaboration of batteryless nodes is essential to their success in replacing traditional battery-based systems. Energy-harvesting sensor nodes experience spatio-temporal fluctuations of energy availability. These fluctuations become especially critical when sensor nodes do not have sufficient energy storage to compensate for them. Understanding the challenges and opportunities of operating groups of batteryless sensor nodes requires to record and reproduce spatio-temporal characteristics of real energy environments. We thus present Shepherd, a testbed for the batteryless IoT. Shepherd allows to record synchronized energy traces with a resolution of 3 µA and 50 µV at a rate of 100 kHz, and to faithfully replay these traces to any number of sensor nodes to study their behavior. We release Shepherd as an open-source tool for the community, facilitating research into time synchronization, wireless networking, and other distributed algorithms for batteryless systems.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Sensor networks**; • **Hardware** → **Analysis and design of emerging devices and systems**;

KEYWORDS

Energy harvesting, Batteryless, Intermittent power, Transient computing, Intermittent networking, Testbed, Recording, Emulation

ACM Reference Format:
Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. 2019. Shepherd: A Portable Testbed for the Batteryless IoT. In *The 17th ACM Conference on Embedded Networked Sensor Systems (SenSys '19), November 10–13, 2019, New York, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3356250.3360042

1 INTRODUCTION

As the Internet of Things (IoT) grows to trillions of devices [26], sustainability and reliability of this computing infrastructure become matters of utmost importance. One possible path to sustainability is the adoption of *batteryless* devices that buffer harvested energy in a capacitor, and execute when there is energy available in the capacitor. Batteryless devices promise to overcome the drawbacks of (rechargeable) batteries, such as bulkiness, wear-out, toxicity, uncertain remaining charge, etc. The limited energy capacity of capacitors, however, requires intermittently executing the software, which may harm reliability [13] despite checkpointing techniques [2, 18], platform support [6, 11], and dedicated programming models [5].

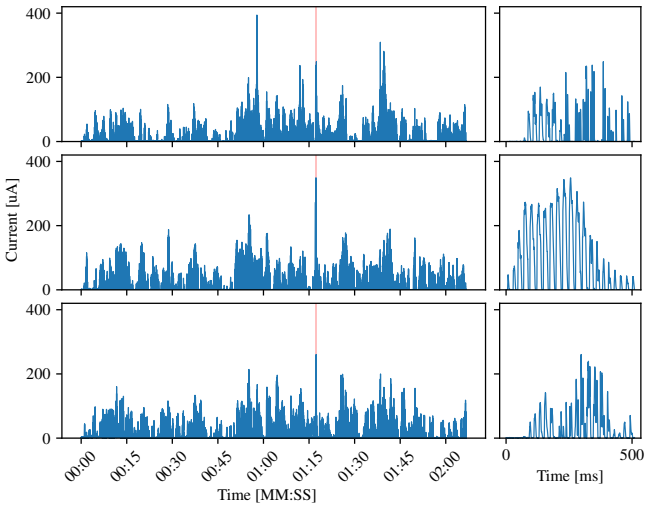


Figure 1: Synchronized traces of kinetic harvesting current, recorded with three Shepherd nodes mounted at different locations of a car. Shepherd can replay such harvesting current and voltage traces to distributed batteryless devices.

Focusing on *multiple* rather than individual batteryless devices not only enables exciting new applications (*e.g.*, swarms of nanosatellites [20]), but may also offer a new perspective on the reliability issue. Fault tolerance in conventional (*i.e.*, continuously-powered) distributed systems often relies on exploiting redundancy [22]: If the level of fault tolerance provided by a single server is unacceptable, then multiple servers executing replicas must be used. From this experience, we may ask research questions like the following: Is it possible to operate a distributed collection of batteryless devices so that the group is more reliable than each device alone? How would efficient and reliable programming models, wireless communication protocols, and runtimes for collections of batteryless devices look like? Can the cooperation of multiple distributed batteryless devices bring about benefits in terms of overall efficiency and effectiveness, similar to cooperation in multi-agent systems [21]?

Motivating example. To illustrate a possible way to investigate those questions, Fig. 1 shows real traces of harvesting current we synchronously recorded at three devices with piezo-electric elements. The devices are mounted at different locations of a car, and harvest energy from the car's vibrations as it drives through a suburban area. This setting is akin to, for example, devices mounted on a large machine in a factory (*e.g.*, for predictive maintenance).

SenSys '19, November 10–13, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6950-3/19/11…$15.00
https://doi.org/10.1145/3356250.3360042

Considering any of the traces individually, we observe that the instantaneous energy availability of a device varies significantly and unpredictably over time. This is indeed one of the key challenges in the design of useful, reliable batteryless applications [12, 17]: In the absence of a large energy storage (*e.g.*, a battery), fine-grained variations in energy availability cannot be abstracted away. Rather, the matter of *when* energy is available may be critical to the correct functioning of a batteryless system, which is comparable to the crucial role of time for the correctness of a cyber-physical system [7].

Looking at all three plots to the left together, we instead notice a similar macroscopic shape of the traces. The spatial proximity of the devices and systematic properties of the energy-harvesting environment result in a positive correlation between the time-varying energy availabilities: If one device has energy, then the other two tend to have energy as well. This also holds in other environments, such as indoor solar energy harvesting (see Sec. 7). Although there are non-negligible differences in energy availability across the devices, as visible in the zoomed-in plots to the right, exploring ways to exploit such correlations (*e.g.*, for synchronizing and networking batteryless devices) may eventually provide answers to the challenging research questions posed above and elsewhere [3, 12, 17].

Problem. Unfortunately, the research community lacks a tool that enables such scientific endeavors. An appropiate tool needs to synchronously *record* the rapidly changing energy conditions at different points in space. Even having such traces, it is hard to accurately model and predict the performance and behavior of a real batteryless system because of the complex behavior of circuits exposed to an intermittent power supply. To develop and compare novel designs, it is thus necessary to experiment under the constraints of time-varying energy availability by faithfully *reproducing* energy environments from recorded traces or spatio-temporal models.

Recording and replaying harvested energy is hard, and few solutions exist for individual devices. For example, using a source measure unit (SMU) one can profile and emulate a single harvester. However, such equipment is expensive (*i.e.*, thousands of USD per unit), while the sampling speed may not be sufficient to capture rapidly changing energy availability. To address this problem, Ekho [10] uses custom-designed, affordable hardware to record and emulate an energy source with limited accuracy and resolution.

Testbeds support synchronous recording of current draw [15] or energy consumption [23] at multiple distributed devices. Although this is one piece of the puzzle, it is not possible to profile the complex behavior of an energy-harvesting system with existing testbeds by reproducing an energy environment. The FlockLab testbed offers the possibility to vary the supply voltage, emulating a discharging battery [15]. However, as described in Sec. 2, an energy harvester has a characteristic IV curve that determines the current flowing at a particular voltage. Emulating this behavior requires an inherently different approach than what is found in existing testbeds.

Contribution. This paper presents Shepherd, a portable testbed for the batteryless IoT that fills this gap. Shepherd's main novelty is the combined capability of accurately recording and replaying high-resolution voltage and current traces synchronously and at high rates across spatially distributed batteryless devices. With this, Shepherd provides unprecedented visibility into energy environments across time and space (see Fig. 1), and faithfully reproduces those real-world conditions for the systematic development and evaluation of distributed batteryless applications and services.

Shepherd is a complement of hardware and software. Its modular hardware architecture rests upon a powerful observer platform with a custom-designed analog frontend, deep local storage, and real-time processing capabilities. Different harvesting sources, energy buffers, and sensor nodes can be attached to an observer using well-defined interfaces. Shepherd's software architecture tackles the challenges of tight synchronization among observers that may be kilometers apart (*e.g.*, batteryless LPWAN), and by providing reliable, high-throughput data transfer subject to timing constraints.

Beyond record and replay, Shepherd's harvesting traces may be analyzed offline or fed into simulators. Conversely, Shepherd can also replay traces generated in software or recorded with other tools, such as Ekho [10], RocketLogger [24], or a SMU. Shepherd is affordable (about 200 USD per observer) and portable (supporting mobile outdoor scenarios) as it does not rely on heavy infrastructure, yet it offers all amenities of existing testbeds, including GPIO tracing, serial logging, and remote programming.

To summarize our main contributions:

- We identify a workflow for the development and evaluation of solutions for distributed batteryless devices, and derive the key requirements of a testbed that supports this workflow.
- We design and build Shepherd, the first testbed that meets those requirements. We open-source Shepherd's hardware/software stack together with extensive documentation and tools that aid users during installation and experimentation.[1]
- We demonstrate Shepherd's utility and capabilities using a real-world distributed batteryless application scenario.
- We evaluate Shepherd's performance and show, for example, that it records traces with a resolution of 3 µA/50 µV at a rate of 100 kHz, it replays traces with a mean error below 0.1 %, while ensuring a synchronization accuracy of 2.4 µs or better.

2 BACKGROUND

This section provides some background on the device and energy-harvesting architecture of batteryless systems.

2.1 Batteryless Device Architecture

In its simplest form, a batteryless device consists of a harvester and a sensor node. The *harvester* is a transducer that converts some form of ambient energy, such as solar radiation or movement, into electrical energy. The *sensor node* operates from the energy extracted by the harvester, and typically includes a microcontroller unit (MCU), sensors, volatile and non-volatile memory, and a wireless radio.

In this configuration, the harvester must deliver the minimum voltage and power required to operate the sensor node (*e.g.*, about 1.8 V to operate the MCU and about 10 mW to send a packet over an active IEEE 802.15.4 radio). Adding an *energy buffer*, usually a fixed-size capacitor, allows to decouple the node operation from the instantaneous energy availability. While the node is inactive, energy accumulates in the buffer. When the energy in the buffer reaches a threshold, the node operates and consumes the buffered energy. This way, the node can operate although the voltage or power from the harvester is momentarily insufficient.

[1] https://shepherd.nes-lab.org

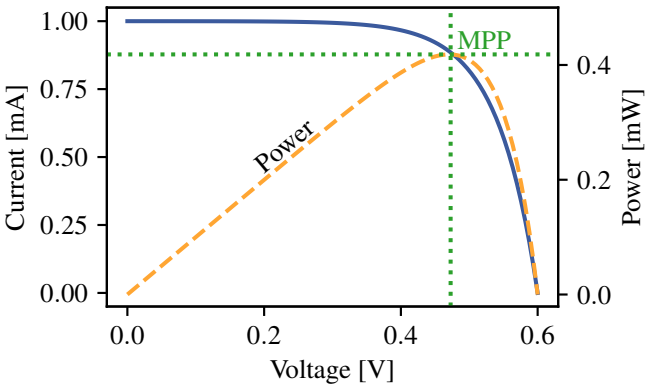


Figure 2: Characteristic IV curve of a solar cell. The voltage (V) on its output depends on the current (I) that is drawn and vice versa. The maximum power point (MPP) is the operating point where the extracted power reaches its maximum.

2.2 Characteristics of Harvesting Source

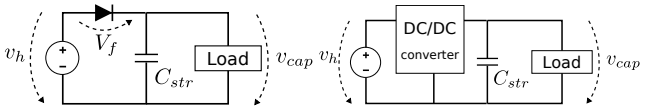
An ideal voltage source provides unlimited current. Instead, a real harvesting source has a distinctive IV characteristic: The voltage (V) on its output depends on the current (I) that is drawn and vice versa. The voltage together with the corresponding current determine the harvester's operating point. Fig. 2 shows the characteristic *IV curve* (solid line) of a solar cell, illustrating that the extracted *power* (dashed line) crucially depends on the operating point. The operating point where the extracted power reaches its maximum is called *maximum power point (MPP)*. This has important implications for the extraction of energy from the harvesting source.

2.3 Energy-harvesting Architecture

There exist two fundamental approaches to extract energy from a harvester, converter-less and converter-based, each with their own strengths that can be exploited for different batteryless applications.

Converter-less. As shown in Fig. 3a, a *converter-less* architecture consists of the energy-harvesting source, a diode, a capacitor, and the load. The operating point of the harvester depends on the state of charge of the capacitor as the harvesting voltage v_h is the sum of the capacitor voltage v_{cap} and the diode drop V_f. This prevents effective energy extraction as the harvester's operating point can be far off its maximum power point. Moreover, the harvester must be carefully selected and dimensioned to ensure minimum voltage and power conditions. Otherwise, the sensor node may never be able to operate, because the harvester delivers no current at a high enough voltage. For example, a typical solar cell delivers current only up to a voltage of about 600 mV, whereas a typical MCU needs at least 1.8 V to operate. Nevertheless, a converter-less approach requires only a minimum number of components and is thus highly cost-efficient, robust, and allows for extremely small form factors.

Converter-based. As shown in Fig. 3b, a *converter-based* architecture uses a DC/DC converter in order to operate the harvester at an operating point different from the load: The harvesting voltage v_h can be set independently of the capacitor voltage v_{cap}. Thus, with knowledge about the IV characteristic of the source, the system can optimize power yield by dynamically adapting the operating



(a) Converter-less. **(b) Converter-based.**

Figure 3: Two fundamental energy-harvesting architectures.

point of the harvester, which is known as *maximum power point tracking*. This allows to efficiently harvest energy from a variety of low-voltage sources independent of the state of charge of the capacitor. Drawbacks of this approach include increased complexity, size, and cost because adding a DC/DC converter involves adding an integrated circuit and a handful of passive components.

3 REQUIREMENTS AND OVERVIEW

Shepherd is the first testbed for distributed batteryless systems. This section outlines the key requirements for such a testbed and provides an overview of how Shepherd addresses them.

3.1 A Typical Workflow

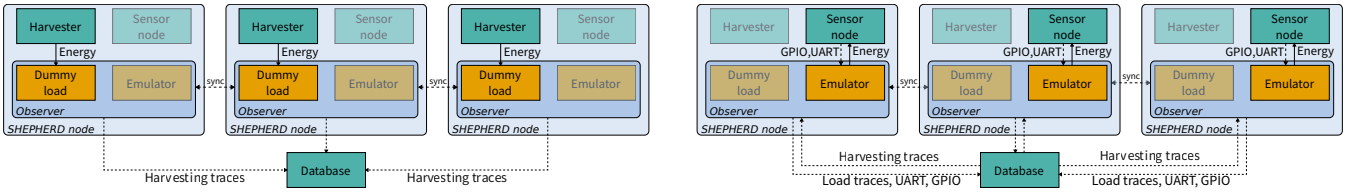
We envision the following typical workflow for the development and evaluation of distributed batteryless systems. A number of testbed nodes equipped with the desired energy-harvesting technology are deployed in the energy environment of interest. The testbed records the harvested energy at each node for a user-defined period of time. The user retrieves the data and analyzes them to gain an understanding of the characteristics of the recorded energy environment. With the help of the testbed, the user can then develop, test, and validate ideas involving (one or) multiple batteryless devices by repeatedly replaying the recorded energy traces to the device(s). This enables repeatable, experiment-driven research into open problems such as time synchronization, wireless networking, or distributed sensing and actuation using collections of batteryless devices [3, 11, 17]. A solution can then be validated by deploying it to the testbed in the target RF environment and analyzing the behavior and performance of the system. Replaying the same energy conditions allows to rigorously compare different solutions.

3.2 Key Requirements

From this envisioned workflow we derive the following key requirements for a useful testbed for distributed batteryless systems.

High accuracy and resolution. Harvested energy must be recorded and replayed accurately and precisely to be able to draw meaningful conclusions. Typical voltages of harvesting transducers like solar cells or piezo-electric elements range from hundreds of mV to a few V. Harvesting currents have an even wider range, typically from µA to tens of mA. Voltage and current draw of a common sensor node are in a similar range. From these figures, we can derive our first two requirements: Current should be recorded with a resolution of 1 µA within a range up to 50 mA, and voltage should be recorded with a resolution of 1 mV within a range up to 3 V.

High sampling rate. Current and voltage need to be sampled at a rate high enough to capture the fine-grained characteristics of a harvesting source. For example, we found that a solar cell changes its voltage within tens of µs in response to a light being switched on. A sampling rate of at least 100 kHz is needed to capture such rapid changes in energy-harvesting conditions.



(a) Recording. (b) Emulation.

Figure 4: Essential components of Shepherd and their interactions during Shepherd's two main modes of operation.

Time synchronization. To record and replay the energy environment and behavior of a batteryless network, the testbed nodes need to be tightly time-synchronized. In particular, the synchronization error must be significantly less than the sampling interval to unambiguously map samples from different nodes on a common timeline. We therefore target a synchronization accuracy of 1 µs.

Debugging facilities. Next to remote programming, the testbed should offer state-of-the-art debugging facilities such as synchronized tracing of GPIO pins and serial logging (*e.g.*, via `printfs`).

Portability, affordability, and customizability. A testbed for distributed batteryless devices must be exposed to different energy environments with unique characteristics and requirements. Thus, unlike conventional testbeds that are installed at a fixed location, a portable testbed is needed that users can afford to build and easily set up in various locations, which poses strict limitations on infrastructure and costs. This includes the ability to support new harvesting modalities and node platforms with minimal effort.

3.3 Shepherd Overview

To meet the above requirements, Shepherd consists a network of *Shepherd nodes* that are synchronized and operate in two modes:

- During *recording* (see Fig. 4a), energy flows from the harvester to a dummy load, which is part of a powerful *observer* platform. The observer measures current and voltage, and timestamps the data with respect to the testbed-wide timeline. The timestamped data are first buffered locally on the observers, and then shipped to a remote database.
- During *emulation* (see Fig. 4b), data are sent from the database to the Shepherd nodes from where they are fed into a harvesting emulator that outputs the corresponding voltage and current to an attached node. The data can be from previous recordings with Shepherd or some other tool, or generated using, for example, a spatio-temporal model of an energy environment. While replaying, the observer also monitors the sensor node's power draw, samples the GPIO pins, and records any serial messages from the node.

In the following two sections, we detail Shepherd's hardware and software architecture. Sec. 6 describes how users interact with Shepherd. Sec. 7 illustrates the capabilities and utility of Shepherd through a real-world use case, while Sec. 8 systematically evaluates the performance characteristics of Shepherd's hardware/software stack using a series of controlled experiments.

4 SHEPHERD HARDWARE

As shown in Figs. 5 and 6, a *Shepherd node* consists of an *observer* and three *capelets*, and measures 90 mm×55 mm×40 mm. The *harvesting capelet* hosts the harvesting transducer and all components

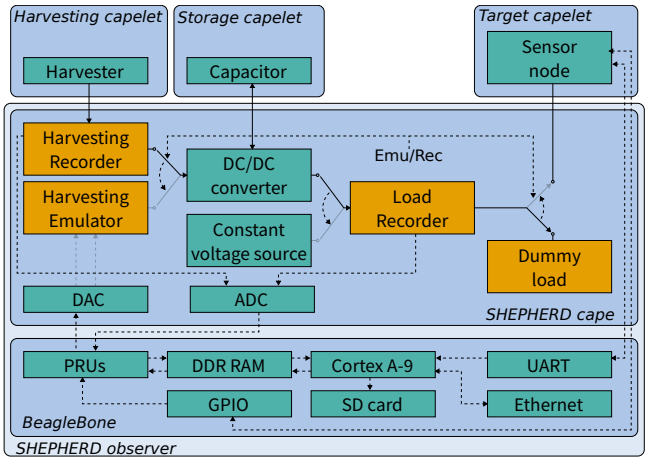


Figure 5: A Shepherd node consists of a BeagleBone SBC, the custom Shepherd cape, and three attached capelets.

required to operate it; the *storage capelet* hosts the energy buffer, usually a capacitor with the desired size; and the *target capelet* hosts the sensor node. The three capelets are connected to the observer through well-defined interfaces, which makes for a modular design that users can easily customize depending on their needs in terms of harvesting modality, energy storage, and sensor node platform.

The observer includes a custom-designed analog frontend, the *Shepherd cape*, and a *BeagleBone* single-board computer (SBC). Multiple observers connect via their BeagleBones' Ethernet ports with each other and to a host that stores the data and runs a tool we provide for orchestrating a collection of distributed Shepherd nodes. The Shepherd cape hosts all components and circuitry required for the recording and replaying of energy-harvesting traces.

4.1 BeagleBone

The BeagleBone is responsible for time synchronization, hardware interfacing, and data processing. We base our design on this platform as it is a mature single-board computer with superb software support and a living community. Two features make the BeagleBone particularly suitable for our needs compared to similar platforms.

First, the Ethernet controller of the BeagleBone's system-on-chip supports timestamping of Ethernet packets. As described in Sec. 5.2, we use this feature to tightly time-synchronize a collection of distributed Shepherd nodes using Precision Time Protocol (PTP).

Second, the Programmable Real-time Unit Sub-System (PRUSS) of the BeagleBone's system-on-chip includes two deterministic RISC cores, the Programmable Real-time Units (PRUs), which we dedicate to time-critical tasks, such as interacting with the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC)

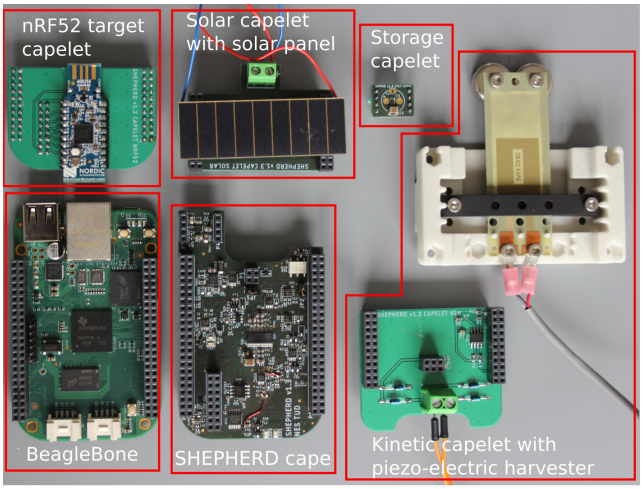


Figure 6: A Shepherd node is 90 mm×55 mm×40 mm in size, including the harvesting, storage, and target capelets.

on the analog frontend, as illustrated in Fig. 5. Less critical tasks, such as storage and networking, are instead handled by the high-throughput ARM Cortex-A9. The PRUs and the ARM core can exchange data and control signals through shared memory and the system bus, which we use to implement a bidirectional communication protocol (see Sec. 5.1). The PRUs also have direct, low-latency access to some peripherals including the GPIOs. This is essential to achieve a GPIO sampling latency in the low µs range.

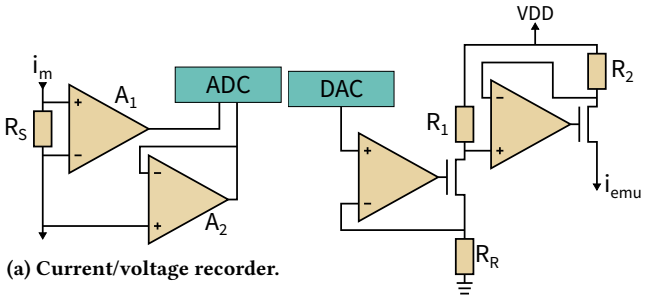
Using expansion headers, the analog fronted is stacked onto the BeagleBone as a cape (hence the name Shepherd cape), which in turn serves as the base board for the capelets discussed next.

4.2 Capelets

Shepherd supports different harvesting sources, energy buffers, and node platforms. To this end, we physically separate the Shepherd cape (*i.e.*, the analog frontend) from these components by introducing harvesting, storage, and target capelets, making it easy to exchange them (modularity) without affecting the behavior of the Shepherd cape or the BeagleBone (composability).

Target capelets. Target capelets are similar to adapter boards in Flocklab [15]: They provide a hardware interface that allows connecting a specific sensor node to Shepherd. It essentially connects the node to the capacitor-buffered output voltage of Shepherd's DC/DC converter. The well-defined connector includes all signals required to control and monitor a wide spectrum of sensor nodes through Shepherd. For example, Serial Wire Debug (SWD) signals allow remote programming and debugging of many modern ARM-based nodes. Universal Asynchronous Receiver/Transmitter (UART) pass-through allows programming of targets with a serial bootloader (*e.g.*, TelosB) in addition to serial logging and injection of commands. Finally, four General Purpose Input Output (GPIO) lines facilitate high-resolution monitoring of logical program states.

We provide a *nRF52840 capelet* as reference implementation of a target capelet. It interfaces the increasingly popular off-the-shelf nRF52840 dongle from Nordic Semiconductor with Shepherd. This target capelet merely serves as an adapter to connect the pins of the dongle to the 16-pin target connector on the Shepherd cape.



(a) Current/voltage recorder. **(b) Current emulator.**

Figure 7: High-level schematics of recorder and emulator.

Harvesting capelets. Harvesting capelets provide Shepherd nodes with the energy-harvesting source of choice, such as a solar panel or piezo-electric element. Harvesting capelets may have a small flash memory that can be used to store an ID and specific parameters for the mounted type of harvester. We provide two harvesting capelets.

The *solar capelet* allows to mount a solar panel to a screw terminal, directly connecting this DC source to Shepherd. Using a voltage divider, the operating point of the maximum power point tracker of the DC/DC converter (see Sec. 4.3) is set to 80 % of the open-circuit voltage, which is typical for solar energy harvesting.

The *kinetic energy harvesting (KEH) capelet* is a more complex example of a harvesting capelet. The AC voltage from the piezo-electric element connected to the screw terminal is rectified with a full-bridge rectifier before connecting it to Shepherd's DC/DC converter. In contrast to solar energy harvesting, the optimal operating voltage of a piezo-electric harvester is often not derived from the open-circuit voltage, but set to a fixed value. For this reason, the KEH capelet has a flash memory that is used to inform Shepherd about the type of capelet and the desired operating voltage.

Storage capelets. Storage capelets host a capacitor as energy buffer that is directly connected to the DC/DC converter of the Shepherd cape. We provide a storage capelet with a 150 µF ceramic capacitor.

4.3 Analog Frontend: The Shepherd Cape

During recording the observer measures the power extracted from the harvester, and during emulation it replays (this or some other) power trace to the sensor node while measuring the node's consumption. The custom-designed analog frontend hosts the circuitry and components required to support these two modes of operation, node programming and debugging, as well as ensuring a defined initial filling level of the energy buffer before an experiment starts. Our design also caters for use cases where Shepherd serves as a portable testbed for non-harvesting sensor nodes.

DC/DC converter. Existing work on recording and emulating harvesting traces considers converter-less systems [10]. Because the harvested energy depends on the load voltage in this case, as explained in Sec. 2.3, the whole IV characteristic has to be sampled for every point in time by rapidly altering the load voltage.

With Shepherd's converter-based approach, the harvester can be operated at a defined point, independent of load behavior and capacitor voltage. Therefore, it is sufficient to sample current and voltage at the defined operating point for any single point in time, allowing orders of magnitude faster sampling. During replay the DC/DC converter is then forced into the very same operating point.

We use TI's BQ25504, an industry-standard boost converter that charges a storage element from a harvesting source with a voltage of 100 mV or higher. It provides over- and under-voltage protection, and a *storage-voltage-in-operating-range signal* that indicates whether the attached storage element is in a configurable operating range. This signal changes to high when the voltage on the storage element rises above the upper threshold and changes to low when the voltage falls below the lower threshold. An attached load, such as a sensor node, can use this signal to schedule its activity based on the amount of energy in the buffer. We also use it to control our dummy load, as explained in the following paragraph.

Dummy load. The DC/DC converter operates the harvester independent of the state of charge of the capacitor, but this only holds within certain limits. For example, the BQ25504 can only charge a storage element up to 5.5 V. Above this threshold it switches off the regulator so no more current is flowing from the harvester. Similarly, once the voltage on the capacitor falls below 1.8 V, the BQ25504 operates in cold-start mode, where a charge pump operates the harvester at a fixed operating voltage with reduced efficiency. Thus, it is important to keep the voltage on the storage element within these limits to guarantee independence of the harvester's operating point and the capacitor voltage during recording. To this end, we use a *dummy load* that consumes the harvested energy from the capacitor when the voltage reaches a defined threshold (currently 2.8 V). We make use of the storage-voltage-in-operating-range signal of the BQ25504 to switch on an electric load consisting of two parallel light-emitting diodes (LEDs). This load remains on until the capacitor voltage falls below a lower threshold (currently 2.3 V).

Harvesting recorder. The harvesting recorder measures voltage and current flowing from the harvester to the DC/DC converter.

Accurately measuring current at high rates over a wide dynamic range is challenging. The challenge lies in converting the range of current to a range of voltage that matches the input range of the ADC without negatively affecting the signal source and without introducing excessive noise into the signal chain.

We use a shunt am-meter (R_S and A_1 in Fig. 7a) that measures the voltage drop as current flows through a resistor. Using a large resistor results in a large voltage drop, effectively reducing the voltage seen by the load (burden voltage). Conversely, a small resistor produces a small voltage drop, requiring significant amplification before the ADC. Since the collective noise at the input stage of the amplifier is also amplified this results in a lower signal-to-noise ratio (SNR). The noise is usually broadband and can be reduced by downsampling; however, this reduces the effective sampling rate.

We use a small shunt resistor of $R_S = 1\,\Omega$ to keep the burden voltage low. The voltage drop is amplified with an ultra-low noise instrumentation amplifier (A_1), keeping noise levels to a minimum. The amplified voltage is sampled using a high-resolution successive approximation register ADC that has a second-order low-pass filter on the input to band-limit the noise entering the ADC.

Measuring voltage is relatively easy as typical harvesting voltages are well within the range of commonly used ADCs. However, the input impedance of the ADC is relatively low, allowing current to flow from the harvesting source into the inputs of the ADC affecting the measurement. Therefore, we use a low-noise op-amp with very low input bias current as voltage buffer (A_2 in Fig. 7a).

Harvesting emulator. For replaying traces previously recorded or generated in software, we must independently set voltage and current at the input of the DC/DC converter. The BQ25504 can be supplied with a reference voltage to which it regulates the input. We use a high-speed, high-resolution DAC to dynamically generate this reference voltage according to the sequence of digital values in the recorded/generated trace. We set the current using the precision high-side voltage-to-current converter shown in Fig. 7b. This circuit provides a current to a ground-referenced load (R_R) proportional to the voltage on the input. This voltage is generated with a second DAC that is identical to the one used for the reference voltage. The resulting current flows through resistor R_1, causing a voltage drop. The amplifier A_2 regulates the current through R_2 such that the corresponding voltage drop equals the voltage over R_1, effectively setting i_{emu} proportional to the input voltage from the DAC.

Load recorder. To study the collective behavior of a group of harvesting nodes running a distributed application, it is essential to monitor their power draw. To this end, we provide a *load recorder* that measures the voltage on the capacitor and the instantaneous current drawn by the sensor node. The load recorder has similar requirements as the harvesting recorder and is thus almost identical to the one shown in Fig. 7a. The only difference is a slightly cheaper instrumentation amplifier with a wider common-mode range supporting current measurements at load voltages above 4 V.

Constant voltage source. We also add a *constant voltage source* that can be dynamically enabled and disabled in software at runtime. The voltage can be set to values in the range from 2.2 V to 3.3 V using a potentiometer. The constant voltage source serves three main purposes: (*i*) it supplies the sensor node with a stable voltage during programming/debugging; (*ii*) it allows to pre-charge the energy buffer to a defined initial filling level; (*iii*) it allows to use Shepherd as a portable testbed for non-harvesting sensor nodes.

EEPROM. The Shepherd cape also features a 256 kB Electrically Erasable Programmable Read-only Memory (EEPROM) to store the name of the cape, a unique ID, and hardware calibration parameters. Based on the name of the cape, the software running on the BeagleBone can automatically configure peripherals; for example, it configures the corresponding GPIO pins as input. Storing calibration data on the cape hardware also relieves the user from the task of keeping track of which cape is mounted on which BeagleBone.

5 SHEPHERD SOFTWARE

The Shepherd software stack consists of four main components: The *PRU firmware* controls the hardware on the Shepherd cape. The *kernel module* provides an interface between the PRU firmware and the user-space code, and synchronizes the PRU clock to the Linux host clock. The *user-space code* handles data storage and retrieval, and exposes a high-level user interface to manage all underlying software and hardware. Finally, the *user interface* provides a convenient way to start/stop recording and emulation from the user's machine on a collection of distributed Shepherd nodes.

5.1 Data Handling

The key challenges in Shepherd's software architecture include time synchronization and high-throughput data exchange among

user-space code, Linux kernel code, and the PRUs subject to timing constraints. Our solution uses four mechanisms: (*i*) interrupts between the ARM Cortex-A9 and the two PRUSS cores; (*ii*) the Linux *remoteproc* framework that manages resources and controls the state of the PRUs; (*iii*) the Remote Processor Messaging (RPMSG) protocol, a standardized messaging solution for communication with the PRUs; and (*iv*) shared access to the main DDR RAM.

In the following, we describe the data exchange during emulation as the more general case that involves bi-directional data exchange: harvesting data is transferred from the database to the frontend, and load recordings are sent from the frontend to the database. We use the *remoteproc* framework to allocate an area in the DDR RAM that can be accessed from Linux and from the PRUs. We divide this memory into 64 buffers that can each store 10,000 current and voltage samples. The user-space code starts by copying data from the database into memory, buffer by buffer. After writing a complete buffer, the corresponding index is sent to the PRU core that is responsible for data acquisition (PRU1) as an RPMSG. The RPMSG communication is double-buffered such that both sides can keep writing into the corresponding queues, while the other side is busy. PRU1 retrieves the buffer index from the queue and transfers the samples one by one to the DAC based on a sampling trigger generated by PRU0. After sending one sample to the DAC, it samples the ADC and overwrites the memory from which the DAC sample was read. After processing a complete buffer, PRU1 returns the buffer index to the RPMSG queue. The user-space code receives the buffer index from the queue, copies the data to the database, and fills the buffer with the next block of emulation data.

Database. We currently use the *hdf5* file format to store data locally on each Shepherd observer, using an SD card or USB flash storage. Together with the raw data, we store calibration values retrieved from the EEPROM on the Shepherd cape. This approach allows to easily share data that can be viewed and replayed independent from the hardware it was recorded with. After recording, data can be conveniently downloaded from individual nodes and merged into a single file using the command-line utilities we provide (see Sec. 6). Similarly, for emulation, the corresponding data are uploaded to each Shepherd node before the experiment starts.

5.2 Time Synchronization

Meaningful experimentation with a collection of batteryless, energy-harvesting devices requires that both the recording and the emulation of harvested energy happens synchronously on all devices under test. Similarly, recordings of load voltage and current have to be time-synchronized in order to interpret node interactions.

We use an approach where each sample is scheduled at a defined point in time. At a sampling rate of 100 kHz, samples are always taken at the wrap of full 10 µs. This has two advantages: (*i*) While synchronized, all nodes take the same number of samples on average. (*ii*) It is sufficient to timestamp the first sample in any 'block' of samples: The timestamp of all following samples in that block can be derived from this first timestamp using the sampling interval.

The goal of time synchronization is to take and replay samples at the same specified time on a number of Shepherd observers with as little jitter as possible. In essence, this means that the PRUSS cores on the observers, which handle the direct interaction with the hardware on the analog frontend, need to act in concert. We describe the synchronization on two levels: (*i*) How to synchronize the Linux host clocks of a number of Shepherd observers using (a) Global Positioning System (GPS) and (b) PTP, and (*ii*) how to locally synchronize the PRUSS cores to the Linux host clock.

GPS. Using GPS allows to globally absolutely synchronize clocks with high accuracy: GPS fundamentally relies on accurate time-of-flight measurements between the receiver and multiple satellites. For this purpose, the receiver has to be tightly synchronized to the satellites. Many receivers output a pulse per second (PPS) signal, a sharp rising or falling edge of an electrical signal with the wrap of every second. Together with information about the global time of that second, it is relatively easy to synchronize another node to the globally accurate GPS time. Using *gpsd* and *chrony*, the host clock of the Linux OS running on the BeagleBone can be synchronized.

The advantage of using GPS is that the Shepherd nodes do not need to be physically connected. For example, Shepherd can be used to explore solar energy harvesting LPWAN networks where nodes can be kilometers apart making physical connections infeasible. However, GPS receivers can be expensive and always require line of sight to the satellites, severely limiting deployment options.

PTP. As an alternative, we consider PTP, which is a time synchronization protocol based on transmission time measurements over standard Ethernet links. A number of slave nodes synchronizes to a common master, which can be elected automatically based on clock quality estimation. TI's AM3358 SoC, the core of the BeagleBone, implements hardware time-stamping of Ethernet packets, significantly improving the achievable accuracy of PTP. PTP is readily available in Linux and, combined with *phc2sys* allows to synchronize the Linux host clocks of multiple Shepherd observers. When combining GPS and PTP, every partition that does not have an Ethernet connection to all other nodes must contain one GPS master to achieve global time synchronization across all nodes.

PRUs. The last stage of the synchronization hierarchy is the synchronization between the Linux host clock and the PRUs. To the best of our knowledge, there exists no standard approach so we developed a custom synchronization procedure.

At a fixed time in every interval (currently after 5 ms in a 100 ms interval), the Linux kernel module timestamps the Linux host clock and immediately sends an interrupt to one of the PRU cores (PRU0). On reception, PRU0 immediately timestamps its own clock and requests the corresponding timestamp from the kernel module. With these two values, the PRU estimates its offset from the host clock. By storing the last pair of timestamps, it also estimates the clock drift with respect to the host clock. This way, the PRUs can accurately schedule all samples until the next synchronization interval.

6 USING SHEPHERD

Public testbeds expose an application programming interface (API) or graphical user interface (GUI) to the user, while the underlying implementation is only relevant to the testbed maintainers. By contrast, users of Shepherd are exposed to the full hardware/software stack. We anticipate two different types of users: The first group has no specific hardware requirements and wants to get started quickly by using the reference hardware and software we provide. Ideally, a user from this group boots up the BeagleBones and runs the provided software to record and emulate harvesting traces. The

second group of users requires dedicated hardware and/or software solutions to achieve their goal. For this group of users, modularity and composability are key: They must be able to quickly change parts at every level of the hardware/software stack.

We aim for a flexible, yet robust solution that satisfies the needs of both types of users. We implement Shepherd using standard Linux interfaces and the most high-level programming language possible for a given task. That is, only the SPI transfer routines are written in assembly; we use C for the Linux kernel module and the firmware of the PRUs; software running in user-space is written in Python. Users interact with Shepherd in four ways.

1) Installation: After mounting the Shepherd cape onto the BeagleBones, users download and flash the latest Ubuntu image to the SD cards by following the instructions on the BeagleBone website. Then they log into each node using the default credentials to set a unique hostname and configure public/private key based `ssh` access. Finally, they deploy the Shepherd software stack by installing two Debian packages, which we provide as release artifacts on the Shepherd repository. These steps can be executed manually for each node or by running the provided Ansible playbook against all nodes with one command from the user's machine.

2) Calibration: Although Shepherd can work with default values derived from the hardware specs, the recording and emulation accuracy is greatly improved when applying per-node calibration, as evaluated in Sec. 8. We provide one walk-through example on how to fully automate the calibration process using a Keithley SMU, which can be controlled remotely over Ethernet or USB. We also provide a second example for users with access to a stock lab-bench power supply and multimeter. The example guides the user through the calibration process in a step-by-step manner, prompting the required reference values on the command line. Using these scripts, the calibration procedure takes roughly 10 minutes per node.

3) Usage: To record and replay harvesting traces, users invoke two command-line utilities. *shepherd-sheep* is run locally on each Shepherd observer node and provides a rich interface to start recording or emulation, run a GUI webserver, or to start a remote procedure call (RPC) interface. A user may use this interface to investigate the behavior of a single battery-less node. Nevertheless, the key functionality of Shepherd is the ability to orchestrate a number of time-synchronized Shepherd nodes. To this end, users can run the *shepherd-herd* tool on any host with `ssh` access to the network of Shepherd observers, which provides similar functionality as *shepherd-sheep* but takes as an argument a list of Shepherd nodes on which the corresponding commands should be executed.

4) Development: We make Shepherd's hardware and software available as open source, and encourage users to provide feedback and develop new features. We help users getting started with three measures: (*i*) modularity and extensive documentation, (*ii*) a test suit for catching software bugs early and showcasing functionality, (*iii*) automation of build and deployment tasks using standard tools.

7 SHEPHERD IN ACTION

This section demonstrates the utility of Shepherd when testing a distributed algorithm for batteryless nodes in a real environment.

Scenario. We consider an indoor solar energy harvesting scenario. The testbed comprises three Shepherd nodes. Each node consists of a BeagleBone Green, the Shepherd cape, a 7 cm × 2 cm solar panel mounted onto a solar capelet, a nRF52840 capelet, and a storage capelet that hosts a 150 µF ceramic capacitor. The three Shepherd nodes are placed on tables inside a room without windows, receiving only little light through a glass door. Node 2 is slightly tilted, facing that glass door, while nodes 1 and 3 are oriented horizontally.

We develop an example application for the nRF52840 that wakes up from system-off mode when it sees a rising edge on the voltage-in-operating-range pin of the BQ25504, which is triggered when the capacitor voltage exceeds 2.8 V. After initialization, the application senses the supply (*i.e.*, the capacitor) voltage every 125 ms. When the supply voltage reaches 3 V, it executes a task that involves sampling the temperature sensor and transmitting the reading to a remote base station by embedding it into Bluetooth Low Energy (BLE) advertisement packets sent on the three corresponding channels. The application continues to sample its supply voltage until the 3 V threshold is again exceeded or the voltage-in-operating-range pin of the BQ25504 is pulled low as the supply voltage falls below 2.3 V. The latter causes an interrupt that puts the nRF52840 into system-off mode from which it only wakes up on a rising edge of the voltage-in-operating-range pin. The application indicates state changes (system-off, sleeping, sampling supply voltage/ADC, transmitting BLE advertisement packets) using GPIO pins. In this way, we can track the logical state of each node with high resolution.

Recording. We record harvesting current and voltage for 60 seconds synchronously on the three Shepherd nodes. Initially, the room lights are switched off. After 30 seconds, we turn the room lights on and keep recording for another 30 seconds.

Fig. 8 plots the voltage and current recorded on the three nodes. Initially, the voltages are low. The spikes at around 12, 30, 45, and 60 seconds are due to MPP tracking. To this end, the BQ25504 shortly disconnects the harvester and samples the solar panel's open-circuit voltage. For the next 16 seconds, it regulates the voltage to 80 % of that voltage. As soon as the light is switched on, the current sharply rises on all three nodes, as shown in the zoomed-in plots to the right. However, the voltage remains low as the converter keeps it at the regulation point. Only after the next MPP tracking at around 40 to 45 seconds, the voltage is also increased to track the new MPP.

Emulation. We replay the recorded traces to the nRF52840 devices running our example application, while recording capacitor voltage, current, and GPIO traces. Fig. 9 shows that all three nodes start in system-off mode. The small amount of energy extracted from the solar panels is sufficient to slowly charge the capacitors while the nodes are still powered off. Following the recorded harvesting traces, node 2 receives most energy and is thus the first to reach the power-on threshold of 2.8 V. The wake-up causes a significant current spike and corresponding voltage drop, while the MCU initializes memory and peripherals. However, as long as the room lights are switched off, the periodic sampling of the supply voltage using the ADC is not sustainable, leading to a decreasing capacitor voltage and eventually power-off. After switching on the room lights, the nodes power up quickly and accumulate enough energy to read out the temperature sensors and send the readings to the base station. We in the zoomed-in plots to the right that the nodes execute the task at different times and that the execution drains the capacitor voltage much faster than it rises during charging.

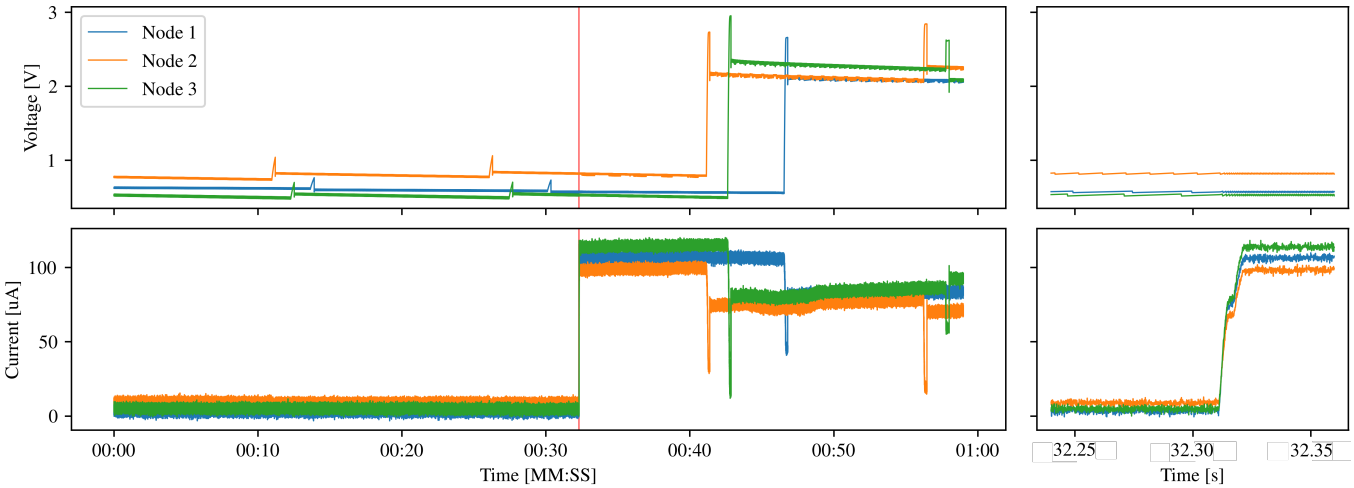


Figure 8: Voltage and current traces recorded by three Shepherd nodes in an indoor solar energy-harvesting scenario. The plots on the right zoom into the moment the room lights are switched on (marked in red on the left).

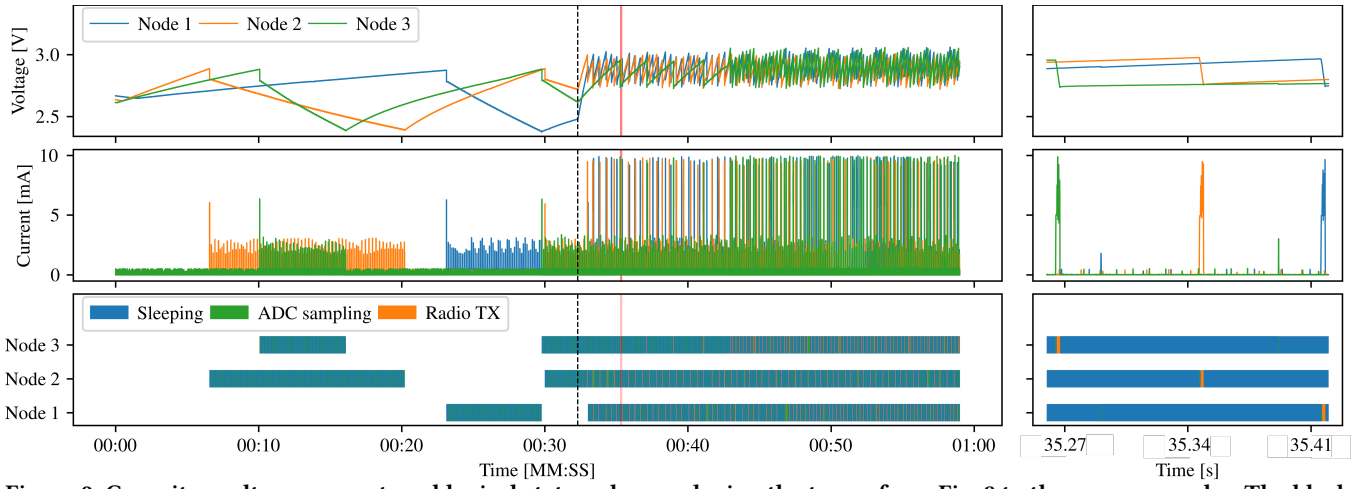


Figure 9: Capacitor voltage, current, and logical states when replaying the traces from Fig. 8 to three sensor nodes. The black dashed line indicates when the lights are switched on. The plots on the right zoom into the time marked with a red solid line.

Reproducibility. One of Shepherd's strengths is its ability to emulate spatio-temporal energy availability. Given a deterministic application, this should also lead to a consistent behavior across successive emulation runs with the same harvesting traces. To quantify this reproducibility, we consider three parameters of our example application that may be of interest to a developer: (*i*) *#packets* is the total number of packets sent by a node during a 60-second experiment with the recorded traces from Fig. 8; (*ii*) *wake-up time* is the time from the start of an experiment until the node executes the sense-and-send task for the first time; (*iii*) *sleep time* is the total time spent in sleep mode (*i.e.*, powered on and waiting for an interrupt).

We replay the traces ten times and measure the three application-level parameters for all three nodes. Table 1 lists for each parameter and node the mean and the *error*, defined as the maximum absolute difference between any two repetitions. The errors are very small across all nodes and parameters. This demonstrates the ability of Shepherd to accurately reproduce application behavior, which can greatly aid in the development and evaluation of batteryless applications and system services. Overall, the observations and insights presented throughout this section would be very difficult, if at all possible, to attain without a tool like Shepherd.

Table 1: Application parameters when replaying the same energy-harvesting traces to three nodes ten times. The *error* is the max. absolute difference between any two repetitions.

	#packets		Wake-up time		Sleep time	
	Mean	Error	Mean	Error	Mean	Error
Node 1	238.5	4	33.4 s	0.1 s	23.6 s	0.7 s
Node 2	177.1	6	32.8 s	0.3 s	36.3 s	0.8 s
Node 3	226.6	9	35.1 s	0.5 s	26.4 s	0.9 s

8 PERFORMANCE EVALUATION

The previous section showed some of Shepherd's capabilities, suggesting that Shepherd meets the performance requirements from Sec. 3. This section shows that this is indeed the case for the current hardware/software stack by answering the following questions:

Table 2: Summary of Shepherd performance specification.

Sampling rate		100 kHz
Range	Harvesting voltage	100 mV-3 V
	Harvesting current	0 mA-50 mA
	Load voltage	0 V-4 V
	Load current	0 mA-50 mA
	Emulation voltage	100 mV-3 V
	Emulation current	0 mA-50 mA
24 h DC Accuracy	Harvesting voltage	19.53 µV ± 0.01 %
	Harvesting current	381 nA ± 0.07 %
	Load voltage	19.53 µV ± 0.01 %
	Load current	381 nA ± 0.01 %
	Emulation voltage	11 µV ± 0.012 %
	Emulation current	191 nA ± 0.025 %
RMS Noise (@1 kHz)	Harvesting voltage	50 µV (14 µV)
	Harvesting current	3 µA (0.4 µA)
	Load voltage	48 µV (10 µV)
	Load current	4.5 µA (0.9 µA)
Bandwidth	Recording channels	15 kHz
Risetime	Emulation voltage	65 ms
	Emulation current	19.2 µs
Max. burden voltage	Harvesting recorder	50.4 mV
	Load recorder	76.1 mV
Min. GPIO sampling rate		580 kHz
Avg. current draw		345 mA
Max. synchronization error		2.4 µs

- What is the time difference between two nodes when taking or replaying a sample at the supposedly same time?
- What are the electrical characteristics affecting the resolution, accuracy, and sampling rate of voltage and current?
- What is the resolution, frequency, and synchronization with which Shepherd can trace GPIO pin changes?
- Does the average power draw of a Shepherd node allow for extended experiments without mains power supply?

For reference, Table 2 summarizes the main performance characteristics of our current Shepherd implementation.

8.1 Time Synchronization Accuracy

Ideally, we would measure synchronization accuracy by comparing the times when the ADCs on two Shepherd nodes close their sample and hold switch. However, this switch is not accessible from outside. According to the datasheet, the ADC starts the conversion on the falling edge of the SPI chip select (CS) signal. We thus measure the delay between 10,000 consecutive falling edges of the SPI CS signal on two nodes to determine the synchronization accuracy. There may be jitter introduced by the circuitry inside the ADC/DAC, but we expect it to be small. We consider two setups:

- **Setup A** (Fig. 10): In this setup, we use two nodes that are connected over an off-the-shelf Ethernet switch and synchronized to a common PTP master within the same network.

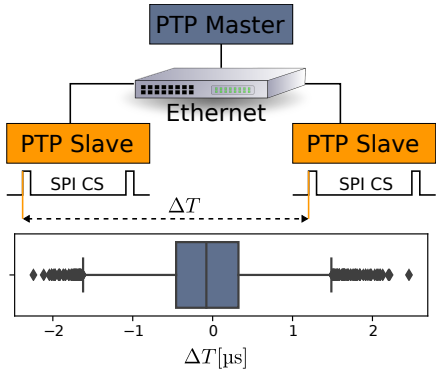


Figure 10: In setup A, two Shepherd nodes are synchronized to one PTP master over a COTS Ethernet switch.

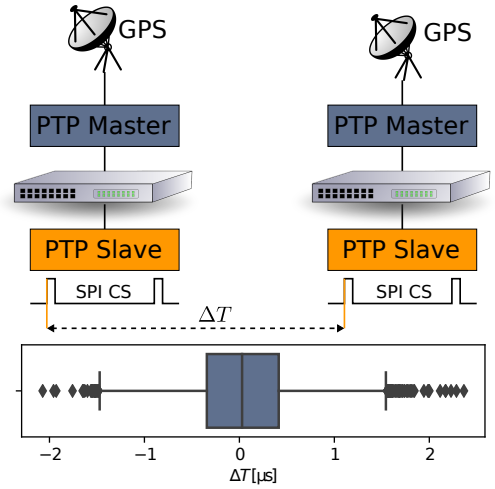


Figure 11: In setup B, two Shepherd nodes are part of two separate Ethernet networks with their own PTP master that is synchronized to global GPS time.

- **Setup B** (Fig. 11): We use four nodes in two separate Ethernet networks. Each network consists of a Shepherd node acting as PTP slave that is connected over Ethernet to a PTP master with a GPS reference clock. The two networks are physically separated, representing scenarios where nodes are mobile or too far from each other for a direct, wired connection.

We plot the median and 25th/75th percentiles of the synchronization error at the bottom of Figs. 10 and 11; the dots to either side of the whiskers are outliers. Even with the outliers, we find that the maximum synchronization error across both setups is as small as 2.4 µs: 91 % of the measurements are within our targeted synchronization accuracy of 1 µs, also for setup B in which the Shepherd nodes have no wired connection between each other (see Fig. 11).

8.2 Electrical Characteristics

Zero-input root mean square (RMS) noise. The noise floor crucially determines the effective resolution, that is, the smallest signal change that can be differentiated. There are various sources of noise in the signal acquisition chain, including switching noise from the DC/DC converter and thermal noise from the shunt resistors. Assuming that the noise has zero mean, is uncorrelated,

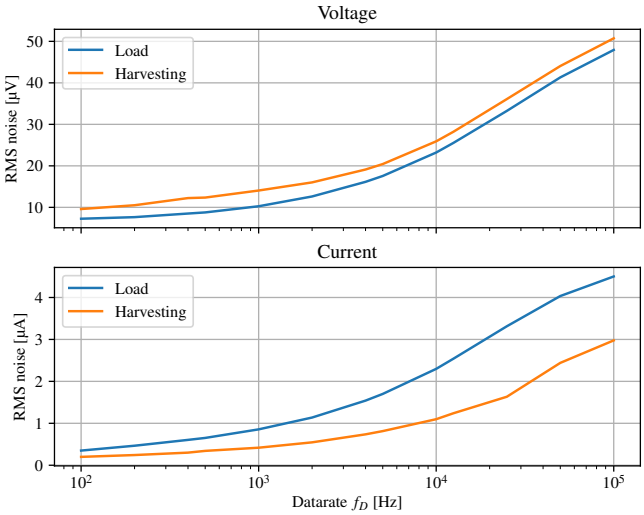


Figure 12: RMS noise against data rate for all four channels.

and uncorrelated to the measured signal, the noise power can be reduced, for example, by a factor of two by averaging over four consecutive samples at the cost of a decreased data rate.

We use a Keithley SMU2604B SMU to apply a zero-input signal to each of the four channels of a Shepherd node (*i.e.*, harvesting voltage/current, and load voltage/current) and sample for 10 seconds at Shepherd's fixed sampling rate of $f_S = 100\,\text{kHz}$. Fig. 12 plots the RMS of all four channels against the data rate f_D after averaging over f_S/f_D samples. We see that Shepherd's voltage resolution is much better than the required 1 mV, and that the current resolution is within the required 1 µA for data rates below 8 kHz.

Table 3: Maximum burden voltage and impedance.

Channel	Maximum burden voltage [mV]	Impedance [Ω]
Harvesting	50.4	1.008
Load	76.1	1.342

Burden voltage. Measuring current with a shunt resistor introduces a voltage drop between the source and the load. Similarly, the switches used to select between recording and emulation for the harvesting channel as well as between dummy load and sensor node for the load channel also cause a voltage drop that would not be present in an ideal system. To measure the burden voltage, we use the Keithley SMU from before to perform a current sweep with a resolution of 1 mA. For every value, we measure the resulting voltage drop over the measurement circuitry and the control logic.

The results reveal a linear dependency between the burden voltage and the current, and that the burden voltage is dominated by the respective shunt resistor. Table 3 lists the maximum burden voltage and the corresponding impedance. Because current and voltage on a harvester typically correlate, the impact of losses due to burden voltage are small; for example, the loss is less than 2 % for the maximum current and voltage supported by Shepherd.

Recording bandwidth. We use a 51 Ω resistor as load and an AIM-TTI TG5011 function generator to measure the bandwidth of all four recording channels by configuring it for an amplitude of 2 V and a linear frequency sweep of up to 50 kHz. We define the bandwidth as the frequency at which the measured amplitude falls below −3 dB with respect to the 2 V input. We find that the bandwidth of all four channels ranges between 15.1 kHz and 15.2 kHz, which corresponds to the nominal bandwidth of the low-pass filter built into the ADC.

Table 4: DC accuracy in terms of mean absolute percentage error (MAPE) without calibration and 24 hours after calibration. LSB represents the measurement resolution.

Channel		LSB	MAPE after 24 h [%]	MAPE uncalibr. [%]
harv. rec.	voltage	19.53µV	0.011	0.150
	current	381nA	0.072	0.918
load rec.	voltage	19.53µV	0.028	0.051
	current	381nA	0.018	0.714
harv. emu.	voltage	11µV	0.012	5.962
	current	191nA	0.025	20.450

Emulation rise time. A critical parameter for the harvesting emulator is the time it takes to change its operating point. We measure the rise time for the current source by inserting a resistor between its output and ground, and then applying a low-frequency square wave with an amplitude from 0 A to 50 mA through the DAC. For the voltage emulation path, we use a 220 µF capacitor and the dummy load, and apply a constant current of 500 µA using a Keithley 2604B. We measure the regulated voltage at the input of the DC/DC converter while applying a low-frequency square wave with an amplitude from 100 mV to 3 V through the DAC.

We measure the time it takes for the signal to rise from 10 % to 90 % of its range with an Agilent MSO7104B oscilloscope. The 19.2 µs rise time of the current source is within expectations and short enough to accurately emulate rapidly changing conditions. For the voltage channel, we observe a long rise time of 64 ms. A closer examination reveals that the set-point voltage provided by the DAC rises to its value within a few µs, but the BQ25504 only applies the reference voltage with its internal duty-cycle of 64 ms. However, this is not critical as the voltage is regulated: it typically does not change in response to changing conditions.

DC accuracy. DC accuracy quantifies how close a value measured or emulated with Shepherd is to the true value. To measure it, we use the Keithley SMU to do a sweep over the full range of the four recording channels, and log the value measured with Shepherd together with the corresponding reference value. For the two emulation channels (harvesting load/current), we reverse the setup and apply the sweep using Shepherd. For the current source, we use a 91 Ω resistor as the load and the SMU as amperemeter. For the voltage emulation path, we use a 220 µF capacitor and the dummy load, and apply a constant current of 500 µA to the SMU. We measure the regulated voltage at the input of the DC/DC converter.

We do these measurements without calibration and then again 24 hours after calibration. The mean absolute percentage error (MAPE) and the least significant bit (LSB) indicating the measurement resolution are shown in Table 4. We see that the calibration process described in Sec. 6 improves the DC accuracy considerably, in particular for the two emulation channels. The MAPE values 24 hours after calibration are below 0.1 % across the board, demonstrating that Shepherd can accurately record and replay harvesting traces.

Table 5: Comparison of Shepherd with Ekho [10].

	Ekho [10]	Shepherd
Sampling rate	0.135 kHz/1 kHz	100 kHz
Nominal current resolution	10 µA	0.381 µA
Current emulation error	(86.9 ± 46.2) µA	(0.51 ± 0.29) µA

8.3 GPIO Tracing Performance

Shepherd can sample and store the state of up to four GPIO pins during recording and emulation. The state is continuously polled by the PRU firmware. We measure the maximum polling interval by toggling a pin within the corresponding routine and measuring the maximum delay between two edges with a Salea Logic 8 logic analyzer. The maximum delay is 1.7 µs. This is the minimum time a pin must be high or low for the change to be recorded. It is also the maximum delay between the event and the recorded timestamp. The size of internal buffers limits the maximum frequency to 163,840 events per second, where an event represents the state change of at least one pin. As GPIO timestamping is done with the same clock used for scheduling samples, the synchronization error is in the same region as the results in Figs. 10 and 11, that is, 2.4 µs at most.

8.4 Power Draw of a Shepherd Node

We expect Shepherd to be used in scenarios without mains power supply. We measure the current draw of a Shepherd node while recording data using a Keithley 2604B SMU, supplying the node with 5 V through the micro USB connector. The average current draw is 345 mA with a peak current of 395 mA. Using a u-blox M8F GPS sensor for time synchronization adds another 28 mA once the receiver has acquired the first fix. This allows for a theoretical recording duration of 19 h from a 10 000 mAh USB power bank.

9 RELATED WORK

Two areas of prior work are related to Shepherd: (*i*) tools to record and emulate energy-harvesting environments, and (*ii*) testbeds for battery-supported nodes, with or without energy-harvesting capabilities. Debugging techniques for intermittent systems, however, are orthogonal to our work; for instance, the platform proposed in [4] may be integrated into Shepherd for energy-interference-free debugging of distributed batteryless applications. Shepherd deals with energy harvesting across a collection of nodes, which is independent of the communication technology used to exchange data between nodes. By developing additional capelets, Shepherd can thus be used to experiment not only with active radios, but also with ambient backscatter [16] or visible light transceivers [14].

Recording and emulation of energy sources. Ekho [10] records IV curves of a harvester and recreates these characterstics in the lab as input to a converter-less node. Ekho supports different harvesting technologies and can reproduce the energy environment of a single node, but it cannot provide insights into the spatio-temporal energy environment and behavior of multiple distributed nodes. Shepherd, instead, offers synchronized recording and emulation of multiple harvesters. The performance of Ekho has not been characterized in terms of noise levels, DC accuracy, or dynamic range, making a quantitative comparison with Shepherd difficult. Nevertheless, Table 5 lists the performance results provided in [10] along with the corresponding values for Shepherd, obtained by computing the mean and standard deviation of the absolute current emulation error across 24 hours based on the measurements underlying Table 4. We find that Shepherd achieves orders of magnitude better sampling rate, nominal current resolution, and current emulation error.

RocketLogger [24] is a hand-held device that enables in-situ measurements across four voltage and two current measurement channels with high accuracy and wide dynamic range. It also records temperature, illuminance, etc. to support the analysis of environmental statistics. Similar to Shepherd, RocketLogger aims to bring the capabilities of high-quality, expensive, wall-powered lab equipment into a compact and portable measurement device. Unlike Shepherd, RocketLogger cannot replay recorded or generated traces, and is only applicable to single energy-harvesting devices.

Recent work aims at repeatable indoor testing of solar-powered nodes by controlling the intensity of a light source (*e.g.*, based on real illumination data) to induce a solar cell or panel to generate a desired level of power [9, 19, 27]. Using enclosures for the node or photovoltaics, it is possible to create repeatable light conditions. All solutions are specifically designed for a certain battery-supported platform. By contrast, Shepherd's design is platform-agnostic and applicable to batteryless devices and different harvesting sources. Moreover, it can be used to record the harvesting conditions, which is not possible with any of the proposed solutions.

Testbeds for embedded wireless nodes. Testbeds for battery-supported nodes enable development, distributed debugging, and performance measurements. The capabilities of existing testbeds range from basic support for reprogramming and serial logging [8] through synchronized GPIO tracing/actuation [15] and JTAG debugging [25] to power profiling [15] and the generation of controllable Wi-Fi interference [23]. Shepherd is inspired by these testbeds and brings many of their services to batteryless, intermittent systems. In addition, it adopts a fundamentally different approach to provide services like recording and emulation of spatio-temporal energy availability that no existing testbed offers.

Indeed, a few papers outline the challenges and desired capabilities of a testbed for energy-harvesting [28] or transiently powered sensor nodes [1]. The prototypes, however, lack even basic features such as time synchronization and energy consumption measurements [1] or record and replay of harvesting traces [28]. Shepherd provides a more powerful solution that is available as an affordable, portable, and open-source tool for the research community.

10 CONCLUSIONS

We have presented Shepherd, a testbed for collections of batteryless devices that can accurately record and replay the spatio-temporal characteristics of real energy environments. Shepherd's modular design is agnostic to harvesting source, energy storage, node platform, and communication technology; it is affordable and portable; and our experiments show that it provides adequate accuracy, resolution, sampling rate, and time synchronization. We believe Shepherd can be a valuable tool for the research community to investigate exciting questions that have been out of reach so far.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) within the Cluster of Excellence cfaed (grant EXC 1056) and the Emmy Noether project NextIoT (grant ZI 1635/2-1).

REFERENCES

[1] Henko Aantjes, Amjad Y. Majid, and Przemysław Pawełczak. 2016. A Testbed for Transiently Powered Computers. arXiv:arXiv:1606.07623

[2] Naveed Anwar Bhatti and Luca Mottola. 2017. HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[3] Albert Cohen, Xipeng Shen, Josep Torrellas, James Tuck, Yuanyuan Zhou, Sarita Adve, Ismail Akturk, Saurabh Bagchi, Rajeev Balasubramonian, Rajkishore Barik, Micah Beck, Ras Bodik, Ali Butt, Luis Ceze, Haibo Chen, Yiran Chen, Trishul Chilimbi, Mihai Christodorescu, John Criswell, Chen Ding, Yufei Ding, Sandhya Dwarkadas, Erik Elmroth, Phil Gibbons, Xiaochen Guo, Rajesh Gupta, Gernot Heiser, Hank Hoffman, Jian Huang, Hillery Hunter, John Kim, Sam King, James Larus, Chen Liu, Shan Lu, Brandon Lucia, Saeed Maleki, Somnath Mazumdar, Iulian Neamtiu, Keshav Pingali, Paolo Rech, Michael Scott, Yan Solihin, Dawn Song, Jakub Szefer, Dan Tsafrir, Bhuvan Urgaonkar, Marilyn Wolf, Yuan Xie, Jishen Zhao, Lin Zhong, and Yuhao Zhu. 2018. *Inter-Disciplinary Research Challenges in Computer Systems for the 2020s.* Technical Report. USA.

[4] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson P. Sample. 2016. An Energy-interference-free Hardware-Software Debugger for Intermittent Energy-harvesting Systems. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[5] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proceedings of the 2016 ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*.

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

[7] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni Vincentelli. 2012. Modeling Cyber-Physical Systems. *Proc. IEEE* 100, 1 (2012).

[8] Mun Choon Doddavenkatappa, Manjunath Chan and Ananda A. L. 2011. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *Proceedings of the ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*.

[9] Lars Hanschke, Christian Renner, Jannick Brockmann, Tobias Hamann, Jannes Peschel, Alexander Schell, and Alexander Sowarka. 2017. Light in the Box: Reproducible Lighting Conditions for Solar-Powered Sensor Nodes. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[10] Josiah Hester, Lanny Sitanayah, Timothy Scott, and Jacob Sorber. 2017. Realistic and Repeatable Emulation of Energy Harvesting Environments. *ACM Transactions on Sensor Networks* 13, 2 (2017).

[11] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[12] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[13] Neal Jackson, Joshua Adkins, and Prabal Dutta. 2019. Capacity over Capacitance for Reliable Energy Harvesting Sensors. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks (IPSN)*.

[14] Toshihiko Komine and Masao Nakagawa. 2004. Fundamental Analysis for Visible-Light Communication System using LED Lights. *IEEE Transactions on Consumer Electronics* 50, 1 (2004).

[15] 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 ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[16] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R. Smith. 2013. Ambient Backscatter: Wireless Communication out of Thin Air. In *Proceedings of the ACM SIGCOMM Conference*.

[17] Brandon Lucia, Vignesh Balaj, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proceedings of the 2nd Summit on Advances in Programming Languages (SNAPL)*.

[18] Kiwan Maeng and Brandon Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[19] Robert Margolies, Maria Gorlatova, John Sarik, Gerald Stanje, Jianxun Zhu, Paul Miller, Marcin Szczodrak, Baradwaj Vigraham, Luca Carloni, Peter Kinget, Ioannis Kymissis, and Gil Zussman. 2015. Energy-Harvesting Active Networked Tags (EnHANTs): Prototyping and Experimentation. *ACM Transactions on Sensor Networks* 11, 4 (2015).

[20] Sujay Narayana, R. Muralishankar, R. Venkatesha Prasad, and Vijay S. Rao. 2019. Recovering Bits from Thin Air: Demodulation of Bandpass Sampled Noisy Signals for Space IoT. In *Proceedings of the 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[21] Rui Rocha, Jorge Dias, and Adriano Carvalho. 2005. Cooperative Multi-Robot Systems: A Study of Vision-based 3-D Mapping using Information Theory. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*.

[22] Fred B. Schneider. 1990. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22, 4 (1990).

[23] Markus Schuß, Carlo Alberto Boano, Manuel Weber, Matthias Schulz, Matthias Hollick, and Kay Römer. 2019. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controlable and Repeatable Wi-Fi Interference. In *Proceedings of the 16th International Conference on Embedded Wireless Systems and Networks (EWSN)*.

[24] 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 EDAA Conference on Design, Automation & Test in Europe (DATE)*.

[25] 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)*.

[26] Philip Sparks. 2017. *The route to a trillion devices: The outlook for IoT investment to 2035.* Technical Report.

[27] Wilson M. Tan, Paul Sullivan, Hamish Watson, Joanna Slota-Newson, and Stephen A. Jarvis. 2017. An Indoor Test Methodology for Solar-Powered Wireless Sensor Networks. *ACM Transactions on Embedded Computing Systems* 16, 3 (2017).

[28] Ashok Samraj Thangarajan, Fan Yang, Wouter Joosen, and Danny Hughes. 2018. Real-time Distributed In-Situ Benchmarking of Energy Harvesting IoT Devices. In *Proceedings of the 5th ACM Workshop on Middleware and Applications for the Internet of Things (M4IoT)*.