Delft University of Technology
Master of Science Thesis in Embedded Systems

Ultra-low-power Architecture for Wireless Internet of Things

Haozhe Tang

TU Delft Delft University of Technology

Ultra-low-power Architecture for Wireless Internet of Things

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Haozhe Tang
h.tang-1@student.tudelft.nl
philo.tang@foxmail.com

23th August 2021

Author
Haozhe Tang (h.tang-1@student.tudelft.nl)
(philo.tang@foxmail.com)
Title
Ultra-low-power Architecture for Wireless Internet of Things
MSc Presentation Date
27th August 2021

Graduation Committee

Dr. Przemysław Pawełczak (chairman)	Delft University of Technology
Dr. Zekeriya Erkin	Delft University of Technology

The work presented in this thesis has led to a paper that is going to be submitted to a conference for publication soon.

Abstract

Battery, one of the least ecology-friendly components of (embedded) computing systems, should be removed and replaced by renewable energy sources. However, renewable energy is unstable and unpredictable. Therefore, a large body of research work focuses on making computing systems operational despite frequent power interrupts. Unfortunately, efforts in making wireless communication systems intermittently-powered without any compromise to their functionality have been fruitless. To address this challenge we present an architecture for intermittently-powered wireless communication systems. Core novelties of our architecture are: (i) time-deterministic process-driven state restoration architecture, and (ii) time and peripheral abstraction layer. As a case study for our architecture, we build FreeBie: a battery-free intermittently-powered Bluetooth Low Energy device. To the best of our knowledge FreeBie is the first battery-free active radio (non-backscatter) wireless system that sustains bi-directional communication on intermittent harvested energy supply.

Preface

This MSc thesis has been carried out at the Delft University of Technology, within the Sustainable Systems Lab of the Embedded and Networked Systems group, under the daily supervision of Jasper de Winkel, a Ph.D. candidate of the group.

The COVID-19 pandemic hit the world hard during the work of this thesis. For the majority of the time, I had to work on this thesis from home. Many components that we needed for the project were sold out, and the logistics were extensively delayed. Despite all the challenges, the thesis is finished. I am very grateful for the strong support that I receive from Jasper de Winkel and Dr. Przemysław Pawełczak during this difficult time. Without them, I do not think I can make this happen. I also want to thank Dr. Zekeriya Erkin for accepting to join the thesis committee and attend my defense. Finally, I want to thank my parents for their wholehearted support, which I always get no matter where I am or what I do. I always love them just as they always love me.

Haozhe Tang

Delft, The Netherlands
23th August 2021

Contents

Chapter 1

Introduction

More than billions of Internet of Things (IoT) devices will surround us in the coming years [90, 7]. This implies that the batteries (powering the IoT) world need to be regularly replaced, monitored, and recycled [31]—inducing a major environmental impact to our planet and monetary cost to the consumer [28, 47]. While the search for a battery that is longer-operational [31], better recyclable [86] and having high-energy density [114] continues, the road leading to such batteries is still long [126, 13].

A dedicated line of research addressing the battery sustainability problem is radical in its approach, as it tries to find out how we can build (wireless) IoT completely independent from batteries [50, 110, 101, 92]. In such systems conventional battery is either removed or replaced by cheaper, non-aging, non-leaking, lighter,[1] less dependent on sparsely-available chemical elements and temperature-stable energy reservoir: a capacitor. The energy to power such battery-free system comes from ambient sources, for example from solar radiation [22] or Radio Frequency (RF) emissions [43]. Nonetheless, unique properties of battery-free systems—small storage and unpredictable and intermittent energy supply—create one of its kind challenges for the design of IoT wireless communication.

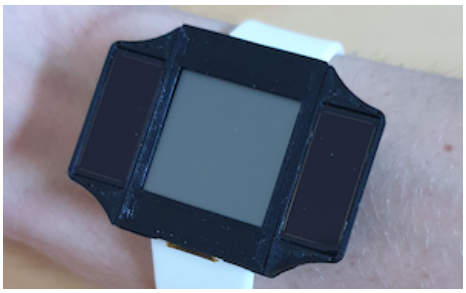


Figure 1.1: **First truly battery-free open-source [2] smartwatch with FreeBie: intermittently-powered BLE communication.**

[1] Battery was the heaviest component of the weight-optimized BLE node [63, Table 1], or a battery is just bigger than size-optimized BLE node [15, Figure 1].

1.1 Problem Statement

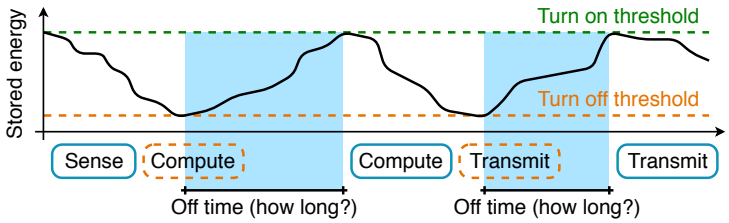


Figure 1.2: **Symbolic time trace example of stored energy for intermittently-powered device: active computation periods interwoven by periods of complete off times—both periods of unknown duration. [26]**

Any IoT sensor, battery-based or battery-free, requires a wireless link to communicate. This link needs to be (i) low-power—to operate as long as possible on a single energy charge, (ii) belonging to one of the mainstream wireless standards (such as LoRa [66] or BLE [128])—to be backward compatible with already deployed networks, (iii) independent from external infrastructure—to be flexibly deployable, and (iv) bidirectional—to enable remote maintenance and firmware updates [6, 9]. To address requirement (i)–(ii) plenty of research effort has been dedicated to enabling ultra-low-power communication for popular wireless communication standards—leading to battery-free operation—through the principle of backscatter. Recent examples of such wireless standard-compliant backscatter-based transmitters include [66] (LoRa), [128] (BLE) or [20] (Long Term Evolution (LTE)). These wireless systems however do not address requirement (iii). Simply, backscatter-based transmitters must either rely on dedicated signal exciters or other active radio infrastructure to backscatter on. A related approach based on wake-up radio [94] is also not viable for the same reason. Therefore the only solution to battery-free IoT that addresses requirements (iii) is an ultra-low-power active radio. However, intermittent operation of a battery-free active radio (i.e. non-backscatter) IoT node implies that the active radio communication becomes also intermittent, see Figure 1.2. Therefore, a design of a truly bi-directional (connection-based) wireless active link for a battery-free IoT sensor—addressing requirement (iv)—needs to sustain already established connection despite frequent power interrupts. Sadly, all battery-free intermittently-powered state-of-the-art active radio platforms available are connection-less such as beacon-only (best-effort) BLE communication [42]. Simply, classical techniques to prolong IoT life based on duty cycling do not apply, as intermittent operation takes away control over the wake-up times of the device. Moreover, power off the device from intermittent harvested energy de-synchronizes connection between the end device and host and disallows precise timing. All this results in wireless communication being one of the unsolved challenges in intermittent computing domain [115, Section 4(b)]. Therefore a battery-free IoT that is useful, i.e. that addresses requirements (i)–(iv), has yet to be achieved.

1.2 Contribution

Addressing the problem of battery-free intermittently-powered active radio IoT, we provide the following contributions.

C1 **Architecture for intermittently-powered connection-based active radio system.** We propose a new wireless system architecture in which the intermediate state of a non-backscatter-based battery-free radio system (that is, application and all lower layers, including physical layer) is stored in external non-volatile memory and is being restored after enough ambient energy is harvested. In consequence, this enables sub-zero current standby without any changes to wireless radio front-end and/or Microcontroller Unit (MCU) electronics while sustaining the already established connection. Our core novelty is a new system state checkpointing in which separate checkpoint layers for application and radio layers are handled individually and where the checkpoints are triggered by the radio process completion time. The separation of layers allows for much faster store and memory reload operation, especially when the memory footprint of the wireless radio stack and upper layers differ. Checkpoints triggered by the process end allow for a low probability of radio state disruption by unnecessary in-between checkpoints. Moreover, our novel super-low power timekeeping architecture cooperating with our time-deterministic checkpointing system allows for microsecond-level time measurement granularity allowing for keeping track of radio protocol state in-between power interrupts and resuming already established communication when energy conditions allow. Both of the architectural novelties give rise to a new concept of power supply time and peripheral abstraction to the radio layer: the radio software stack never sees that the radio is powered off even when the storage capacitor is fully depleted.

C2 **Implementation of fully functional bi-directional intermittently-powered active wireless link.** Based on our proposed architecture we pick one of the most ubiquitous wireless technology for the IoT: Bluetooth,[2] and in particular its low energy configuration BLE [12], and build a truly intermittently-powered non-backscatter BLE radio system which we release as open source [2]. With our architecture, battery-free BLE node can communicate bidirectionally with a conventional BLE access point and can sustain already established bi-directional BLE link, even after multiple full power supply outages at the battery-free BLE node. FreeBie used in the system evaluation integrated into a fully-functional battery-free smartwatch is shown in Figure 1.1.

C3 **Novel battery-free IoT applications.** Our intermittently-powered radio architecture enables never before seen applications, in particular fully-functional BLE device firmware update performed battery-free on an intermittent power supply.

[2]Bluetooth Special Interest Group expects almost 5 billion Bluetooth device shipments in 2022 alone [10].

Chapter 2

Motivation

Since the first publication advocating battery-free computation and sensing [92], many battery-free devices of increased level of complexity have been demonstrated, ranging from simple battery-free switches [130], computational Radio Frequency Identification (RFID) tags [121] and battery-free handheld gaming consoles [27] to autonomous sensors [73] and implantable on-body [117] and disposable sensors [29] where use of batteries is simply impossible. Recent real-life sensor networks deployment reports articulate that a quest for battery-free IoT must be intensified. For instance, a report on 3.5 year-long underground archaeological site wireless sensing concluded that [4, Section 4.2] "(battery) maintenance represents a hampering factor regardless of the value of the data". Beyond academia, calls for freeing the IoT from batteries is echoed by the IoT industry. For instance, a recent study of large-scale (more than 12 000 devices) commercial three-year-long BLE deployment [30] concluded that BLE beacons were out of power before the expected operation of 24 months after the deployment; moreover the BLE beacons themselves were polluting the environment as abandoned beacons were never collected back due to high cost of human labor.

As we indicated in Chapter 1 battery-free operation powered by harvested energy implies that power supplied to the (IoT) device is intermittent. Taking a very recent example, long-term experiments run with a commercial-grade battery-free BLE node of [24] demonstrate that time of the day, orientation, and location inside of a deployment location can affect the duration of continuous operation: from almost constant operation to few transmission activities throughout the day only [69, Section 5]. Therefore necessary system support for intermittently-powered devices performing computation is needed that takes care of [81, Section 2] (i) control flow—to guarantee that the device will start from the state right before the last power failure instead of starting from scratch, (ii) data consistency—to guarantee that the system will restore correctly from power failure, (iii) environmental consistency—to guarantee that the time-sensitive data will be handled correctly when collected sensor data becomes outdated after restoring from power failure, (iv) concurrency—to enable execution of concurrent applications, and (v) inter-device communication—to enable wireless communication between (intermittently-powered) devices and guarantee synchronization despite power interrupts.

While there are already plenty of frameworks available that aim at attaining points (i)–(iv), sustaining a communication of a radio link (let it be backscatter-

based or active) powered by an intermittent source is (i.e. attaining point (v)) is far from being solved—none of the frameworks we are aware of is able to support radio communication fulfilling all requirements listed in Chapter 1.

2.1 Infeasible Solutions for "Intermittent" Communication

A typical workaround is a provision of sufficiently-large capacitor or large energy harvesting source (e.g. large area solar panels) that allows for a single atomic operation—sending of a packet, or a train of packets if connection-based data exchange is needed between transmitter and receiver (for instance, for updating device' firmware). Large capacitor or large energy harvester, however, induces long charge time or large size of the whole device that are detrimental to the sampling frequency for sensing-driven applications, but most importantly it still does not guarantee that the chosen capacitor size would be sufficient—even for cascaded capacitor architectures as in [22, 49]. So all active radio intermittently-powered devices trade-off capacitor size for communication functionality, sending connection-less data, such as beacons in case of BLE, e.g. [65, 111]. Simply, when the device powers off mid-transmission of a beacon, the device will re-start and re-send the beacon again until successful. This approach, however, is not feasible for connection or handshake-oriented protocols, especially if they have strict timing requirements to establish and sustain a connection.

Other classical techniques such as duty cycling (setting system regularly to low power or sleep mode) [18] and transmission power control [35] are also infeasible. Duty cycling is infeasible for intermittently-powered wireless communication, as the main capacitor may deplete within the sleep period resetting connection state, while duty cycling itself consumes energy. Transmission power control while saves energy of the device the transmitted power reduction does not translate to large overall gains for the device; for an example BLE module 6.3 times reduction in transmission power reduces current consumption only by 1.44 [112, Table 7] (see also [71, Table 4]).

The ultimate goal is to create a framework that would enable real communication on intermittently-powered budget. Such a framework needs to fulfill the following set of requirements.

Requirement 1: Safe and reliable restoration of wireless protocol state after power failure. If timing requirements are not violated—that is, if time passed in-between power failures is within the allowed delay—wireless protocol should seamlessly resume without the need to restart the already established connection. This resumption must be supported by a dedicated software framework. Such framework is responsible for making a copy of the protocol state i.e. volatile registers, to a non-volatile memory right before the power interrupt and restoring the last saved state from non-volatile memory back to the respective volatile registers.

There are two main classes of such frameworks that optimize the amount of memory to store and resume: task-based, e.g. [84] and checkpoint-based, e.g. [73]. Task-based systems are built around the concept of a task, i.e. a section of code with a defined input and output variables. Tasks, through these input

	InK [125]	**TICS** [73]	**Coala** [84]	**Capybara** [22]	**MPatch** [27]	**Empire** [4]	**FreeBie**
Checkpoint trigger	Task end	In-code checkpoint	Task end	Task end	Voltage threshold	Voltage threshold	Process end
Region of state preservation	Task	Checkpoint	Task	Task	Checkpoint	Task	Process
Requires code instrumentation	Yes	No	Yes	Yes	No	Yes	No
Non-volatile main memory	No	Yes	No	No	Yes	No	No
Interrupt support	Yes	Yes	No	No	Yes	No	Yes
Real-time restoration	No	No	No	No	No	No	Yes
Supports dynamic memory	No	Yes	Yes	No	No	No	Yes
Concurrent applications support	Yes	No	No	No	No	No	Yes

Table 2.1: **Comparison of relevant existing software support systems for intermittently-powered devices. Color scheme: denotes non-desired or limiting feature from the perspective of wireless communication protocol and denotes the opposite.**

and output variables, are connected with each other to form a state machine. Tasks store results of their intermediate variables to a non-volatile memory right at the end of their operation. If a system fails from power off then it resumes from the last state in a task chain restoring only the variables used by the last task. Unfortunately transformation of code into tasks, in addition to annotating by when a certain variable is not useful anymore after resumption from power off, is not feasible due to the very large program code base and complexity of function dependencies of a typical wireless protocol implementation—we speculate that it would take months to do so manually, while nobody has demonstrated a successful automatic task transformation for such a large code base. Therefore task-based systems are not suited for intermittently-powered wireless protocols, as we need a framework that can deal with such complexity with no penalty to the developer (in terms of development time) or to the system (in terms of monetary cost deploying such a system). A conceptual counterpart to a task is a checkpoint, i.e. a function inserted manually or automatically at compile time to the original code base that stores the state of the program until that checkpoint. Checkpoints can be defined such that they can store all system state (non-volatile registers, stack) or store specific register variables to non-volatile memory (reducing memory footprint and store and restoration time). While, intuitively, a checkpoint-based system is simpler to implement than a task-based system (they require less code transformation), both tasks and checkpoints introduce computation penalty—the store and restore operation—which will break the timing of the original wireless protocol implementation. In other words, checkpoints or tasks cannot be placed inside of any of the wireless protocol functions.

Fortunately, state saving to and restoration from non-volatile memory can be triggered not only by software, i.e. by the end of a task or location of a checkpoint in the code, but also by the internal timer (state saving is performed at specific time intervals) or the capacitor voltage monitor (saving system state when the capacitor voltage drops below a given threshold). However, existing non-software checkpointing methods are also not suited for the wireless protocol state restoration as the timer would have to be aligned to the protocol state and the voltage-based trigger would break the system state (it would not guarantee state resumption at a specific time). The only viable option for wireless protocol is when the protocol state is at the end of the protocol process, i.e. the only atomic operation that needs to be executed as is.

Finally, the restoration time from the checkpoint must be constant and time-bounded—otherwise, the system will break the real-time requirement of the wireless protocol state machine. This means that, for instance, checkpointing methods that store varying memory regions at each checkpoint moment (like in differential [27] or logging-based [73] checkpoints) are also not feasible, as the larger the checkpoint size—the longer the restoration time.

To make requirements even more stringent, a needed framework would have to separate state tracking of all running applications of the intermittently-powered device, i.e. not only wireless protocol code but also application code, forcing prioritization by skipping application state checkpoint to checkpoint communication stack to keep the connection alive as long as possible. Moreover, the required framework must support interrupts as radio firmware must communicate with peripherals.

Table 2.1 summarizes the most representative frameworks supporting inter-

mittent operation from the perspective of all requirements listed here. We conclude that no existing system fulfills Requirement 1.

Needless to say, techniques that trade-off accuracy of computation for faster completion time in the intermittent computing domain are also not feasible for the reasons described above.

Requirement 2: Abstraction of time and peripherals. Wireless protocol operating on an intermittently-powered device requires a dedicated software abstraction layer to mask time and peripherals state that would normally be disrupted by intermittent operation. Such an abstraction layer should provide undisputed time and peripheral information after resumption from a previous power-off state.

We conjecture that one intermittently-powered device component—on-board timer—must be continuously powered. Once this requirement is fulfilled, while still keeping the timekeeping device battery-free and ultra-small, the intermittently-powered communication support framework can start abstracting time. Furthermore, the time restoration process needs to be accurate and fast enough as not to disrupt the already-established connection event timing. Existing state preservation and restoration frameworks presented in Requirement 1 do not provide an accurate device time, including dedicated timekeeping systems for intermittently-powered devices [26].

2.2 Call for Action

Motivated by all the above requirements, we proceed with the description of the architecture that attains the goal of sustaining wireless protocol communication for battery-free intermittently-powered devices.

Chapter 3

Architecture

We propose a novel system architecture for an intermittently-powered battery-free wireless node schematically presented in Figure 3.1. Our architecture enables the battery-free device to be fully off—instead of relying on only low power or sleep modes of the MCU—and to recover from these power outages whilst meeting the real-time networking or application requirements, thus significantly reducing power consumption.

3.1 Network Structure

We consider a host-end device network structure where the host is tethered or battery-based and runs an unmodified wireless communication protocol of choice. The end node, on the contrary, is completely battery-free and intermittently-powered (by means of any form of energy harvesting). Additionally, we assume the devices do not share a common signal that can be used to synchronize the devices. As with common network protocols, the end device has to announce its presence to a host for a connection to be established. Finally, we assume that both a host and an end node share a common process architecture, including a network process, an operating system process and processes per individual application running as shown in Figure 3.1.

3.2 State Preservation and Restoration

In our system unlike other checkpointing frameworks, we separately checkpoint the memory regions of each individual process. This results in three types of checkpoints targeting: (i) network, (ii) application and (iii) operating system process. Note that as there can be multiple applications running at the same time, multiple application checkpoints may exist. Our framework supports dynamic memory and interrupts as most modern network stacks utilize these features.

Dynamic memory is checkpointed and restored in the operating system checkpoint. The operating system checkpoint represents the minimum that has to be restored at each restoration including the scheduler and other operating system functionality together with the peripheral state. Additionally, depending on the scheduling, the network or application processes can be restored when the device

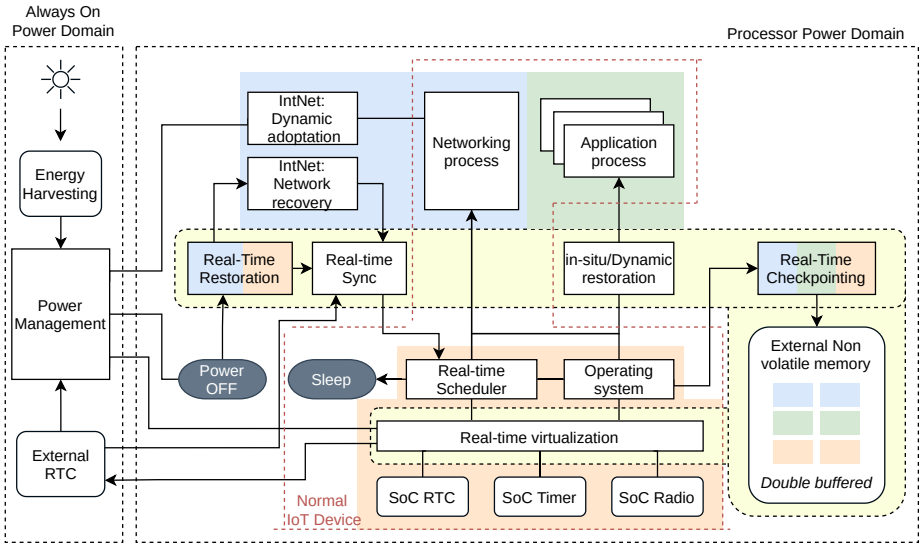


Figure 3.1: **Proposed system architecture of the intermittently-powered battery-free wireless system.**

turns back on. Application and network process checkpoints are only committed together with an operating system process checkpoint. This protects from a situation where memory can be dynamically allocated within a process—in that case checkpointing the application process without the operating system process would lose the dynamic memory. In order to make the system incorruptible, each checkpoint is double-buffered, making restoration possible even if the system loses power during checkpointing.

With the proposed above architecture we only need to restore the checkpoints required for the period the device is on. We differentiate three different power cycles: (i) application only, (ii) network only and (iii) application and network i.e. combined.

The scheduler determines the next cycle during the previous cycle before switching off. During application only cycles, the network does not have to be restored or checkpointed. Similarly, during network only cycles, the system does not require application restoration. Combined cycles occur when the application is scheduled for execution close to a networking event or when the networking event triggers the application to run. To support the latter combined scenario, application dynamic loading is required.

3.2.1 Dynamic Loading

Before the scheduler executes any application process, the process checkpoint is dynamically restored, just right before the execution. Here we differentiate a (i) non-real-time and (ii) real-time application processes. Non-real-time processes checkpoints can be restored by the scheduler just before execution. Real-time processes, however, are restored together with the operating system process when the device restarts after a power failure before the device resumes operation. These include any real-time application processes and the network

process. These processes must be restored before their respective scheduled deadlines, otherwise events can be missed.

3.2.2 Real-Time Virtualization

After the MCU has been switched off, not only volatile memory (process state) is lost but also the state of the peripherals, including the notion of time. Thus upon boot, we synchronize to an external time reference after restoration but before resuming operation. This synchronization point is carefully chosen in advance before the device switches off, allowing sufficient time for restoration of the real-time processes and boot time for the microcontroller. We describe this next wake-up point as $T_{\text{wakeup}} = T_{\text{next\_event}} - T_{\text{boot}} - T_{\text{restore}}$, where T_{boot} is fixed due to the boot time of the MCU and other various fixed delays and T_{restore} is determined dynamically. Each time the device switches on T_{restore} is measured. If T_{restore} exceeds the allotted time, the expected restore time is increased. In this case, the timing for this power cycle is invalid, although non-timing critical processes can still be executed. This mechanism leverages the concept of recovering the connection if a single network event can be missed, a single overshoot does not lead to catastrophic failure, which is further described in Section 3.3.1.

In order to minimize changes to existing network stacks and operating systems, we introduce a new virtualization layer in-between the peripherals and the pre-existing software. During restoration, peripherals are restored and the proposed software layer compensates the timing peripherals for the off-time of the device. This way the network stack or application does not experience any effects of the intermittent operation. When the external synchronization is completed and the offset to the timing peripherals is applied, operation is resumed as normal.

3.2.3 Process Separation

In order to separate the memory of the different processes, we introduce a new linker-based method. Unlike in task-based models, where the programmer has to define each variable associated with a task, we separate the memory of different processes using directories. Hence, a quick separation between application and networking processes can be achieved by placing the source files in different folders, memory declared in the source files present in each of the respective folders is allocated in different non-volatile memory sections and then it is checkpointed in different sections. Our framework provides, (i) application and (ii) network folders. The remainder of the source files that are not placed under one of these folders are assumed to be operating system sources. This method requires minimal intervention by the developer, allowing this technique to be applied to different software stacks.

Figure 3.2 shows the the resulting memory architecture. Our proposed process separation method does not require any modifications to the linker itself. One limitation of our approach is that during execution, the memory from a non-restored process can not be read or written to. However, using operating system features such as messages, interaction between processes can still be achieved. With memory protection units of modern microcontrollers, this can be enforced at minimal overhead.

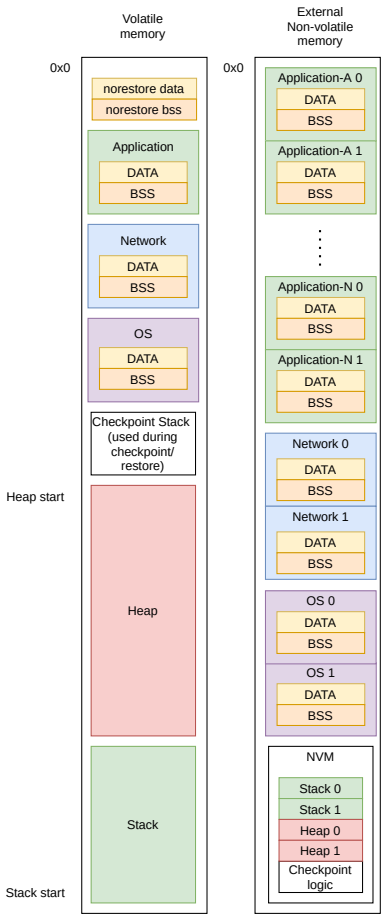


Figure 3.2: **Checkpointing memory architecture.**

We acknowledge that this is not the optimal method of selecting which memory has to be checkpointed for each process. Manually determining which variable should be restored in what process would be more efficient. However, with software stacks of hundredths of thousands of lines of code as shown in Table 3.1, this is not a reasonable solution. We instead aim to provide a generic solution that can be applied to different code stacks with ease.

3.3 Scheduler Adaptation

In order to allow the device to fully turn off, we change the scheduler behavior allowing the device to turn off if the next event is sufficiently far enough in the future. This complements the normal scheduler behavior of going to sleep. If there is insufficient time to fully turn off, sleep mode is still leveraged. The process of turning off can be described as follows. Firstly, the next time the device powers on is determined by the virtualization layer. Next, based on the scheduler queue, the real-time processes that need to be restored during the next cycle are detected. Then, any process that has been executed during the

	Lines of code
Bluetooth (Cordio) [80]	397 200
TCP/IP (LWIP) [39]	88 100
Thread (OpenThread) [46]	250 500

Table 3.1: **Lines of code of different network stacks implementations. Only C/C++ lines of code are counted. Measured using cloc [25].**

current cycle is checkpointed, followed by the operating systems checkpoint, committing all checkpoints of the power cycle. Finally, the device is switched off.

Upon wake-up, the device restores the real-time processes, operating system checkpoint, compensates the virtualization layer for the off time and then synchronizes with the external time source. Then the device resumes operation as normal. If the device is unable to power back up at the desired time after switching off, or if the device switches off due to insufficient energy during operation, recovery is executed.

3.3.1 Connection Recovery

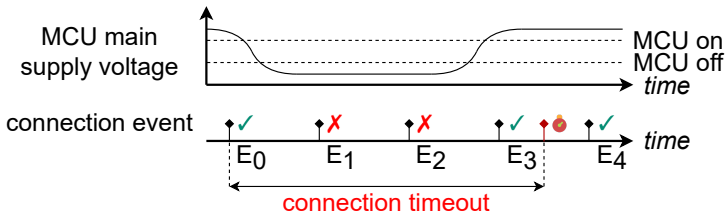


Figure 3.3: **When it is possible to restore from power failure.**

When the device does not turn off by itself but the device turns off unexpectedly due to a power failure, recovery could still be possible. Most wireless networks operate based on timeouts: when a connection has been established but packets are not received for a predefined duration of time, the connection is dropped, which is known as the connection timeout. If enough energy is harvested in-between the power failure and the connection timeout is exceeded, when the device power recovers before resuming operation, a connection recovery can be attempted, as illustrated in Figure 3.3. Based on the connection parameters and the current time, the next network event can be scheduled and the missed event can be skipped.

Chapter 4

Implementation

We proceed with presenting the details of the implementation of the architecture introduced in Chapter 3. As a case study, we select BLE as a wireless protocol of choice and denote its intermittently-powered version as FreeBie.

4.1 FreeBie Hardware

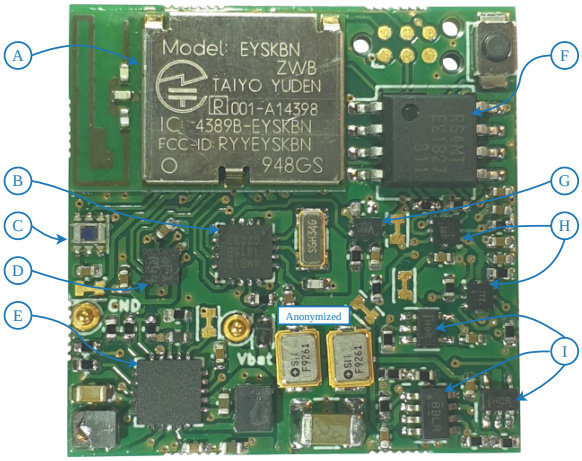


Figure 4.1: **FreeBie hardware fabrication details. The main components are: (A) Wireless module [23] featuring Nordic nRF52840 MCU [106], (B) Ambiq AM1815 RTC [53], (C) Lux meter OPT3004 [56], (D) Accelerometer BMA400 [109], (E) Texas Instruments BQ25570 energy harvester [55], (F) Fujitsu MB85RS4MT 512 KB FRAM [78], (G) External power switch [113], (H) Display flip-flop [59] and buffer [60], and (I) Miscellaneous logic chips (switch buffer [58] and inverter [57]).**

Photo of the designed and fabricated FreeBie node is presented in Figure 4.1. Its design files are open-sourced and available via [2].

4.1.1 Wireless Connectivity and Logic

At the center of FreeBie is a wireless module [23] containing nRF52840 BLE System on Chip (SoC) [106]. To store the state of the system in-between power failures, a fast non-volatile memory Ferroelectric Random Access Memory (FRAM) MB85RS4MT [78] is used and connected to the MCU through an Serial Peripheral Interface (SPI).

4.1.2 Timekeeping

To keep track of time, we chose the AM1815 Real Time Clock (RTC) [53] due to its resolution and low power consumption, creating a similar architecture as proposed in [67]. We did not use a hardware timer like TPL5111 [61], as used by [63] or completely battery-free timekeeping architectures as recently proposed in [26] due to timing accuracy requirements for Bluetooth.

4.1.3 Sensors

FreeBie contains two external sensors: (i) Inter-Integrated Circuit (I2C)-connected accelerometer BMA400 [109] and (ii) a luminosity censor OPT3004 [56]. Both sensors are powered through the MCU itself and are turned on only when required.

4.1.4 Energy Management

The FreeBie mote is separated into two power domains: (i) the system power domain and (ii) an always-on power domain. This design allows the system domain to turn completely off when not required. An external RTC not only keeps track of time in the continuous power domain but also is capable of switching the system power domain on and off.

A Texas Instruments BQ25570 [55] energy harvester is included in FreeBie. Its boost converter boosts the voltage generated by EXL2-1V50 solar panels [76] being the main energy source to FreeBie. [1] The harvested energy is stored in two 7.5 mF capacitors [54]. The system domain is powered by the internal buck converter in the energy harvester configured to generate an 1.8 V supply. The energy harvester switches the buck converter on when the voltage on the storage capacitors reaches 2.6V and switches it off when the capacitor voltage drops below 2.05V.

4.2 FreeBie Software

We have chosen Packetcraft open-source BLE stack [89], which is based on the Arm Mbed Cordio stack [80], as a foundation to implement FreeBie's networking architecture. We remark that we are aware of other open-source implementations of BLE protocol, namely Apache Nimble [5], and Zephyr [127]. Out all of these implementations only Packetcraft's stack supports BLE 5.2.

While most vendors only release their Bluetooth stack in the binary format, the Packetcraft stack released all the source code from the application layer

[1] We have also fabricated a FreeBie version with another energy harvester circuit from Nowi [88]. However, for the experiments, a version with the TI energy harvester was chosen.

to the lowest layer that configures the hardware registers. It also can be built with the standard GCC toolchain [1]. In addition, it is lightweight and can be deployed on Nordic nRF52 series MCUs, rather than requiring a desktop environment or is tightly coupled in a development infrastructure such as MBed and Zephyr. Finally, our preference for Packetcrafts stack was dictated by the native support by the support of our chosen SoC and the lightweight stand-alone implementation.

4.2.1 Real-Time Virtualization

The real-time virtualization presented in Section 3.2.2 is implemented as follows. The external timer we used has a time resolution of only hundredths second in Binary Coded Decimal (BCD) format, while it is sourced by a 32 kHz crystal. In this case, the minimal modulo 10 division of a second is 25 hundredths second, without introducing errors of fractions. Hence, we use 25 hundredths second as the base interval, and the synchronization interval can only be multiples of the base interval. It also means that the absolute time of synchronization can only be multiples of 25 hundredths second.

Originally, the time reference used by the software is the on-chip RTC of the MCU, which resets to zero on every boot. As the abstraction layer, we add an offset value on top of the raw on-chip RTC counter value. The offset value starts from zero at the first boot and it is checkpointed and restored as an internal state of the Operating System (OS). We modify the software so that it always gets the current time from the abstraction layer rather than directly the on-chip RTC, so the software gets a continuous time reference.

The wake-up time and synchronization time are calculated as described in Section 3.2.2. When it is determined by the scheduler that the device should be turned off, the alarm of the external timer is set to the wakeup time and then it turns the MCU off. It turns on the MCU when the alarm is triggered. When the MCU boots up, the on-chip RTC is cleared to zero but not started. After booting is finished, the MCU sets the alarm of the external timer to the synchronization time, then it goes to sleep and waits for the interrupt from the external timer. At the synchronization time, the external timer alarm triggers the interrupt on the MCU, which starts the on-chip RTC immediately. After this, the software has a synchronized time reference.

When there are power failures, it is possible that the calculated wakeup time and the synchronization time are already passed when the MCU turns on. In this case, the MCU will not get the interrupt from the external timer because the alarm is set to the past. If the synchronization is not finished within the expected time, then the MCU should try to recover as described in Section 3.3.1. We note however that the recovery procedure is not yet implemented. In this scenario, the MCU resets everything in this implementation.

4.2.2 Process Separation

Process separation, described in Section 3.2.3, is implemented as follows. Our OS layer is the default one that comes with the Cordio stack, which is called Wireless Software Foundation (WSF). We take out the source files of the WSF and its underlying components, Platform Abstraction Layer (PAL) and the MCU peripheral drivers, and put them in the operating system folder. The

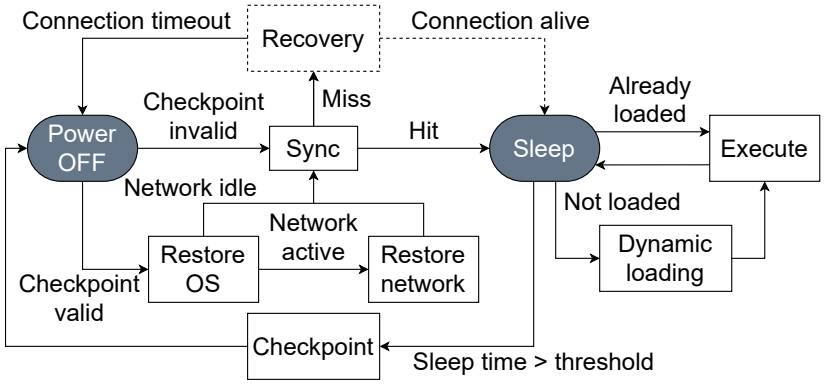


Figure 4.2: **Scheduling.**

other source files of the Cordio stack are put in the network folder. The source files of our top-level applications are put in the application folder. Finally, all the remaining source files are put in the operating system folder. We customize our linker file to allocate memory based on which folder the source file is in, which results in the memory map as shown in Figure 3.2.

4.2.3 Checkpoint

Our non-volatile FRAM is connected to the MCU by the Quad Serial Peripheral Interface (QSPI). We use the customized linker file to mark the starting and the ending address of each memory region.

When restoring, data are read from the corresponding region in FRAM then written to the corresponding region in Random Access Memory (RAM) or registers. For the network stack and each application, there is a boolean flag to indicate whether the region is restored or not, which is cleared to false on every boot and set to true after the restoration of the region. However, all these boolean flags are set to true in the first boot.

When checkpointing, data are read from the corresponding region in RAM or registers then written to the corresponding region in FRAM. A region is checkpointed only when the region is restored. Nevertheless, the OS region is always checkpointed in the end. After that, the boolean flag indicating the checkpoint is valid is set. This boolean flag is false on the first boot.

4.2.4 Scheduling

The implemented scheduling policy is illustrated in Figure 4.2.

The first part of the scheduling code is added in the sleep function of PAL before its original contents. For the first step, we check the time of the next scheduled process, and calculate the time interval from now to the next wakeup time. If the calculated time interval is smaller than two times the sum of restore time and margin time described in Section 3.2.2, then the checkpoint is skipped and the original sleep function is executed. Otherwise, the scheduler checks the relative time between the next network process and the next application process to determine the type of the next power cycle. If the next application process is earlier than the next network process for 250 ms, our base synchronization

interval, then we know for sure that the network will be idle in the next power cycle. Otherwise, we conservatively assume that the network will be active in the next power cycle. Corresponding boolean flags are set accordingly for the restoration procedure in the next power cycle. Finally, the MCU executes the checkpoint and configures the external timer to turn itself off.

The second part of the scheduling code is added at the beginning of the main function. When the MCU is turned on again, it initializes the peripherals and checks if the checkpoint is valid. If the checkpoint is invalid, the MCU initializes everything as if it is its first boot. If the checkpoint is valid, the MCU starts restoring. The OS is always restored, and the network stack is restored only when it is predetermined that the network will be active in this power cycle. After that, the MCU synchronizes the time reference. Finally, the MCU continues execution from the end of the checkpoint code, which is the sleep function of PAL.

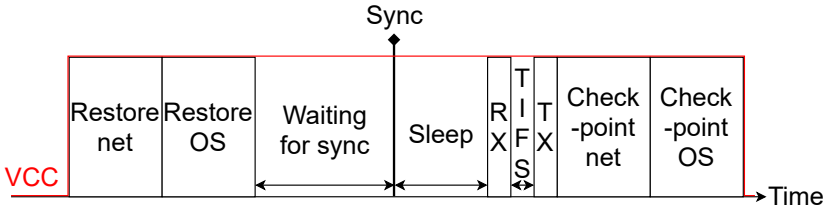


Figure 4.3: **Events in one network-only cycle.**

An example of a typical network-only cycle is illustrated in Figure 4.3. When the MCU is turned on, it finds that the checkpoint is valid and the network is expected to be active. Hence, it restores the OS and the network stack. Once restored, it waits for the synchronization interrupt from the external timer. After synchronization, it goes back to sleep. The BLE radio activities will be triggered accurately, which are receiving, Time Inter Frame Space (TIFS) or the specified time interval between consecutive packets on the same radio channel, and transmitting. When the radio activities are finished, the MCU goes back to sleep but finds out sleep time is greater than the threshold, so it checkpoints the restored memory regions then turns itself off. Note that the receive time, TIFS and the transmit time are relatively very short (at microseconds level) compared to restore and checkpoint time (at milliseconds level). The figure presents the concept but does not reflect the true time length of each event.

Finally, each application process has a boolean flag indicating whether the process is loaded or not, and the flag is reset to false on every boot. At the beginning of every application process, that boolean flag is checked and the corresponding application process is loaded dynamically if the flag is false. After the loading is finished, the flag is set to true. If the flag is already true, then loading is skipped.

4.2.5 Connection Parameters Handling

We make sure the Bluetooth connection parameters are suitable to work intermittently under a tight power budget.

Firstly, we modify the application framework of the Cordio stack to reject

the connection parameter request from the BLE master if the connection parameters are not suitable. The connection interval should be greater than 250 ms, our base synchronization interval. Secondly, we modify the Cordio stack so that it actively requests suitable connection parameters. Once a connection is established, FreeBie starts requesting desired connection parameters. In addition, even though our device as a BLE peripheral cannot reject the update of connection parameters from the BLE central, it will immediately request new connection parameters if the ones updated by the BLE central are not suitable.

4.2.6 BLE Stack Configuration

Firstly, we have changed the system low frequency clock source. Originally, it is the 32 kHz crystal, but the crystal requires typically 250 ms start-up time, which is not acceptable. Another option is its internal 32 kHz Resistor-Capacitor (RC) oscillator, but it requires calibration to be used as an accurate clock on every boot. In the end, we have to use the clock that is synthesized from the system high frequency clock. Its start-up time is short enough and it is also accurate enough. Secondly, to achieve a higher power supply efficiency of the MCU, the on-chip Direct Current (DC)/DC regulator is used rather than the default Low-dropout (LDO) regulator.

4.3 Case Study Implementation

Two case studies are implemented for evaluation.

4.3.1 Smartwatch

We have developed the smartwatch application that implements two standard Bluetooth services, (i) the Current Time Service [11, /current-time-service-1-1] and (ii) the Alert Notification Service [11, /alert-notification-service-1-0], operating on top of FreeBie. The smartwatch works as the Attribute Protocol (ATT) client [12, Section 6.4.1] of those two services. For the BLE central we have developed an Android application working as the ATT server of those two services.

In this application, we have a Liquid-crystal display (LCD) screen connected to the MCU by SPI working as the smartwatch display. On the first boot, the smartwatch starts from the default time. Once a connection is established, the smartwatch tries to find the Current Time Service and the Alert Notification Service on the BLE central. After service discovery completes, the smartwatch enables all the notifications of both services. When the minute of the real-world time increments for the first time after the connection is established, the BLE central sends the current time to the smartwatch, which updates the time immediately and also restarts the one-minute timer to update the watch time so that it can keep aligned with the real-world time. In addition, the BLE central can send an unread email notification to the smartwatch anytime, and the smartwatch will show on the screen an email icon and the number of unread emails according to the received notification.

4.3.2 Firmware Update

First successful demonstrations of over-the-air reprogramming of specific battery-free devices, i.e. Computational Radio Frequency Identifications (CRFIDs) that is backscatter-based node, were reported in [118, 3]. We refer to two recent comprehensive surveys on the topic of over-the-air firmware update and reprogramming [6, 9]. The purpose of this case study is not to create a novel firmware update application but to evaluate FreeBie. Hence, only the very basic functions of a firmware update application are implemented.

In this application, we have reserved two memory regions in FRAM for the firmware. On the first boot, the default firmware is populated from the Flash memory to the first firmware region in FRAM, and then the firmware is executed periodically from there. When updating, the new firmware is written to the inactive firmware region in FRAM. Before the update is finished, the old firmware can still be executed. Once the update is finished, the new firmware will be executed, and the old firmware becomes inactive.

We have created a simple custom BLE service for our firmware update application, and FreeBie works as the ATT server of that service. Once connected, the BLE central should first send the firmware length and then initiate the update. Once initiated, our device starts requesting an 8-byte firmware section by sending the index of the section. Once our device receives the requested firmware section, it writes the section to the corresponding address in the inactive firmware region. If necessary, the same firmware section can be requested again. This process repeats until all the firmware sections are obtained by our device, and finally it sends a notification to the BLE central indicating that the update is completed.

Chapter 5

Evaluation

We now proceed with the evaluation of FreeBie. We start with describing the setup used in gathering data.

5.1 Evaluation Setup

5.1.1 Code Generation and Programming Tools

We use the GNU Arm Embedded Toolchain [77] version 9.3.1 (9-2020-q2-update). In addition, we use CMake [72] (version 3.19.1) to build the project. Finally, we use the educational J-Link [102] to program the MCU, and the version of J-Link software [103] is 6.88a.

5.1.2 Recording Tools

We use a Saleae Logic Pro 16 logic analyzer [99] to record the signals that indicate the events and status of our system. The version of Logic software [98] used with the logic analyzer is 2.3.33. For the BLE communications, we use the nRF Sniffer [104] to capture packets. The nRF Sniffer firmware is programmed to an nRF52 Development Kit [105], and on the PC side we use Wireshark [124] (version 3.4.7) to control the nRF Sniffer.

5.1.3 Environment Setup

During the evaluation, we put FreeBie into a light box, and we use a Philips Hue smart light bulb [93] as the only light source to create the targeted evaluation environment. We use a UNI-T UT383 lux meter [120] to verify the luminance of the created environment. We evaluate our system at three luminance levels: 300 lx, 600 lx and 10 klx.

5.1.4 BLE Central

In the evaluation of the smartwatch application, we developed an Android application, which runs on a Google Pixel 3a smartphone [122] with Android

BLE Central	Connection Interval	Slave Latency	Supervision Timeout	Clock Accuracy
Android	45 ms	0	5 s	500 ppm
Nordic	2 s	1	32 s	20 ppm

Table 5.1: **Initial connection parameters.**

Luminance Level	Connection Interval	Slave Latency	Supervision Timeout
300 lx	3998.75 ms	3	32 s
600 lx	2 s	1	32 s
10 klx	1 s	0	32 s

Table 5.2: **Requested parameters to update.**

11 [45]. For the firmware update application, we used the nRF52840 Development Kit [107] as the BLE central, and the corresponding firmware was developed with the Nordic BLE stack SoftDevice S140 [108].

5.1.5 Advertising and Connection Parameters

The advertising interval of FreeBie is always fixed at 2 s. The initial connection parameters of the two BLE centrals are shown in Table 5.1. For Android, we do not have any Application Programming Interface (API) to configure these parameters so we must use them and these parameters can be vendor-specific. For the Nordic BLE central, we can configure these parameters so we use parameters that are suitable for intermittent operations at the beginning of a connection.

According to the Bluetooth Core Specification [12]: (i) the connection interval shall be a multiple of 1.25 ms in the range 7.5 ms to 4 s and (ii) supervision timeout shall be a multiple of 10 ms in the range 100 ms to 32 s and it shall be larger than $(1 + \text{Slave\_Latency}) \times \text{Connection\_Interval} \times 2$.

The requested parameters from FreeBie vary with the luminance level, as shown in Table 5.2. When it is only 300 lx, the amount of energy that can be harvested is very limited, so we use the maximal allowed parameters to let FreeBie stay off for the longest possible time. As the luminance level increases, FreeBie can harvest more energy so we can decrease the connection interval and the slave latency accordingly.

5.2 Code Size

Code size of the two developed applications is presented in Table 5.3. We expect the time it takes to checkpoint and restore to be proportional to the object code size of the corresponding memory region.

5.3 Case Study: Smartwatch

Figure 5.1a shows the execution of smartwatch at 300 lx. On the top left, we can see the storage capacitor voltage, on the bottom left we can see the sys-

	Smartwatch			Firmware Update		
	data	**bss**	**text**	**data**	**bss**	**text**
No restore	24	468		16	468	
Application	0	2368		0	12	
Network	376	5668		320	5168	
OS	700	10096		548	7164	
Total	1100	18600	240392	884	12812	208893

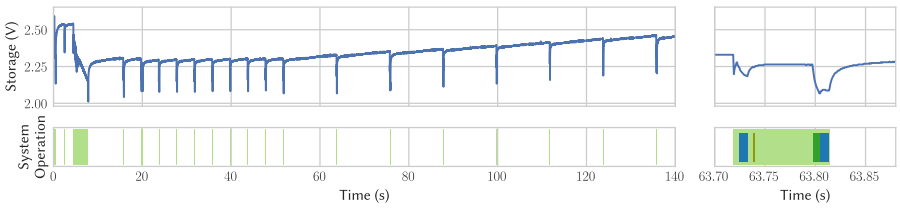
Table 5.3: **Object code size of the applications (in bytes).**

tem operation which shows a light green bar when FreeBie is turned on and operating. The operating time of FreeBie can be relatively so short that it only shows as a very thin light green bar. In this case, a zoom-in view of one power cycle is provided on the right side. In the zoom-in view of system operation on the bottom right, more details can be presented. Restoration and checkpoint are shown as dark blue bars. The thin brown bar indicates synchronization. The BLE radio activities are shown as the dark green bar and the smartwatch application activities as the red bar.

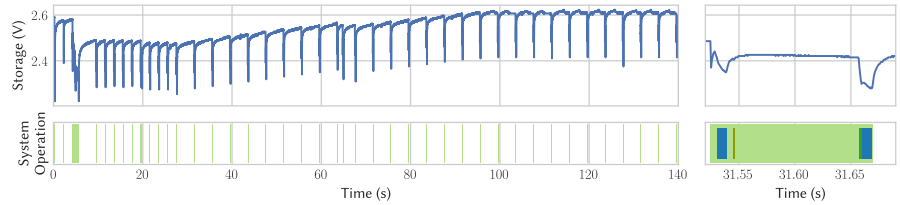
The execution started when the storage capacitor voltage reached 2.6 V. At the beginning of the first operation, the voltage dropped sharply due to the inrush current and the workload of initialization. However, once the first power cycle was finished, the storage voltage could recover to above 2.5 V. After one round of advertising, it was connected with the Android BLE central. Since the connection was established, FreeBie was turned on continuously for a relatively long time, and the voltage also dropped significantly. This is because FreeBie must work with the Android initial connection parameter, which has a very short connection interval so that FreeBie cannot checkpoint or turn off. At the end of this continuous on period, the requested connection parameters were finally applied, and from there FreeBie can checkpoint and turn off. After 60 s, when the Bluetooth services were configured and both BLE peripheral and BLE central started sending empty packets, the slave latency was used and the off time of FreeBie was further increased so that the storage capacitor could be gradually charged up. Note that in this scenario the slave latency being used was 2 but not the requested 3, because Android as the BLE central only granted 2 and FreeBie must work with it. We can observe the continuous increasing trend of the storage voltage, and it could be eventually charged to the maximal level (2.6 V) even though not shown in the figure.

The zoom-in view shows a network only power cycle. At the beginning of the power cycle, we can see the sharp fall of the storage voltage due to the inrush current. The voltage recovered a little, but it dropped again when FreeBie was restoring as indicated by the dark blue bar. Once restored, FreeBie was soon synchronized as indicated by the thin brown bar and the voltage recovered to a stable level. The next significant voltage drop was due to the network activity shown as the dark green bar. Finally, the workload of this power cycle was finished, and FreeBie checkpointed as indicated by the last dark blue bar and turned itself off.

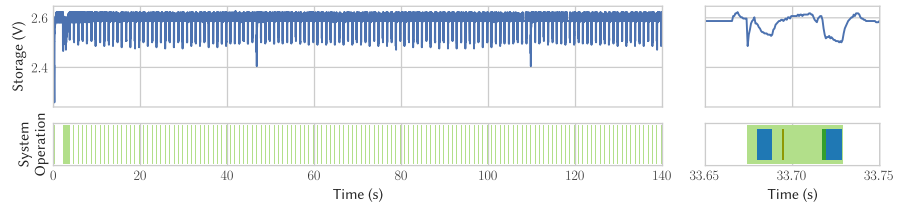
Figure 5.1b shows the execution of smartwatch at 600 lx. We can see that it is very similar to Figure 5.1a. FreeBie could harvest more energy at 600 lx, so it could still run smartwatch very well even though the requested connection



(a) 300 lx.



(b) 600 lx.



(c) 10 klx.

Figure 5.1: **Evaluation of Smartwatch at three diffident luminances.**

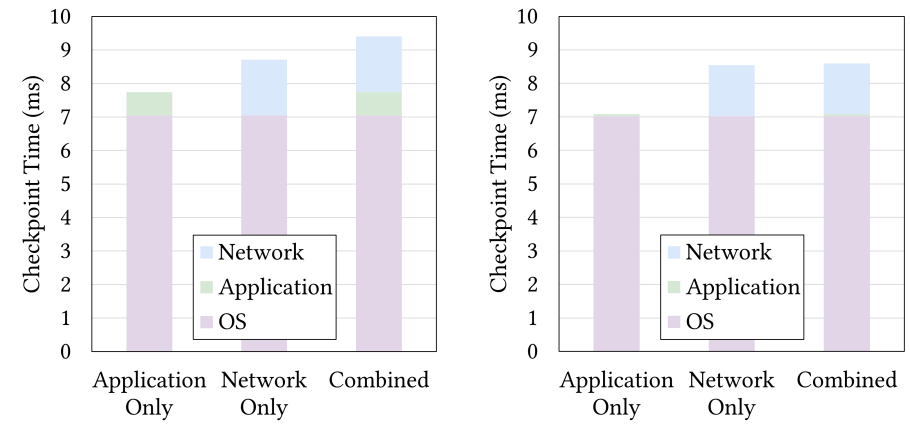
interval is shorter and slave latency is smaller.

Figure 5.1c shows the execution of smartwatch at 10 klx. In this scenario, FreeBie could harvest abundant energy so that the storage voltage never dropped below 2.4 V. However, the zoom-in view is a little different. We can see some spikes on the storage voltage, which are caused by the energy harvester when there is abundant power.

By analyzing the recorded execution trace, we have obtained the checkpoint time of the three types of power cycles in smartwatch, as shown in Figure 5.2a. We can see that the checkpoint time is proportional to the corresponding object code size as reported in Table 5.3.

5.4 Case Study: Firmware Update

Figure 5.3 shows the execution of Firmware Update at 600 lx. On the top, we can see the storage capacitor voltage. In the middle, we can see the system operation which shows a light green bar when FreeBie is turned on and operating. In addition, a light blue bar is shown to indicate the updated firmware. At the bottom, we have the zoom-in view of the three types of power cycles. In Firmware Update, the LCD screen is not used, so we have a few more pins to acquire more details about the events and status of FreeBie during evaluation.



(a) **Smartwatch** (b) **Firmware Update**

Figure 5.2: **Measurement of checkpoint time.**

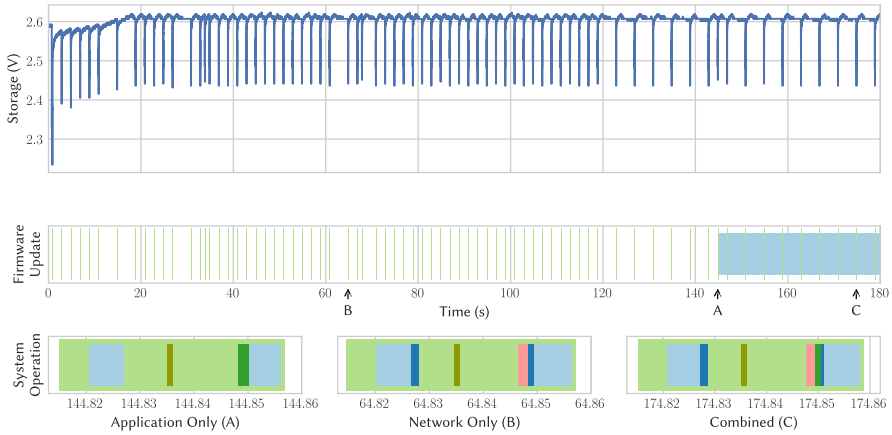


Figure 5.3: **Evaluation of Firmware Update at** 600 lx**.**

Restoration and checkpoint of the OS are shown as light blue bars, and that of the Network as dark blue bars. The brown bar indicates the synchronization. The BLE radio activities are shown as the red bar and the application activities as the dark green bar.

This figure is very similar to Figure 5.1b, its counterpart in smartwatch, but with a major difference. We can see that FreeBie did not have to go through a long continuously on period at the beginning of firmware update, and hence it can keep a higher storage voltage from the start. This is because of the different initial connection parameters by the BLE central, as shown in Table 5.1. If FreeBie can work intermittently from the start of a connection with suitable parameters, it can survive a tighter power budget.

The zoom-in views clearly show the differences among the three types of power cycles. We can see that there are no dark blue bars in an application only cycle, which means the network is not restored or checkpointed. Meanwhile, there are

no dark green bars in a network only cycle, meaning no application activities. A combined cycle has everything. After the network activity ended, indicated as the red bar, FreeBie did not checkpoint but went to sleep because the next event was very near. Finally, after finishing the application activities, FreeBie found that the next event would be far in the future, so it checkpointed all the restored memory and turned itself off.

The checkpoint time of the three types of power cycles in firmware update is shown in Figure 5.2b. We can barely see the Application checkpoint time in the figure, because the application code object size is very small, as reported in Table 5.3, and the corresponding checkpoint time is very short.

Chapter 6

Related Work

Page limits of this paper do not allow to provide a comprehensive literature study of all the domains that intermittently-powered active radio communication spans. Despite this obstruction, we survey the most relevant research results.

6.1 Low-Power Battery-Based Wireless Networking

First attempts in making wireless networks (including BLE) battery-free were preceded by research that focused (and still focusing) on prolonging the lifetime of battery-powered wirelessly-communicating devices. Many techniques are considered including (i) removal of power-consuming components, such as crystals leading to crystal-less radios [19], (ii) connection interval adaptation [70], (iii) duty cycling and its optimization considering various environment and application requirements—in the context of connection-based point-to-point BLE link see e.g. [116, 75], (iv) or even such non-orthodox methods as wireless node optimization for weight [63].[1] Our architecture opens a new avenue in low-power wireless communication—complete system switch-off and restoration for the duration of specific connection interval—that (to the best of our knowledge) has not been addressed by any existing work up to date.

One could still continue with the use of batteries in low-power networks, such as high-density batteries [85], or a hybrid storage made of (lithium-titanate-oxide) battery and capacitor [64]. But no matter how advanced the batteries are, all their limitations listed in Chapter 1 hold.

6.2 Semi Battery-Free Wireless Networking

Another area of research targets merging battery-free and battery-based systems into one with the aim of minimizing the overall node's energy consumption through the use of a backscatter transmission and reception. Such augmentation of IoT with backscatter tags was advocated in [91]. These include [52]

[1] Naturally, the list of techniques listed here is not complete. We refer the reader to numerous surveys in that area, the recent one being [14].

(uplink transmission than can be switched between computational RFID and BLE), [96] (fast switching between all possible combinations of passive and active reception and transmission), [51] (enabling or disabling career generation for transmitter or receiver depending on the remaining battery supply), [40] (RF backscatter (uplink) and Visible Light Communication (VLC) (downlink)). All these techniques, needless to say, need (i) a battery and (ii) a carrier generation source.

A separate class of such devices is nodes based on wake-up radios [94]. Wake-up radios are a preferred choice for quick synchronization of a network of battery-free devices but they still consume power when listening (which increases with receiver sensitivity [94, Figure 12]) and requires the same infrastructure investment as backscatter-based radios, i.e. dedicated wake-up radio transmitter. An example of industrial monitoring battery-free sensors based on proprietary low-power radio wake-up radio technology can be found in [34] (although details of the sensor design are not disclosed).

6.3 Battery-Free Wireless Networking

First attempts to make wireless devices battery-free were based on backscatter transmission. There are numerous of such systems existing, including the ones that modulate specific communication protocols on top of already existing signals such as LoRa [66] or LTE [20]. None of the backscatter-based systems we are aware of enables bi-directional intermittently-powered operation.

An alternative approach for battery-free wireless networking focuses on active radios. In all such systems known to date storage capacitor is dimensioned such that one complete packet transmission—in the majority of the cases: a broadcast—will run out of a single capacitor charge, even if the rest of the computation on the device is performed intermittently. Such systems include intermittently-powered battery-free network for long hard-to-reach location monitoring [4] and self-synchronizing BLE battery-free network for contact tracing [42]. Full implementation of battery-free but not intermittent energy-neutral real-time operating system communication via IEEE 802.15.4 or LoRa is presented in [97]. In a similar spirit deep reinforcement learning for battery-free non-intermittently-powered node to optimize duty cycling of on-board sensors and and sleeping was presented in [38] and [37], respectively.

Yet another conceptual approach is to perform transformations of already existing protocols to have them failure-resilient from the power supply. Such protocol transformation has been presented for a simple message dissemination protocol in multi-hop Wireless Sensor Network (WSN) [95]. The network however assumed a conservative case in which node with a power-off had its all memory flushed and needs to initialize from zero. Mathematical analysis of channel capacity of intermittent communication point-to-point is presented in [68].

A different approach of providing energy to battery-free systems is based on wireless power transfer. A recent example of such system include [87, 62]. Needless to say, while providing instantaneously higher energy than energy harvesting, wireless power transfer is energetically inefficient from the transmitter perspective.

6.4 Battery-Free Bluetooth

As we mentioned already the first battery-free radios were build around the concept of backscatter. The first academic work on such architecture for Bluetooth can be found in [33]. Backscatter-based Bluetooth communication is continuously extended to new setups such as the infrastructure-assisted Bluetooth/WiFi backscatter-based system presented in [129]. Commercial implementations of backscatter-based architecture for Bluetooth include [123].

All of the battery-free backscatter- as well as non-backscatter-based BLE nodes we are aware of do not operate intermittently when considering BLE protocol itself. In each of such nodes one connection-less beacon transmissions can be sent within a single capacitor charge from harvested energy; the recent examples of non-backscatter version of such system are [17, 44, 36, 100, 65, 119] (academic work) and [32, 24] (industry). Another (but less popular) approach for battery-free Bluetooth is based on providing power wirelessly to the BLE nodes [79]. Except for our work no studies on intermittenly-powered Bluetooth are presented beyond general calls for such system. In particular, an advocacy for intermittently-powered BLE where a peripheral could "record a 'snapshot' of the current connection state and then recall it when the device resumes power" is given in [48, Section 5.2.1].

A battery-free BLE node of a similar hardware architecture to ours (i.e. an RTC-driven SoC with external FRAM) has been presented in [111]. The fundamental difference, however, is that the node of [111] unsurprisingly does not allow for state retention of the intermediate state of the BLE protocol (and other applications).

Finally, as a matter of record, we also point to the new generations of ultra-low-power Bluetooth SoC [8] that can further benefit from our state retention architecture.

6.5 Battery-free Systems

Beyond battery-free devices listed in Chapter 2, recently many more devices have been demonstrated, for example on-device machine learning [74] or distributed voice command interpretation [83].

We note a smartwatch with energy harvesting and BLE but with battery supply [82].

6.6 Battery-free Systems Development Tools

Although we have evaluated FreeBie with in-house measurement tools, we note that many tools exist already aiding in evaluating intermittently-powered devices. These include (i) platforms to replay recorded energy harvesting traces [41], (ii) ns-3-based simulation frameworks [16] (of LoRa-based battery-free networks in particular), (iii) planning tools for location of solar-powered battery-free sensors to maximize solar radiation [69], or (iv) debugging tools [21].

6.7 Intermittently-Powered Systems Software Frameworks

Software supporting intermittently-powered operation have already been comparatively presented in Table 2.1.

Chapter 7

Conclusions

This paper presented a new system architecture enabling battery-free operations of a wireless communication protocol. Our architecture enables to sustain an already established wireless connection despite power interruptions. The proposed architecture was used in developing FreeBie: first truly intermittently-powered Bluetooth Low Energy (BLE) system that is not based on backscatter or connection-less (beacon broadcast) transmissions.

Chapter 8

Future Work

8.1 Battery-free Access Point

In our architecture, only the end device is battery-free and intermittently powered. The next research goal is the intermittently-powered access point. The main research challenge would be a synchronization mechanism (a good starting point would be an architecture [42]) and an efficient wake-up schedule for end devices.

Bibliography

[1] GNU compiler collection. https://gcc.gnu.org/git.html, July 2021. Last accessed: Jul. 21, 2021.

[2] FreeBie open source repository. https://github.com/tudssl/XYZ, June 2021. Last accessed: Jun. 13, 2021.

[3] Henko Aantjes, Amjad Y. Majid, Przemysław Pawełczak, Jethro Tan, Aaron Parks, and Joshua R. Smith. Fast downstream to many (computational) RFIDs. In *Proc. INFOCOM*, Atlanta, GA, USA, May 1–4 2017. IEEE. https://doi.org/10.1109/INFOCOM.2017.8056987.

[4] 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. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proc. SenSys*, pages 368–381, Virtual Event, 2020. ACM. https://doi.org/10.1145/3384419.3430722.

[5] Apache Software Foundation. Apache Mynewt operating system Bluetooth stack source code repository. https://github.com/apache/mynewt-nimble, August 2021. Last accessed: Aug. 5, 2021.

[6] Konstantinos Arakadakis, Pavlos Charalampidis, Antonis Makrogiannakis, and Alexandros Fragkiadakis. Firmware over-the-air programming techniques for IoT networks - a survey, September 2020. https://arxiv.org/pdf/2009.02260.pdf.

[7] Rauf Arif. With an economic potential of $11 trillion, Internet Of Things is here to revolutionize global economy. Forbes, https://www.forbes.com/sites/raufarif/2021/06/05/with-an-economic-potential-of-11-trillion-internet-of-things-is-here-to-revolutionize-global-economy/, June 2021. Last accessed: Jul. 6, 2021.

[8] Atmosic Technologies. ATM2201/ATM2221/ATM2202 extreme low power Bluetooth 5.0 SoC. https://atmosic.com/technology/, April 2021. Last accessed: Jul. 31, 2021.

[9] Jan Bauwens, Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, and Eli De Poorter. Over-the-air software updates in the Internet of Things: An overview of key principles. *IEEE Commun. Mag.*, 58(2):35–41, February 2020. https://doi.org/10.1109/MCOM.001.1900125.

[10] Bluetooth Special Interest Group, Inc. 2021 market update. https://www.bluetooth.com/wp-content/uploads/2021/01/2021-Bluetooth_Market_Update.pdf, January 2021. Last accessed: Jul. 27, 2021.

[11] Bluetooth Special Interest Group, Inc. Bluetooth specifications list. https://www.bluetooth.com/specifications/specs/, July 2021. Last accessed: Aug. 8, 2021.

[12] Bluetooth Special Interest Group, Inc. Core specification 5.3. https://www.bluetooth.com/specifications/specs/core-specification/, July 2021. Last accessed: Jul. 31, 2021.

[13] Rodney Brooks. The battery revolution is just getting started. IEEE Spectrum, https://spectrum.ieee.org/energy/batteries-storage/the-battery-revolution-is-just-getting-started, July 2021.

[14] Alison Burdett. Ultra-low-power wireless systems: Energy-efficient radios for the internet of things. *IEEE Solid-State Circuits Mag.*, 7(2):18–28, Spring 2015. https://doi.org/10.1109/MSSC.2015.2417095.

[15] Bradford Campbell, Joshua Adkins, and Prabal Dutta. Cinamin: A perpetual and nearly invisible BLE beacon. In *Proc. NextMote Workshop (EWSN 2016 Workshop)*, Graz, Austria, February 15 2016. https://www.ewsn.org/file-repository/ewsn2016/331_332_campbell.pdf.

[16] Martina Capuzzo, Carmen Delgado, Jeroen Famaey, and Andrea Zanella. An ns-3 implementation of a battery-less node for energy-harvesting Internet of Things. In *Proc. Workshop on ns-3*, Virtual Event, June 21 2021. https://arxiv.org/pdf/2103.14596.pdf.

[17] Carlo Signer. Batteryless Bluetooth low energy communication. Bachelor's thesis, ETHZ, Switzerland, February 2017. https://pub.tik.ee.ethz.ch/students/2017-FS/BA-2017-03.pdf.

[18] Ricardo C. Carrano, Diego Passos, Luiz C. S. Magalhães, and Célio V. N. Albuquerque. Survey and taxonomy of duty cycling mechanisms in wireless sensor networks. *IEEE Commun. Surv. Tutorials*, 16(1):181–194, First Quarter 2014. https://doi.org/10.1109/SURV.2013.052213.00116.

[19] Xing Chen, Abdullah Alghaihab, Yao Shi, Daniel S. Truesdell, Benton H. Calhoun, and David D. Wentzloff. A crystal-less BLE transmitter with clock recovery from GFSK-modulated BLE packets. *IEEE J. Solid-State Circuits*, 56(7):1963–1974, July 2021. https://doi.org/10.1109/JSSC.2020.3046610.

[20] Zicheng Chi, Xin Liua, Wei Wang, Yao Yao, and Ting Zhu. Leveraging ambient LTE traffic for ubiquitous passive communication. In *Proc. SIGCOMM*, pages 172–185, Virtual Event, 2020. ACM. https://doi.org/10.1145/3387514.3405861.

[21] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson Sample. An energy-interference-free hardware/software debugger for intermittent energy-harvesting systems. In *Proc. ASPLOS*, pages 577–589, Atlanta, GA, USA, 2016. ACM. https://doi.org/10.1145/2980024.2872409.

[22] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proc. ASPLOS*, pages 767–781, Williamsburg, VA, USA, 2018. ACM. https://doi.org/10.1145/3173162.3173210.

[23] Taiyo Yuden Corp. EYSKBNZWB Wireless Module. https://www.yuden.co.jp/eu/product/category/module/bluetooth/EYSKBNZWB.html, 2021. Last accessed: Jun. 13, 2021.

[24] Cypress Semiconductor Corp. CYALKIT-E02 solar-powered BLE sensor beacon reference design kit. https://www.cypress.com/documentation/development-kitsboards/cyalkit-e02-solar-powered-ble-sensor-beacon-reference-design, June 2020. Last accessed: Aug. 4, 2021.

[25] Al Danial. cloc - Count Lines of Code repository. https://github.com/AlDanial/cloc, May 2021. Last accessed: Aug. 10, 2021.

[26] Jasper de Winkel, Carlo Delle Donne, Kasım Sinan Yıldırım, Przemysław Pawełczak, and Josiah Hester. Reliable timekeeping for intermittent computing. In *Proc. ASPLOS*, pages 53–67, Lausanne, Switzerland, March 16–20 2020. ACM. https://doi.org/10.1145/3373376.3378464.

[27] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. Battery-free game boy. *ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(3):111:1–111:34, September 2020. https://doi.org/10.1145/3411839.

[28] Maurizio Di Paolo Emilio and Roy Anirban. Macro environmental effect of micro energy harvesting. https://www.powerelectronicsnews.com/macro-environmental-effect-of-micro-energy-harvesting/, March 2021. Last accessed: Jul. 27, 2021.

[29] Can Dincer, Richard Bruch, Estefanía Costa-Rama, Maria Teresa Fernández-Abedul, Arben Merkoçi, Andreas Manz, Gerald Anton Urban, and Firat Güder. Disposable sensors in diagnostics, food, and environmental monitoring. *Adv. Mater.*, 31(30), July 2019.

[30] Yi Ding, Ling Liu, Yu Yang, Yunhuai Liu, Desheng Zhang, and Tian He. From conception to retirement: a lifetime story of a 3-year-old wireless beacon system in the wild. In *Proc. NSDI*, pages 859–872, Virtual Event, April 12–14 2021. USENIX. https://www.usenix.org/system/files/nsdi21spring-ding.pdf.

[31] Kristina Edström (Executive Publisher). Horizon 2020 EU program battery 2030+: Inventing the sustainable batteries of the future: Research needs and future actions. https://battery2030.eu/digitalAssets/861/c_861350-l_1-k_roadmap-27-march.pdf, March 2020. Last accessed: Jul. 7, 2021.

[32] EnOcean GmbH. STM 550B multisensor module (BLE) with NFC interface. https://www.enocean.com/en/products/enocean_modules_24ghz_ble/stm-550b-multisensor-module/, September 2020. Last accessed: Aug. 4, 2021.

[33] Joshua F. Ensworth and Matthew S. Reynolds. Every smart phone is a backscatter reader: Modulated backscatter compatibility with Bluetooth 4.0 low energy (BLE) devices. In *Proc. RFID*, pages 78–85, San Diego, CA, USA, April 15–17 2015. IEEE. https://doi.org/10.1109/RFID.2015.7113076.

[34] Everactive. Always-on intelligence. https://everactive.com/batteryless-technology/, August 2021. Last accessed: Aug. 5, 2021.

[35] Duarte Fernandes, André G. Ferreira, Reza Abrishambaf, José Mendes, and Jorge Cabral. Survey and taxonomy of transmissions power control mechanisms for wireless body area networks. *IEEE Commun. Surv. Tutorials*, 20(2):1292–1328, Second Quarter 2018. https://doi.org/10.1109/COMST.2017.2782666.

[36] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. Pible: Battery-free mote for perpetual indoor BLE applications. In *Proc. BuildSys*, pages 168–171, Shenzen, China, November 7–8 2018. ACM. https://doi.org/10.1145/3276774.3282822.

[37] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K. Gupta. ACES: Automatic configuration of energy harvesting sensors with reinforcement learning. *ACM Trans. Sens. Netw.*, 16(4):36:1–36:31, October 2020. https://doi.org/10.1145/3404191.

[38] Francesco Fraternali, Bharathan Balaji, Dhiman Sengupta, Dezhi Hong, and Rajesh K. Gupta. Ember: Energy management of batteryless event detection sensors with deep reinforcement learning. In *Proc. SenSys*, pages 503–516, Virtual Event, November 16–19 2020. ACM. https://doi.org/10.1145/3384419.3430734.

[39] Free Software Foundation, Inc. lwIP - A Lightweight TCP/IP stack website. https://savannah.nongnu.org/projects/lwip, June 2021. Last accessed: Aug. 10, 2021.

[40] Ander Galisteo, Ambuj Varshney, and Domenico Giustiniano. Two to tango: Hybrid light and backscatter networks for next billion devices. In *Proc. MobiSys*, pages 80–93, Toronto, ON, Canada, June 15–19 2020. ACM. https://doi.org/10.1145/3386901.3388918.

[41] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. Shepherd: a portable testbed for the batteryless IoT. In *Proc. SenSys*, pages 83–95, New York, NY, USA, November 10–13 2019. ACM. https://doi.org/10.1145/3356250.3360042.

[42] Kai Geissdoerfer and Marco Zimmerling. Bootstrapping battery-free wireless networks: Efficient neighbor discovery and synchronization in the face of intermittency. In *Proc. NSDI*, pages 439–455, Virtual Event, April 12–14 2021. USENIX. https://www.usenix.org/system/files/nsdi21-geissdoerfer.pdf.

[43] Shyamnath Gollakota, Matthew S. Reynolds, Joshua R. Smith, and David J. Wetherall. The emergence of RF-powered computing. *IEEE Computer*, 47(1), January 2014. https://doi.org/10.1109/MC.2013.404.

[44] Andres Gomez. Demo abstract: On-demand communication with the batteryless mirocard. In *Proc. SenSys*, pages 629–630, Virtual Event, November 16–19 2020. ACM. https://doi.org/10.1145/3384419.3430440.

[45] Google LLC. Android 11 mobile operating system (with security update: July 5, 2021). https://www.android.com/android-11/. Last accessed: Aug. 8, 2021.

[46] Google LLC. Openthread. https://github.com/openthread/openthread, June 2021. Last accessed: Aug. 10, 2021.

[47] Mike Hayes, Giorgos Fagas, Julie Donnelly, Raphaël Salot, Guillaume Savelli, Peter Spies, Gerd vom Boegel, Mario Konijnenburg, David Stenzel, Aldo Romani, Claudio Gerbaldi, Francesco Cottone, and Alex Weddell. Research infrastructure to power the Internet of Things. https://www.tyndall.ie/contentfiles/EnABLES_Research_Infrastructure_Position_Paper.pdf, February 2021. Last accessed: Aug. 3, 2021.

[48] Steven Lain Hearndon. An analysis of Bluetooth Low Energy in the context of intermittently powered devices. Master's thesis, Clemson University, SC, USA, December 2016. https://tigerprints.clemson.edu/all_theses/2537/.

[49] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proc. SenSys*, pages 5–16, Seoul, South Korea, 2015. ACM. https://doi.org/10.1145/2809695.2809707.

[50] Josiah Hester and Jacob Sorber. The future of sensing is batteryless, intermittent, and awesome. In *Proc. SenSys*, pages 21:1–21:6, Delft, The Netherlands, 2017. ACM. https://doi.org/10.1145/3131672.3131699.

[51] Pan Hu, Pengyu Zhang, Mohammad Rostami, and Deepak Ganesan. Braidio: An integrated active-passive radio for mobile devices with asymmetric energy budgets. In *Proc. SIGCOMM*, pages 384–397, Florianopolis, Brazil, August 22–26 2016. ACM. https://doi.org/10.1145/2934872.2934902.

[52] Ivar in 't Veen, Qingzhi Liu, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. BLISP: Enhancing backscatter radio with active radio for computational RFIDs. In *Proc. RFID*, Orlando, FL, USA, May 3–5 2016. IEEE. https://doi.org/10.1109/RFID.2016.7488010.

[53] Ambiq Micro Inc. AM1815. https://ambiq.com/artasie-am1815/, 2021. Last accessed: Aug. 20, 2021.

[54] Seiko Instruments Inc. CPX3225A752D. https://www.sii.co.jp/en/me/datasheets/chip-capacitor/cpx3225a752d/, 2021. Last accessed: Aug. 20, 2021.

[55] Texas Instruments Inc. BQ25570. https://www.ti.com/product/BQ25570, 2021. Last accessed: Aug. 20, 2021.

[56] Texas Instruments Inc. OPT3004. https://www.ti.com/product/OPT3004, 2021. Last accessed: Aug. 20, 2021.

[57] Texas Instruments Inc. SN74AUP1G04. https://www.ti.com/product/SN74AUP1G04, 2021. Last accessed: Aug. 20, 2021.

[58] Texas Instruments Inc. SN74AUP1G34. https://www.ti.com/product/SN74AUP1G34, 2021. Last accessed: Aug. 20, 2021.

[59] Texas Instruments Inc. SN74AUP2G79. https://www.ti.com/product/SN74AUP2G79, 2021. Last accessed: Aug. 20, 2021.

[60] Texas Instruments Inc. SN74AUP3G34. https://www.ti.com/product/SN74AUP3G34, 2021. Last accessed: Aug. 20, 2021.

[61] Texas Instruments Inc. TPL5111. https://www.ti.com/product/TPL5111, 2021. Last accessed: Aug. 20, 2021.

[62] Vikram Iyer, Elyas Bayati, Rajalakshmi Nandakumar, Arka Majumdar, and Shyam Gollakota. Charging a smartphone across a room using lasers. *ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4):143:1–143:21, December 2017. https://doi.org/10.1145/3161163.

[63] Vikram Iyer, Maruchi Kim, Shirley Xue, Anran Wang, and Shyamnath Gollakota. Airdropping sensor networks from drones and insects. In *Proc. MobiCom*, pages 813–826. ACM, September 21–25 2020. https://doi.org/10.1145/3372224.3419981.

[64] Neal Jackson, Joshua Adkins, and Prabal Dutta. Capacity over capacitance for reliable energy harvesting sensors. In *Proc. IPSN*, pages 193–204, Montreal, QC, Canada, April 16–18 2019. ACM. https://doi.org/10.1145/3302506.3310400.

[65] Kang Eun Jeon, James She, Jason Xue, Sang-Ha Kim, and Soochang Park. LuXbeacon—a batteryless beacon for green IoT: Design, modeling, and field tests. *IEEE Internet Things J.*, 6(3):5001–5012, June 2019. https://doi.org/10.1109/JIOT.2019.2894798.

[66] Mohamad Katanbaf, Anthony Weinand, and Vamsi Talla. Simplifying backscatter deployment: Full-duplex LoRa backscatter. In *Proc. NSDI*, pages 955–972, Virtual Event, April 12–14 2021. USENIX. https://www.usenix.org/system/files/nsdi21spring-katanbaf.pdf.

[67] Giannis Kazdaridis, Nikos Sidiropoulos, Ioannis Zografopoulos, Polychronis Symeonidis, and Thanasis Korakis. Nano-things: Pushing sleep current consumption to the limits in IoT platforms. In *Proc. IoT*, pages 1–8, Malmö, Sweden, October 6–9 2020. ACM. https://doi.org/10.1145/3410992.3410998.

[68] Mostafa Khoshnevisan and J. Nicholas Laneman. Intermittent communication. *IEEE Trans. Inf. Theory*, 63(7):4089–4102, July 2017. https://doi.org/10.1109/TIT.2017.2692239.

[69] Daeyong Kim, Junick Ahn, Jun Shin, and Hojung Cha. Ray tracing-based light energy prediction for indoor batteryless sensors. *ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(1):17:1–17:27, March 2021. https://doi.org/10.1145/3448086.

[70] Philipp Kindt, Daniel Yunge, Mathias Gopp, and Samarjit Chakraborty. Adaptive online power-management for Bluetooth Low Energy. In *Proc. INFOCOM*, pages 2695–2703, Hong Kong, China, 26 Apr.–1 May 2015. IEEE. https://doi.org/10.1109/INFOCOM.2015.7218661.

[71] Philipp H. Kindt, Daniel Yunge, Robert Diemer, and Samarjit Chakraborty. Energy modeling for the Bluetooth Low Energy protocol. *ACM Trans. Embed. Comput. Syst.*, 19(2):13:1–13:32, March 2020. https://dl.acm.org/doi/10.1145/3379339.

[72] Kitware. CMake. https://cmake.org/, 2021. Last accessed: Aug. 19, 2021.

[73] Vito Kortbeek, Kasım Sinan Yıldırım, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. In *Proc. ASPLOS*, pages 85–99, Lausanne, Switzerland, March 16–20 2020. ACM. https://doi.org/10.1145/3373376.3378476.

[74] Seulki Lee, Bashima Islam, Yubo Luo, and Shahriar Nirjon. Intermittent learning: On-device machine learning on intermittently powered system. *ACM Interact. Mob. Wearable Ubiquitous Technol.*, 3(4):141:1–141:30, December 2019. https://doi.org/10.1145/3369837.

[75] Andreina Liendo, Dominique Morche, Roberto Guizzetti, and Franck Rousseau. Efficient Bluetooth low energy operation for low duty cycle applications. In *Proc. ICC*, pages 1–7, Kansas City, MO, USA, May 20–24 2018. IEEE. https://doi.org/10.1109/ICC.2018.8423011.

[76] Lightricity. EXL2-1V50. https://lightricity.co.uk/excelllight-exl2-1v50-1, 2021. Last accessed: Aug. 20, 2021.

[77] Arm Limited. GNU arm embedded toolchain. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads, 2020. Last accessed: Aug. 19, 2021.

[78] Fujitsu Semiconductor Limited. MB85RS4MT 4 MB FRAM Memory with SPI Interface. https://www.fujitsu.com/global/documents/products/devices/semiconductor/fram/lineup/MB85RS4MT-DS501-00053-1v0-E.pdf, 2018. Last accessed: Jun. 13, 2021.

[79] Qingzhi Liu, Wieger IJntema, Anass Drif, Przemysław Pawełczak, Marco Zuniga, and Kasım Sinan Yıldırım. Perpetual Bluetooth communications for the IoT. *IEEE Sens. J.*, 21(1):829–837, January 2021. https://doi.org/10.1109/JSEN.2020.3012814.

[80] Arm Ltd. Mbed Cordio Bluetooth Low Energy solution official website. https://os.mbed.com/docs/mbed-cordio/, August 2021. Last accessed: Aug. 5, 2021.

[81] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent Computing: Challenges and Opportunities. In *Proc. SNAPL*, pages 8:1–8:14, Alisomar, CA, USA, May 7–10 2017. https://drops.dagstuhl.de/opus/volltexte/2017/7131/pdf/LIPIcs-SNAPL-2017-8.pdf.

[82] Michele Magno, Xiaying Wang, Manuel Eggimann, Lukas Cavigelli, and Luca Benini. InfiniWolf: Energy efficient smart bracelet for edge computing with dual source energy harvesting. In *Proc. DATE*, Grenoble, France, March 9–13 2020. IEEE. https://doi.org/10.23919/DATE48585.2020.9116218.

[83] Amjad Y. Majid, Patrick Schilder, and Koen Langendoen. Continuous sensing on intermittent power. In *Proc. IPSN*, pages 181–192, Sydney, NSW, Australia, April 21–24 2020. IEEE. https://doi.org/10.1109/IPSN48710.2020.00-36.

[84] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasım Sinan Yıldırım, Brandon Lucia, and Przemysław Pawełczak. Dynamic task-based intermittent execution for energy-harvesting devices. *ACM Trans. Sens. Netw.*, 16(1):5:1–5:24, February 2020. https://doi.org/10.1145/3360285.

[85] Millibatt. Nimbus product line. https://www.millibatt.com/product, August 2021. Last accessed: Aug. 4, 2021.

[86] Neeru Mittal, Alazne Ojanguren, Markus Niederberger, and Erlantz Lizundia. Degradation behavior, biocompatibility, electrochemical performance, and circularity potential of transient batteries. *Advanced Science*, 8(2004814), May 2021. https://doi.org/10.1002/ADVS.202004814.

[87] Noor Mohammed, Rui Wang, Robert W. Jackson, Yeonsik Noh, Jeremy Gummeson, and Sunghoon Ivan Lee. ShaZam: Charge-free wearable devices via intra-body power transfer from everyday objects. *ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(2):75:1–75:25, June 2021. https://doi.org/10.1145/3463505.

[88] NOWI. NH2D0245 Energy Harvesting PMIC. https://www.nowi-energy.com/products-nh2/, 2021. Last accessed: Jun. 13, 2021.

[89] Packetcraft, Inc. Packetcraft protocol software source code repository. https://github.com/packetcraft-inc/stacks, August 2021. Last accessed: Aug. 5, 2021.

[90] Arielle Pardes. The WIRED guide to the Internet of Things. WIRED, https://www.wired.com/story/wired-guide-internet-of-things/, September 2020. Last accessed: Jul. 6, 2021.

[91] Carlos Perez-Penichet, Fredrick Hermans, Ambuj Varshney, and Thiemo Voigt. Augmenting IoT networks with backscatter-enabled passive sensor tags. In *Proc. HotWireless*, pages 23–27, New York City, NY, USA, October 3–7 2016. ACM. https://doi.org/10.1145/2980115.2980132.

[92] Matthai Philipose, Joshua R. Smith, Bing Jiang, Alexander Mamishev, Sumit Roy, and Kishor Sundara-Rajan. Battery-free wireless identification and sensing. *IEEE Pervasive Comput.*, 4(1):37–45, Jan.–Mar. 2005. https://doi.org/10.1109/MPRV.2005.7.

[93] Philips. Philips hue smart light bulbs. https://www.philips-hue.com/en-us/products/smart-light-bulbs, 2021. Last accessed: Aug. 19, 2021.

[94] Rajeev Piyare, Amy L. Murphy, Csaba Kiraly, Pietro Tosato, and Davide Brunell. Ultra low power wake-up radios: A hardware and networking survey. *IEEE Commun. Surv. Tutorials*, 19(4):2117–2157, Fourth Quarter 2017. https://doi.org/10.1109/COMST.2017.2728092.

[95] David Richardson, Arshad Jhumka, and Luca Mottola. Protocol transformation for transiently powered wireless sensor networks. In *Proc. SAC*, pages 1112–1121, Virtual Event, March 22–26 2021. ACM. https://doi.org/10.1145/3412841.3441985.

[96] Mohammad Rostami, Jeremy Gummeson, Ali Kiaghadi, and Deepak Ganesan. Polymorphic radios: A new design paradigm for ultra-low power communication. In *Proc. SIGCOMM*, pages 446–460, Budapest, Hungary, August 20–25 2018. ACM. https://doi.org/10.1145/3230543.3230571.

[97] Michel Rotteluthner, Thomas C. Schmidt, and Matthias Wählisch. Sense your power: The ECO approach to energy awareness for IoT devices. *ACM Trans. Embed. Comput. Syst.*, 20(3):24:1–14:23, March 2021. https://dl.acm.org/doi/10.1145/3441643.

[98] Saleae. Logic analyzer software from saleae. https://www.saleae.com/downloads/, 2021. Last accessed: Aug. 19, 2021.

[99] Saleae. Saleae logic pro 16 usb logic analyzer. http://downloads.saleae.com/specs/Logic+Pro+16+Product+Fact+Sheet.pdf, 2021. Last accessed: Aug. 19, 2021.

[100] Nurani Saoda and Bradford Campbell. No batteries needed: Providing physical context with energy-harvesting beacons. In *Proc. ENSSys*, pages 15–21. ACM, 2019. https://doi.org/10.1145/3362053.3363489.

[101] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. The computing landscape of the 21st century. In *Proc. HotMobile*, pages 45–50, Santa Cruz, CA, USA, 2019. ACM. https://doi.org/10.1145/3301293.3302357.

[102] SEGGER. J-Link educational debug probe. https://www.segger.com/products/debug-probes/j-link/models/j-link-edu/, 2021. Last accessed: Aug. 19, 2021.

[103] SEGGER. J-Link software and documentation pack. https://www.segger.com/downloads/jlink/, 2021. Last accessed: Aug. 19, 2021.

[104] Nordic Semiconductor. nRF sniffer for bluetooth le. https://www.nordicsemi.com/Products/Development-tools/nRF-Sniffer-for-Bluetooth-LE, 2021. Last accessed: Aug. 19, 2021.

[105] Nordic Semiconductor. nRF52 dk. https://www.nordicsemi.com/Products/Development-hardware/nRF52-DK, 2021. Last accessed: Aug. 19, 2021.

[106] Nordic Semiconductor. nRF52840. https://www.nordicsemi.com/Products/nRF52840, 2021. Last accessed: Aug. 20, 2021.

[107] Nordic Semiconductor. nRF52840 dk. https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK, 2021. Last accessed: Aug. 19, 2021.

[108] Nordic Semiconductor. SoftDevice s140. https://www.nordicsemi.com/Products/Development-software/s140, 2021. Last accessed: Aug. 20, 2021.

[109] Bosch Sensortec. BMA400. https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/bma400/, 2021. Last accessed: Aug. 20, 2021.

[110] Esther Shein. A battery-free Internet of Things. *Commun. ACM*, 64(7):16–18, 2021. https://doi.org/10.1145/3464937.

[111] Lukas Sigrist, Rehan Ahmed, Andres Gomez, and Lothar Thiele. Harvesting-aware optimal communication scheme for infrastructure-less sensing. *ACM Trans. Internet Things*, 1(4):22:1–22:26, October 2020. https://doi.org/10.1145/3395928.

[112] Silicon Laboratories Inc. BLE112 data sheet version 1.8. https://www.silabs.com/documents/public/data-sheets/BLE112-DataSheet.pdf, December 2020. Last accessed: Aug. 19, 2021.

[113] Vishay Siliconix. SIP32432. https://www.vishay.com/docs/66597/sip32431.pdf, 2021. Last accessed: Aug. 20, 2021.

[114] Patrice Simon, Yury Gogotsi, and Bruce Dunn. Where do batteries end and supercapacitors begin? *Science*, 343(6176):1210–1211, March 2014. https://doi.org/10.1126/science.1249625.

[115] Sivert T. Sliper, Oktay Cetinkaya, Alex S. Weddell, Bashir Al-Hashimi, and Geoff V. Merrett. Energy-driven computing. *Phil. Trans. R. Soc. A.*, 378(2164), February 2020. https://doi.org/10.1098/rsta.2019.0158.

[116] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer. BLEach: Exploiting the full potential of IPv6 over BLE in constrained embedded IoT devices. In *Proc. SenSys*, pages 1–14, Delft, The Netherlands, November 6–8 2017. ACM. https://doi.org/10.1145/3131672.3131687.

[117] Tucker Stuart, Le Caia, Alex Burton, and Philipp Gutruf. Wireless and battery-free platforms for collection of biosignals. *Biosensors and Bioelectronics*, 178, April 2021. https://doi.org/10.1016/j.bios.2021.113007.

[118] Jethro Tan, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. Wisent: Robust downstream communication and storage for computational RFIDs. In *Proc. INFOCOM*, San Francisco, CA, USA, 2016. IEEE. https://doi.org/10.1109/INFOCOM.2016.7524574.

[119] Pietro Tedeschi, Kang Eun Jeon, James She, Simon Wong, Spiridon Bakiras, and Roberto Di Pietro. Privacy-preserving and sustainable contact tracing using batteryless BLE beacons, March 2021. https://arxiv.org/pdf/2103.06221.pdf.

[120] UNI-T. UT383 mini light meter. https://www.uni-trend.com/html/product/Environmental/Environmental_Tester/Mini/UT383.html, 2020. Last accessed: Aug. 19, 2021.

[121] University of Washington, Seattle, WA, USA. Wireless identification and sensing platform GitHub page. https://github.com/wisp, November 2010. Last accessed: Jul. 20, 2021.

[122] Wikipedia contributors. Google pixel 3a—Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Pixel_3a. Last accessed: Aug. 8, 2021.

[123] Wiliot. Wiliot Battery Free IoT Pixel BLE Tags Website. https://www.wiliot.com/product/iot-pixel, 2021. Last accessed: Jul. 22, 2021.

[124] Wireshark. Wireshark. https://www.wireshark.org/, 2021. Last accessed: Aug. 19, 2021.

[125] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Pawełczak, and Josiah Hester. InK: Reactive kernel for tiny batteryless sensors. In *Proc. SenSys*, pages 41–53, Shenzhen, China, November 4–7 2018. ACM. https://doi.org/10.1145/3274783.3274837.

[126] G. Pascal Zachary. The search for a better battery. IEEE Spectrum, https://spectrum.ieee.org/at-work/innovation/the-search-for-a-better-battery, April 2016. Last accessed: Jul. 7, 2021.

[127] Zephyr Project. Zephyr real-time operating system source code repository. https://github.com/zephyrproject-rtos/zephyr, August 2021. Last accessed: Aug. 5, 2021.

[128] Maolin Zhang, Si Chen, Jia Zhao, and Wei Gong. Commodity-level BLE backscatter. In *Proc. MobiSys*, pages 402–414, Virtual Event, Jun. 24—Jul. 2 2021. ACM. https://dl.acm.org/doi/pdf/10.1145/3458864.3466865.

[129] Pengyu Zhang, Mohammad Rostami, Pan Hu, and Deepak Ganesan. Enabling practical backscatter communication for on-body sensors. In *Proc. SIGCOMM*, pages 370–381, Florianopolis , Brazil, August 22–26 2016. ACM. https://doi.org/10.1145/2934872.2934901.

[130] Yang Zhang, Yasha Iravantchi, Haojian Jin, Swarun Kumar, and Chris Harrison. Sozu: Self-powered radio tags for building-scale activity sensing. In *Proc. UIST*, pages 973–985, New Orleans, LA, USA, October 20–23 2019. ACM. https://doi.org/10.1145/3332165.3347952.