Intermittently-Powered Bluetooth that Works

Jasper de Winkel
Delft University of Technology
Delft, The Netherlands
j.dewinkel@tudelft.nl

Haozhe Tang
Delft University of Technology
Delft, The Netherlands
philo.tang@foxmail.com

Przemysław Pawełczak
Delft University of Technology
Delft, The Netherlands
p.pawelczak@tudelft.nl

ABSTRACT

We present an architecture for intermittently-powered wireless communication systems that does not require any changes to the official protocol specification. Our core idea is to save the intermediate state of the wireless protocol to non-volatile memory within each connection interval. The protocol state is then deterministically restored at a predefined (harvested energy-dependent) time, which follows the connection interval. As a case study for our architecture, we introduce FreeBie: a battery-free intermittently-powered Bluetooth Low Energy (BLE) mote. To the best of our knowledge FreeBie is the first battery-free active wireless system that sustains bi-directional communication on intermittent harvested energy. The strength of our architecture is articulated by FreeBie consuming at least 9.5 times less power during device inactivity periods than a state-of-the-art BLE device.

CCS CONCEPTS

• **Hardware → Emerging architectures**; **Wireless devices**; • **Networks → Mobile networks**; • **Computer systems organization → Embedded systems**.

KEYWORDS

Intermittent Computing, Battery-free, Embedded Systems, Bluetooth, Mobile Networks, Energy Harvesting

ACM Reference Format:

Jasper de Winkel, Haozhe Tang, and Przemysław Pawełczak. 2022. Intermittently-Powered Bluetooth that Works. In *The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), June 25–July 1, 2022, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3498361.3538934

1 INTRODUCTION

Billions of Internet of Things (IoT) devices, such as the one seen in Figure 1, will surround us in the coming years [8, 78]. This implies that the batteries (powering the conventional IoT ecosystem) would need to be regularly replaced, monitored, and recycled [31]—inducing an environmental impact to our planet and monetary cost to the consumer [30, 48]. While the search for a battery that is longer-operational [31], better recyclable [69] and having high-energy density [93] continues, the road leading to such batteries is still long [17, 110].

MobiSys '22, June 25–July 1, 2022, Portland, OR, USA

ACM ISBN 978-1-4503-9185-6/22/06.
https://doi.org/10.1145/3498361.3538934

Figure 1: First battery-free open-source [1] smartwatch with FreeBie: intermittently-powered bi-directional BLE link.

A dedicated line of research addressing the battery sustainability problem tries to find out how we can build (wireless) IoT completely independent from batteries [50, 80, 88, 91]. In battery-free systems, a conventional battery is either removed or replaced by *a capacitor*: an energy reservoir that is cheaper, non-ageing, non-leaking, lighter,[1] less dependent on sparsely-available chemical elements and temperature-stable. The energy to power such a battery-free system comes from ambient sources, for example from solar radiation [23] or Radio Frequency (RF) emissions [43]. Nonetheless, the unique properties of battery-free systems—small storage and unpredictable and intermittent energy—create one-of-its-kind challenges for the design of IoT wireless communication.

► ***Problem Statement:*** An IoT sensor requires a wireless link to communicate. This link needs to be ❶ low-power—to *operate as long as possible* on a single energy charge, ❷ belonging to one of the mainstream wireless standards—to be *backward compatible* with already deployed networks, ❸ independent from specific infrastructure, such as RF carrier wave generators to backscatter on—to be *flexibly deployable*, and ❹ bi-directional, as opposed to backscatter IoT tag-to-infrastructure links only—to *enable remote maintenance* and firmware updates [7, 12] as already-deployed IoT devices are severely limited without the ability to reconfigure it wirelessly. To address requirements ❶ and ❷ plenty of research has been dedicated to enabling ultra-low-power communication for popular wireless communication standards—leading to battery-free operation—through the backscatter principle. Recent examples of such wireless standard-compliant backscatter-based transmitters include LoRa [57], BLE [112] and Long Term Evolution (LTE) [22]. These wireless systems however do not address requirement ❸. A related approach based on a wake-up radio [81] is also not viable for the same reason. Therefore the only solution to battery-free IoT that addresses requirement ❸ is an ultra-low-power active radio.

[1] We note that in the weight-optimised Bluetooth Low Energy (BLE) node, the battery is its heaviest component [54, Table 1], while in size-optimised designs the BLE node battery was bigger than the BLE node itself [18, Figure 1].

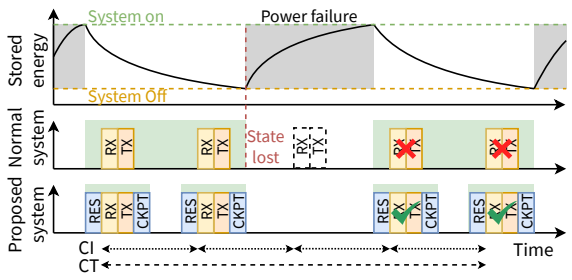

Figure 2: Intermittently-powered device operation. In a non-protected system even a single power failure causes the network state to be lost, requiring unnecessary new handshakes. In our proposed system the network state is stored and restored to/from non-volatile memory enabling to sustain connections. CI: connection interval, CT: connection timeout, RES: state restore, CKPT: state checkpoint, RX/TX: reception/transmission. ▮ region denotes the period of devices on time.

However, intermittent operation[2] of a battery-free active radio, i.e., non-backscatter, IoT node implies that the active radio communication also becomes intermittent. Unless steps are taken to sustain the protocol state despite a power interrupt, even a single one will cause a connection drop, see Figure 2, forcing connection re-establishment which itself consumes time and energy of a battery-free system. Therefore, a design of a truly bi-directional (connection-based) wireless active link for a battery-free IoT sensor—addressing requirement ❹—needs to sustain already established connections despite frequent power interrupts. Unfortunately, all battery-free intermittently-powered state-of-the-art active radio platforms available are connection-less, or require the system to reconnect after each power failure. This includes broadcast-only (best-effort) BLE communication [42]. Simply, classical techniques to prolong IoT life based on duty cycling do not apply: intermittent power supply takes away the guarantee that the energy will be available at the scheduled wake-up times of the device [42, Section 7.1]. All this results in wireless communication being one of the unsolved challenges in the intermittent computing domain [94, Section 4(b)]. This challenge has not been tackled yet due to complexity of the problem: system designer needs to simultaneously consider network protocol specification constraints, application needs, energy demands and energy use. Also, the most common way of dealing with power interrupts: *checkpoining* system state and restoring it as the energy returns [67, Section 3.2], has never been applied to wireless network protocols. Checkpoints restoration have usually random duration and have to be placed at a precise point in time not to break the protocol timing. Therefore, a battery-free IoT that is useful, i.e., that addresses requirements ❶, ❷, ❸, and ❹, has yet to be achieved.

► ***Contributions:*** Addressing the above problem we provide the following contributions:

Contribution 1: **New intermittently-powered wireless systems architecture.** Our core novelty is a *new battery-free networked IoT device state checkpointing system operating on a process level*. Each core class of IoT device processes—application, network and Operating System (OS)—are checkpointed and stored in non-volatile memory individually. Checkpoints are triggered by the process scheduler based on real-time requirements of incoming processes. Checkpoints triggered by the scheduler protect network protocol state timing from unnecessary in-between checkpoints. Moreover, our novel ultra-low power timekeeping architecture allows for microsecond-level time granularity. This enables keeping track of network protocol state in-between power interrupts and resuming already established communication when energy conditions allow. Both architectural novelties give rise to a *new concept of time and peripheral abstraction layer*: the original network software stack never sees that the radio is powered off even when the storage capacitor is fully depleted.

Contribution 2: **Implementation of bi-directional active wireless link powered intermittently.** We pick one of the most ubiquitous IoT wireless system: Bluetooth, and in particular its low energy configuration BLE [15], and build a *BLE system powered intermittently*, called FreeBie, with its hardware and software released as open-source [1]. With our architecture, a battery-free BLE node can communicate bi-directionally with any BLE host and can sustain an already established bi-directional BLE link—even after multiple power outages at the battery-free BLE node.[3]

Contribution 3: **New battery-free IoT applications.** Our novel architecture enables never before seen IoT applications, in particular BLE *firmware updates* performed battery-free on an intermittent power and a *fully-functional battery-free smartwatch*, shown in Figure 1. Our architecture provides a foundation to connect and communicate bi-directionally for the next generation of battery-free IoT devices.

2 BACKGROUND, CHALLENGES AND KEY INSIGHTS

Taking a recent example, long-term experiments with a commercial-grade battery-free BLE node of [25] demonstrated that time of the day, orientation, and deployment location can affect the duration of continuous operation: from almost constant operation to few transmission activities throughout the day only [61, Section 5]. Therefore, necessary system support for intermittently-powered devices is needed that takes care of [67, Section 2] (i) *control flow*—to guarantee that the device will start from the state right before the last power failure, (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 data becomes outdated after restoring from power failure, (iv) *concurrency*—to enable execution of multiple active applications, and (v) ***undisrupted communication***—to enable wireless communication between (intermittently-powered) devices and guarantee synchronisation despite power interrupts.

While there are plenty of frameworks available that aim at attaining points (i)–(iv), see Table 1 and Section 7 later on, sustaining

[2]For ambient energy-powered battery-free devices on/off times can range from microseconds [83, Figure 1] to seconds [61, Figure 2(a)] for solar- and Radio Frequency-powered devices, respectively.

[3]A comparison of battery-free BLE platforms is given in Table 5.

	InK [109]	TICS [63]	Coala [68]	Capybara [23]	MPatch [29]	Empire [3]	*This work*
Checkpoint trigger	Task end	In-code checkpoint	Task end	Task end	Voltage level	Voltage level	Process end
Checkpoint type	Task	Checkpoint	Task	Task	Checkpoint	Task	Process
Requires code instrumentation	Yes	No	Yes	Yes	No	Yes	No
Time-deterministic restoration	No	No	No	No	No	No	Yes
ARM-based processor architecture	No	No	No	Yes	Yes	No	Yes
Dynamic memory allocation	No	No	No	No	No	No	Yes
Interrupt support	Yes	Yes	No	No	Yes	No	Yes
Peripheral support	No	No	No	Yes	No	Yes	Yes
Preemptive scheduling	Yes	No	No	No	No	No	Yes

Table 1: Comparison of relevant existing software support systems for intermittently-powered devices. Colour scheme: denotes non-desired or limiting features from the perspective of wireless communication protocol and denotes the desired features.

a communication of a radio link powered by an intermittent source, i.e., attaining point (v), is far from being solved.

▸ ***Infeasible solutions for 'intermittent' communication:*** Increasing capacitor size or energy harvesting efficiency, is a typical solution to improve battery-free systems. Sadly, a large capacitor/energy harvester increases the device's size and increases charge times. So all active radio intermittently-powered devices trade-off capacitor and/or harvester size for communication functionality, sending connection-less data, such as beacons in case of BLE, e.g. [56, 92]. Simply, when the device powers off mid-transmission, the device will restart and re-send the beacon again. This however is not feasible for connection/handshake-oriented protocols, especially if they require strict timing to establish and sustain a connection. Lastly, reconnection costs packet transmissions. For example, for our smartwatch's BLE link each reconnection would **require about 70 packets to be sent** by the BLE client, taking more than 43 s to reconnect at low (200 lx) ambient light. These packets include connection establishment, negotiation of connection parameters, service discovery and notification configuration. Apart from the above-mentioned connection re-establishment overhead, if the BLE device needs to re-establish the already-existing connection, one would have to build much more complex application. Such application would have to take care of storing authentication information, tracking network status, or transfer progress information—all causing processing overhead, i.e. taking extra time before the existing connection is re-established.

Other techniques, such as 'classical' duty cycling [21] and transmission power control [36], are also infeasible. Duty cycling is problematic, as the main capacitor may get depleted within the sleep period resetting connection state, while duty cycling itself consumes energy during sleep, e.g. for Nordic Semiconductors NRF52840 BLE module sleep current is 3.16 µA at 3 V [74]. Needless to say, it is impossible to turn the system off completely and then wake up: some components like Real Time Clock (RTC) need to remain powered to wake up the system from sleep. In the case of transmission power control, the transmitted power reduction does not translate to significant overall gains for the device. The same observation applies to system adapting transmitted packet lengths: reducing packet length would not scale linearly with energy expenditure, as overhead such as the crystal ramp-up time and pre- and post-processing remain constant [75]. Thus, the goal is to *create a framework that enables unobstructed communication on intermittently-powered budget.*

Protocol type	Implementation	Lines of code[1]
Bluetooth	Packetcraft [10]	397 200
TCP/IP	LWIP [38]	88 100
Thread	OpenThread [47]	250 500

[1] Measured with cloc [27] (rounded to 100 lines).

Table 2: C/C++ lines of code of different network protocol implementations. Compared to the orders of magnitude smaller codebase of software support systems for intermittent operation, e.g. [68, Table 3], the cognitive burden to analyze and instrument protocol code is unprecedented.

Framework Requirement 1: **Time-aware Checkpoints and Real-time Restoration.** We postulate that if the duration of power failures is within the allowed connection timeout (see again Figure 2), the transmitter and the receiver should resume the connection without the need to restart. This resumption must be supported by a framework responsible for copying of the protocol state, i.e., volatile MCU registers and volatile memory to a non-volatile memory before the power interrupt, and restoring the last saved state back to the respective volatile registers.

There exist two classes of frameworks that optimise the amount of memory to store and resume: task-based, e.g. [68] and checkpoint-based, e.g. [63]. A task is a section of code with defined input and output of non-volatile variables. Tasks are connected through these variables to form a state machine. Sadly, automatic/compiler-based transformation of a wireless protocol implementation code into tasks is not feasible due to the large and complex codebase, see Table 2. This means that the developer would have to do this by hand—*a daunting task to achieve* [63, Section 5.4]. A conceptual counterpart to a task is a checkpoint, i.e., a function inserted manually or automatically at compile time (to the original codebase) that stores the program's state until that checkpoint. Sadly, both tasks and checkpoints introduce a computation penalty—the store and restore operation. In other words, *checkpoints or tasks will break the protocol's timing* nullifying their use for wireless networking. Fortunately, state saving and restoration can be triggered not only by the end of a task or a checkpoint, but also by the internal timer or the capacitor voltage monitor. However, the timer would have to follow the protocol state timing *precisely*, while the voltage-based trigger does not follow any timing. Finally, the restoration time from the checkpoint must be constant and time-bounded. Otherwise, the real-time requirement of the wireless protocol state machine will be violated.

To summarize, **the key challenge** is that existing checkpoint and restoration methods, see Table 1, are not suitable for wireless networking support. **Our key insight** to solve the challenge is that *the system state checkpoint can be triggered by the end of the wireless protocol process*, i.e., the only atomic operation that must be executed without interruptions. This way no code instrumentation, fixed checkpoint timers, and voltage monitors are needed.

Framework Requirement 2: **Virtualisation of time and peripherals.** Time and peripheral state would be disrupted by intermittent operation. Thus, dedicated software abstraction layer to mask (*virtualize*) time and peripherals to network protocol is needed. **The key challenge** is that, referring again to Table 1, no software framework is able to support peripherals and time (required for time-deterministic restoration). **Our key insight** to solve the challenge is the following conjecture. To enable masking of time and peripherals *only one intermittently-powered device component, i.e, on-board time reference, must be continuously powered as long as possible* (through a dedicated capacitor). The current draw of modern low-power RTCs is in the order of nA [5]. As the sleep mode consumption reaches μA levels, only keeping RTCs on is *about ten times more energy efficient* than sleeping.

Framework Requirement 3: **Dynamic handling of network connections.** A framework must adapt connection parameters to available harvested energy. The framework must prioritize an already-established connection or focus on sustaining on-device computation and sensing. The system must support preemptive scheduling (to prioritise network processes over application or OS processes). **The key challenge** is that these features are also not supported by existing systems except for InK [109] (although InK requires manual code transformation to achieve this), refer to Table 1. **Our key insight** to solve the challenge is that network protocols allow adjustment of the Connection Interval (CI), this and other parameters can be used as a foundation for connection adaptation.

3 INTERMITTENTLY-POWERED WIRELESS SYSTEM

Driven by Framework Requirements 1, 2 and 3, we propose an *architecture that sustains wireless protocol communication for intermittently-powered devices*, see Figure 3.

3.1 Target Network and Device Architecture

► ***Network Topology and Device Capabilities:*** We consider a star network topology following a Connection Interval (CI)-oriented Connection Timeout (CT)-driven wireless communication protocol of choice, as exemplified in Figure 2. *The host is tethered or battery-based.* The end device, on the contrary, is battery-free and intermittently-powered (by energy harvester). We assume that both devices do not share a common signal that can be used to synchronise them, as in e.g., [41]. As with most network protocols, the end device has to announce its presence to a host for a connection to be established.

► ***End Device Hardware:*** We propose a battery-free end device logically separated into two power domains, see Figure 3: (i) *processor (MCU) power domain* and (ii) *"always-on" ultra-low-power domain*—charging on-board capacitors through onboard solar panels, through which processor power domain and external RTC is powered. The external RTC not only keeps track of time but also is able to switch the processor power domain off completely, as proposed in [58].

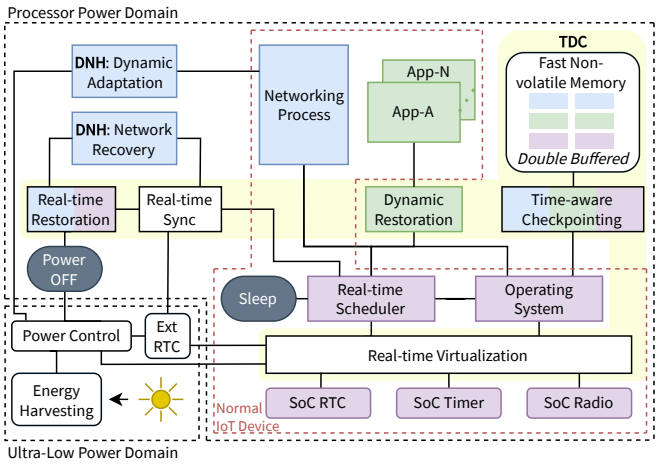

Figure 3: Proposed system architecture for the intermittently-powered battery-free wireless communication system.

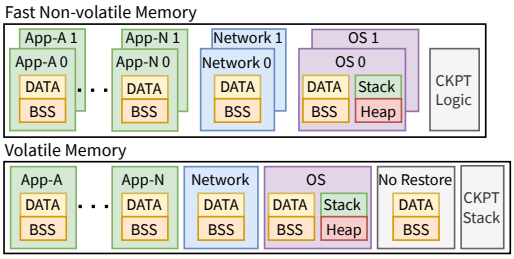

Figure 4: Memory map of intermittently-powered wireless networking device. Numbers 0 and 1 for App-*x*, Network and OS process denote buffer numbers of double buffered memory. Colours match respective processes in Figure 3.

3.2 System Components

3.2.1 Time-aware Checkpoints and Real-time Restoration. We propose *Time-Deterministic Checkpoint (TDC)*: a time-aware checkpoints and real-time restoration system for intermittently-powered wireless networking protocols. TDC, addressing Framework Requirement 1, is built as follows.

► ***Process as Atomic Data Structure Checkpoint:*** We categorize processes in three groups: (i) *network*, (ii) *OS* and (iii) *application* (with one process per concurrently-running application). Implementation-wise, a process is a hand-picked file/directory part of a source code. Each process can be classified as requiring real-time operation or not. Per default the processes network and OS are real-time processes, the application process(es) can be real-time or non-real-time.

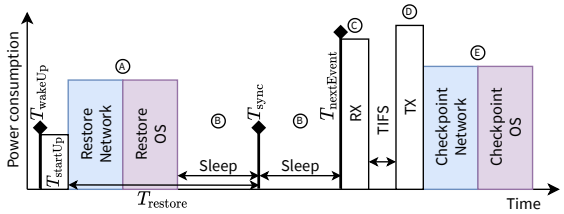

Figure 5: System events in one network-only cycle (power consumption levels marked as Ⓐ–Ⓔ); TIFS: Time Inter-Frame Space, T_{sync}: RTC time synchronisation moment.

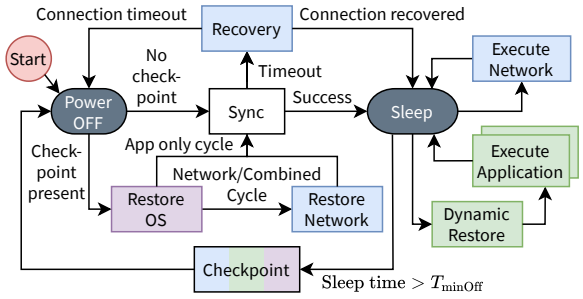

Figure 6: Restoration of a process after power off.

► ***Process Separation and Memory Structure:*** The memory separation for all processes is performed by the linker at the code compilation stage. The developer splits the source code of a complete system into code directories—one per process. Static memory declared in the source files of each directory is allocated in separate regions as shown in Figure 4. The registers, stack and heap are included in the OS process checkpoint.

We assume that the memory from a non-restored process cannot be read or written to. This can be enforced by a Memory Protection Unit (MPU), preventing writes and reads to unrestored memory. Communication between processes occurs through the OS using Inter-Process Communication (IPC) with dynamically allocated messages. Finally, we note that the developer can specify variables (such as a buffer for logging) that do not need to be checkpointed and restored. For this case, a dedicated region in volatile memory is allocated; see Figure 4.

► ***Process Checkpoint Scheduling:*** In modern network protocol stacks, each networking process, at the end of its execution, provides information to the device OS when the next networking process, e.g., beginning of a next connection interval as shown in Figure 2, occurs. Therefore the core component of our architecture is a process scheduler handled by the battery-free device's OS. The scheduler using the virtualisation layer (defined in Section 3.2.2) decides on the next moment in time when the battery-free device powers on again. The power-on moment is determined by the process with the earliest deadline in the scheduler's queue. If more than one process has the same deadline, than these are served as first-come first-served. Each running application shares the same priority and is queued through a scheduling queue, while the network process (through an interrupt) can preempt the application process residing in the queue's head.

Considering what processes are at the top positions of the scheduler queue pending execution three combinations of processes that occur during a power-on cycle exist. These are: (i) application only, (ii) network only, and (iii) combined application and network. In case (i) and (ii) respectively, the network process or application process does not have to be checkpointed and restored. Case (iii) occurs when the application process is scheduled for execution close to a network process, or when the network process triggers the application to run. In this case, the application is dynamically restored by the scheduler. When there is no process awaiting execution, the MCU is placed either into sleep mode or the processor power domain is switched off. The decision of whether to switch off the processor power domain or to sleep is based on the duration of the checkpoint and restoration process. In our architecture, we switch off the processor power domain if the next process event is scheduled to start later than T_{minOff}, i.e., as a minimum the combined time of checkpoint and restoration. Otherwise, the MCU is set into sleep mode. Depending on the implementation T_{minOff} can be set to the break even point between additional energy cost of checkpointing and restoring and the power saved by turning the processor domain off. Then, the processor domain only switches off when it is energy-wise beneficial to do so, however, this does lead to less frequent checkpoints.

Prior to switching off, the state of the system has to be checkpointed. First, the next power-on time is determined—that particular step is performed by the time and peripherals virtualisation layer described in Section 3.2.2. Next, based on the scheduler queue, the real-time processes that need to be restored during the next cycle are determined. This information is stored in the next OS checkpoint. Then, any process combination—either (i), (ii) or (iii)—that has been executed during the current power-on cycle is checkpointed, followed by the OS checkpoint. Finally, the device is switched off. An illustration of a power cycle consisting of only the networking process is presented in Figure 5.

► ***Process Checkpoining:*** Application and network process checkpoints are always committed with an OS process checkpoint. This protects from a situation where memory can be dynamically allocated within an application or network process and then lost due to an incomplete OS checkpoint. In order to make the system incorruptible each checkpoint is double-buffered, with two dedicated memory regions allocated for each process, as shown in Figure 4. The OS process checkpoint must be restored after the scheduled wake-up from power-off as it includes the processor registers, stack, heap, scheduler, OS functionality (such as timers, queues and IPC), and the peripheral state.

► ***Process Restoration:*** Processes can be restored either as non-real-time or real-time. Non-real-time processes are loaded dynamically prior to execution by the scheduler. For non-real-time checkpoints the restoration time is neither monitored nor compensated.

Unlike non-real-time processes, real-time processes are restored in advance prior to the scheduler resuming operation. As process

checkpoint sizes may vary, we add a margin on top of the processes restore time. The allocated time for restoration, including the margin, is denoted as T_{restore}.

Upon resuming from the power-off state, the system begins process restoration, as shown in Figure 6. With a checkpoint present, the OS process is restored together with the network processes and real-time application processes as determined by the scheduler in advance. Once the virtualisation layer compensates for the power-off time the system synchronises with the external time source (refer to Section 3.2.2) and the device resumes operation as normal, i.e., moving to sleep mode and then executing the networking process or other application processes. If the device is unable to power back up at the desired time after switching off, or if the time synchronisation was unsuccessful, a recovery process is instantiated driven by *Dynamic Network Handling (DNH)*, as explained in Section 3.2.3. During the first boot, the system is synchronised to the external time source and starts operation.

3.2.2 Virtualisation of Time and Peripherals. We propose a time and peripheral virtualisation layer, addressing Framework Requirement 2. The virtualization layer is placed logically between the peripherals and the pre-existing network software stack (see Figure 3).

Our architecture needs a method to synchronise to external RTC time for our system to deterministically execute real-time processes. This synchronisation step needs to occur after real-time process restoration and prior to the scheduler resuming operation and is performed as follows. When power is reapplied to the processor power domain, the MCU starts up and the boot time of the MCU, denoted as T_{startUp}, is considered consistent. Next, the real-time processes are restored, as described in Section 3.2.1 (*Process Restoration* paragraph). Finally the system awaits a synchronisation pulse at T_{sync}, where $T_{\text{sync}} \leq T_{\text{nextEvent}}$ and $T_{\text{nextEvent}}$ is the starting time of the upcoming process event. T_{sync} is the point in time that synchronises the system to the external time and marks the resumption of real-time operation and scheduling. T_{sync} should be as close as possible to $T_{\text{nextEvent}}$ as allowed by the external RTC resolution to avoid overhead. The next wake-up time is given as $T_{\text{wakeUp}} = T_{\text{sync}} - T_{\text{startUp}} - T_{\text{restore}}$. Since T_{sync} is known prior to turning off, this value is stored as part of the OS checkpoint and used as a reference starting value for the MCU timing peripherals during the next power-on cycle.

3.2.3 Dynamic Handling of Network Connections. Addressing Framework Requirement 3, we introduce DNH—a final component of our architecture. DNH is responsible for: (i) network recovery and (ii) dynamic network adaptation. *Network recovery* is needed when the device does not turn off according to the schedule but turns off unexpectedly due to a power failure. As most wireless network protocols operate based on Connection Timeouts (CT), when a connection has been established but no packets are received within the CT window, the connection is dropped (see Figure 2). In our architecture, if the device after a power failure can harvest enough energy to turn on before CT is exceeded, connection recovery is executed when the device powers back up before resuming operation. That is, missed connection events are skipped and the network process is scheduled for the next connection event. *Dynamic network adaptation* further improves the performance of our system, by monitoring the available amount of energy. The Connection Interval (CI) is decreased in the case of abundant energy and increased when little energy is available. This method allows the system to adapt to changing energy conditions whilst keeping the connection alive and increases responsiveness in the case of abundant energy.

Figure 7: FreeBie mote (front side). A total size is 1"×1". Components marked as Ⓐ–Ⓙ are explained in Section 4.

4 SYSTEM IMPLEMENTATION: FREEBIE

We proceed with the implementation. As a case study we select BLE and denote its intermittently-powered version as FreeBie.

4.1 FreeBie Hardware

A fabricated FreeBie is shown in Figure 7, with hardware and software open-sourced [1]. Its main blocks are as follows.

► ***Wireless Connectivity and Storage:*** FreeBie is built with a wireless module [24] containing a nRF52840 BLE ARM-based MCU [74] (Ⓐ in Figure 7). To store the state of the system in-between power failures, MB85RS4MT fast non-volatile Ferroelectric Random Access Memory (FRAM) [39] (Ⓖ in Figure 7) is used and connected to the MCU using SPI.

► ***Timekeeping:*** The AM1815 RTC [5] is chosen (Ⓑ in Figure 7) for its 10 ms resolution and low power consumption, i.e., 55 nA at 3 V. We did not use hardware timer like the TPL5111 [103] used by [54], or battery-free timekeeping architectures [28] due to their inability to sustain the clock accuracy of the BLE specification [15, CS 5.3], as stated in Section 2.

► ***On-board Sensors:*** FreeBie contains two external sensors: an OPT3004 luminosity sensor [102] (Ⓒ in Figure 7) and a BMA400 accelerometer [16] (Ⓓ in Figure 7). Both sensors are powered through the MCU only when required. These sensors are included in FreeBie to enable the community to build new applications on top of the FreeBie mote.

► ***Energy Management:*** FreeBie is solely powered by harvested solar energy using a BQ25570 energy harvester [104] (Ⓔ in Figure 7). Its boost converter boosts the voltage generated by two EXL2-1V50 solar panels [65], as seen in Figure 1. The harvested energy is stored in two parallel 7.5 mF capacitors [89] (Ⓕ in Figure 7). The chosen storage size is dictated by the compatibility with an Android [46] OS, used as one of the BLE hosts (see Section 5).

Android initialises a BLE connection with a CI of 45 ms. The chosen storage must sustain this CI until new connection parameters can be applied.

The processor domain is powered by energy harvester's internal buck converter configured to generate an 1.8 V supply. The energy harvester switches the buck converter on when the voltage of the storage capacitors reaches 2.6 V and switches it off when it drops below 2.05 V. With the external power switch [106] (Ⓗ in Figure 7), the external RTC is able to switch off/on power to the processor power domain. In order to protect the MCU while it is off, logic/switches prevent always-on signals from reaching the processor (Ⓙ in Figure 7).

► ***Display:*** We added the option to connect a display to the FreeBie mote. Specifically, a Sharp 1.28" LCD-TFT [90], connected through SPI to the MCU (both connectors for the solar panels and display are present at the back of FreeBie). The display is used in a smartwatch application and is powered from the main energy storage. The display power can be switched on or off by the MCU and the on/off state is maintained using a SN74AUP2G79 flip-flop [98] allowing the display to stay on when the MCU is off (Ⓘ in Figure 7).

4.2 FreeBie Software

The Packetcraft BLE stack [77] was chosen as the basis to implement FreeBie's system architecture. Packetcraft implements all the required Bluetooth standard layers—from layers that configure the MCU's registers to high level layers (such as Attribute Protocol (ATT) profiles).[4]

With our architecture only relatively simple modifications are required to run the BLE networking stack intermittently. First, the code source files needs to be separated into application, network and OS processes, allowing for easy checkpointing and restoration. Second, the scheduler needs to be modified to allow the system to switch off when idle and compensate for the power-off time of the device upon restoration. Finally, dynamic connection handling is build on top of standard BLE features. The implementation is described in detail below.

4.2.1 *Time-aware Checkpoints/Real-time Restoration.*

► ***Process Separation Implementation:*** Process separation described in Section 3.2.1 is implemented as follows. Thanks to Packetcraft's logical separation between OS layer (called Wireless Software Foundation (WSF) with its underlying components—Platform Abstraction Layer (PAL) and the MCU peripheral drivers) and networking layer, manual source file separation is straightforward—WSF together with its underlying components form the OS process source files. The rest of Packetcraft source files are considered to belong to the network process. Applications are considered as a separate third group of process source files. With the separation of the code into processes, the network, application and OS volatile memory is split per process. These processes encompass all volatile memory associated with the process except for dynamically allocated memory that is contained within the OS process.

[4]We are aware of other open-source implementations of BLE, namely Apache Nimble [6], and Zephyr [111]. From these implementations only Packetcraft supports BLE 5.2, can be deployed on Nordic nRF52 series Microcontroller Units (MCUs), and can be built with the standard GCC toolchain [9].

► ***Process Checkpoint Scheduling Implementation:*** The proposed scheduler, introduced in Section 3.2.1 (*Process Checkpoint Scheduling* paragraph), is implemented as follows, taking the WSF scheduler as basis. Normally when the OS scheduler is idle, the function `PalSysSleep()` is called and the system sleeps until the next process event. This function is extended to allow the system to checkpoint and turn off per the proposed scheduler criteria. In our implementation $T_{\text{minOff}} = 20$ ms is experimentally determined, and the type of next on power cycle is determined through the scheduling queues and MCU's RTC compare registers. When the system is able to turn off, first the next power-on time is determined by the virtualisation layer (its implementation will be described in Section 4.2.2). Then, real-time processes to be executed are marked for restoration in the next OS checkpoint. Finally, the currently restored/active processes are checkpointed, followed by the OS checkpoint, after which the processor power domain is switched-off through the virtualisation layer.

Secondly, we introduce a major change to the OS scheduler (function `wsfOsDispatcher()`)—the notion of *restored* and *non-restored* processes. Non-real-time processes are scheduled in a different queue than real-time processes, and each process possesses a state variable that indicates if the process has been restored or not. When a non-real-time process has not been restored prior to execution, the process is first restored, then marked as restored and finally executed. If the process has already been restored, the process is simply executed. Since all real-time processes for the next power-on cycle are identified in advance (and restored prior to the scheduler resuming operation), they are not tracked during operation since these processes (due to their real-time requirements) cannot be loaded on demand.

► ***Process Checkpointing Implementation:*** The checkpointing framework checkpoints the uniquely defined memory sections of each process in volatile SRAM and stores the checkpoints in external FRAM. For the OS checkpoint, the stack size is determined using the stack pointer register and the heap size is determined by tracking the total size of the dynamically allocated memory, only the utilised portions of the reserved space for the stack and heap are checkpointed. As described in Section 3.2.1 (*Process Restoration* paragraph), due to T_{restore}, variations in checkpoint size and thus restoration time of the real-time checkpoints are compensated for.

► ***Process Restoration Implementation:*** When the MCU powers up, first the external RTC is configured for the synchronisation point as defined in Section 4.2.2. The time of this point has been pre-selected before the MCU switches off as defined in Section 3.2.2. Next, the relevant process checkpoints are restored from FRAM to SRAM. Directly reading from external memory is not possible since it is slower than reading from internal SRAM and thus would influence the timing of the BLE network stack. The restore time T_{restore} is set to 10 ms.

4.2.2 *Virtualisation of Time and Peripherals.*

The real-time virtualisation presented in Section 3.2.2 is implemented as follows. Due to the choice of external RTC, we are limited by a resolution of 10 ms. Since 10 ms is not an integer multiple of 32.678 kHz tics, i.e., the frequency of our external RTC, the 10 ms tics source induces additional jitter (in addition to the jitter of the crystal itself). Due

to our strict timekeeping requirements (500 ppm) this is not acceptable. Hence for synchronisation of the system to the external RTC time we synchronise to 250 ms intervals as synchronisation points, as this is the smallest integer multiple of 32.678 kHz tics possible with 10 ms resolution. We note that although the resolution of the external RTC is only 10 ms, on integer multiples, such as 250 ms, the timing is mainly only influenced by the jitter of the crystal itself.

The synchronisation point itself is a hardware signal sent from the external RTC to the MCU that, in turn, instantly starts the enabled MCU's timing peripherals such as the on-board RTC. Since the synchronisation point is a known moment in time, reads and/or writes to the on-board MCU's timing peripherals such as the RTC (as it starts from zero, and the state is lost during power-off) are compensated through the virtualisation layer by applying T_{sync} as an offset. Any read to the RTC time register through the virtualized peripheral API returns the compensated time instead of the raw time. Hereby the effects of intermittency are masked to the running processes.

Using the definition in Section 3.2.2, the start-up time is computed relative to the nearest synchronisation point. The value of $T_{\text{startUp}} = 10$ ms is experimentally found. By setting the alarm in the external RTC to T_{wakeUp} prior to switching off, the system will turn on at the alarm, as the external RTC can control the processor domain power via the external power switch. Finally, after the wake-up alarm is set, using the external RTC, the processor power domain is switched off.

4.2.3 Dynamic Handling of Network Connections. The design presented in Section 3.2.3 is implemented as follows. In the Bluetooth protocol, the host dictates the initial connection parameters. For intermittently-powered devices these parameters, depending on the available energy, might not be suitable. Hence after a connection establishment we automatically request favourable connection parameters corresponding to the energy available in the system. If the BLE host forces a connection update, we will immediately request new connection parameters if the ones chosen by the BLE host are unsuitable.

During the restoration process we sample the voltage of the storage capacitors to determine how much energy is available to FreeBie. For simplicity, we define quantized energy levels as given in Table 3. According to the Bluetooth Core Specification [15, CS 5.3 (page 2255)] (i) CI shall be a multiple of 1.25 ms in the range 7.5 ms to 4 s and (ii) Supervision Timeout (ST) (referred previously as CT) shall be a multiple of 10 ms in the range 100 ms to 32 s and it shall be larger than $(1 + \text{SL}) \times \text{CI} \times 2$, where Slave Latency (SL) specifies how many connection events may be skipped by the end device, i.e., with CI of 4 s and SL of 3 allows FreeBie to stay off for almost 16 seconds). At the low energy level we use the maximal allowed parameters to let FreeBie stay powered off as long as possible. As more energy becomes available, FreeBie harvests more energy so we increase CI and decrease SL accordingly. Table 3 lists the requested connection parameters corresponding to these energy levels. We note that since connection update requests are not instantly applied, if the update is granted—they are applied at a later (specified by the host) connection event.

If during the restoration no synchronisation pulse is received at the expected time, recovery is executed. After reading and synchronising to external time, the decision is made based on the current connection settings if the connection is recoverable. If this is the case, the next connection event is scheduled and the state of the network stack is passed on to the future state. The network stack reattempts transmission of lost packets scheduled during the power failure. During the restore, prior to synchronisation, all real-time processes are restored.

4.3 FreeBie Applications

► ***Benefiting Applications:*** Our architecture is of most benefit to ultra low power systems requiring bi-directional communication. We note that most devices require some form of connectivity not only to send data (like sensor samples), but also for configuration and firmware updates. Our architecture enables bi-directional communication on these severely energy-restricted devices and allows the MCU to switch off completely during idle periods, further reducing power consumption. Examples of such devices include hybrid (classical/smart) watches, IoT devices or on-body sensors—all operating on harvested energy. We chose two applications demonstrating our architecture capabilities: (i) a smartwatch using multiple BLE services to interact with the host; and (ii) firmware updates. Specifically, in the case of firmware updates, as the end device must request and receive firmware fragments from the host while the host transmits the firmware fragments and waits for reception confirmation—the firmware update is a stress-test of bi-directional BLE communication.

► ***Battery-Free Smartwatch:*** We have developed a *battery-free smartwatch* based on two BLE services, (i) the Current Time Service (CTS) [14] and (ii) the Alert Notification Service (ANS) [13], operating on top of FreeBie hardware.

The smartwatch, see Figure 1, works as the ATT client [15, CS 5.3] of those two services. For the BLE host we have developed an application for the Android 11 OS [46] working as the ATT server of those two services. Once a connection is established, service discovery is executed to find the CTS and the ANS on the BLE host. Once successful, the smartwatch enables all notifications of both services. Then, the BLE host sends the current time to the smartwatch, which is later on updated independently of the host through the application process that runs every minute to increment time. In addition, the BLE host sends unread email notifications to the smartwatch which activates its on-screen email icon.

► ***Firmware Update:*** We have also implemented the first *battery-free active-radio over-the-air firmware update*. Although successful demonstrations of battery-free over-the-air reprogramming were done for backscatter-based nodes [2, 96], battery-free active-radio firmware updates (to the best of our knowledge) have never been demonstrated.[5]

We designed a custom BLE service for our firmware update application where FreeBie works as the ATT server of that service. Once connected, the BLE host should first send the firmware length and then initiate the update. Once initiated, FreeBie starts requesting a

[5] We note that this case study aims not to create a novel firmware update application [7, 12] but to evaluate FreeBie.

Energy level	Luminosity	CI	SL	ST
Very low (dark)/low (dimmed)	200 lx/300 lx	4 s[1]	3	32 s
Medium (bright)	600 lx	2 s	1	32 s
High (overcast)	10 klx	1 s	0	32 s

[1] Real value is 3.998 75 s as reference BLE stack [71] forbids 4 s CI.

Table 3: Requested FreeBie connection parameters. CI: Connection Interval, SL: Slave Latency, ST: Supervision Timeout.

firmware section by sending the index of the section. Once FreeBie receives the requested firmware section, it writes this section to the corresponding address in the inactive firmware region. This process repeats until FreeBie obtains the complete firmware.

5 FREEBIE EVALUATION

5.1 Evaluation Setup

We evaluated both FreeBie applications at four light conditions in a controlled environment, 200 lx, 300 lx, 600 lx, 10 klx representing, respectively, (i) dark indoor light, (ii) dimmed indoor light, (iii) bright indoor light, and (iv) overcast day.

► ***BLE Host Hardware and Software:*** For smartwatch, an Android application was built running on Google Pixel 3a [45] with Android 11 OS [46] acting as BLE host. For firmware update, a nRF52840 [72] development kit using SoftDevice [71] acts as BLE host. FreeBie is compared to Packetcraft [77] and SoftDevice [71]; comparison with checkpoint-based systems, e.g. [63], is impossible as *they do not work*, see Section 2 and Table 1.

► ***BLE Advertising and Connection Parameters:*** The advertising interval of FreeBie is fixed at 2 s. For the hosts, per default Android 11 starts with a Connection Interval (CI) of 45 ms, 5 s Supervision Timeout (ST). For the firmware update host, we selected an initial 2 s CI, 32 s ST.

► ***Controlled Test Environment:*** We put FreeBie at the bottom of a closed light box. A wirelessly-controlled LED bulb [62] was installed at the top of the box to create repeatable and controlled light source. A luminosity meter [105] was placed next to FreeBie mote to measure the exact luminosity projecting onto FreeBie's solar panels.

► ***Long-term Evaluation:*** For a day-long evaluation we have collected luminosity values from a modern commercial smartwatch [32] worn on a wrist. Data was collected when the user[6] was performing (mostly outdoor) daily activities. The luminosity trace was then recreated in the controlled test environment described above.

► ***Power Consumption Measurements:*** Connection event and sleep power consumption for the Packetcraft [77], SoftDevice [71] and FreeBie's were measured with X-NUCLEO-LPM01A [95]. FreeBie's power consumption whilst the processor domain is off was measured with Keithley 2450 SMU [59]. Power consumption of Packetcraft and SoftDevice is measured on the NRF52840 development kit [72]; FreeBie power measurements are measured on the FreeBie mote. Further details of the evaluation setup are given in the artifact [1].

[6]The data collection was approved by the human ethics committee of the institution with which the authors of this study are affiliated with.

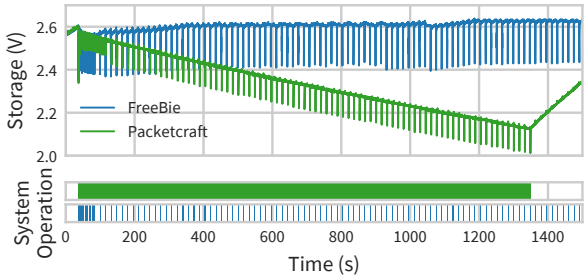

Figure 8: Example BLE connection retention on FreeBie hardware at 200 lx compared against Packetcraft [77]. The system operation bars (bar colour matches system on the 'Storage' plot) indicate when the system is on.

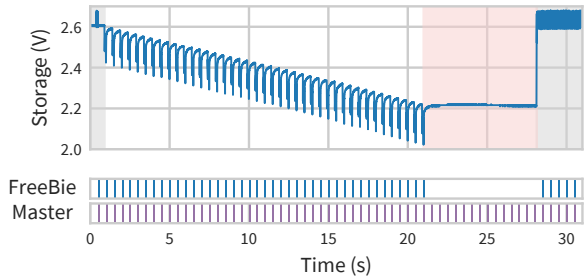

Figure 9: FreeBie power failure recovery. Despite missing several BLE packets FreeBie can recover the connection. : BLE network activity by the host, : FreeBie network activity, : FreeBie is actively powered, : FreeBie power failure.

5.2 FreeBie Evaluation Results

► ***BLE Connection Retention:*** First, to demonstrate that our system can sustain a BLE connection at intermittent power, we run a basic ≈30 min-long BLE connection. The result is presented in Figure 8. We clearly see that our system consumes less power operating intermittently and maintains the connection, while the default Packetcraft network stack [77] powers off below ≈2 V and never resumes the connection.

► ***BLE Connection Recovery:*** To show that FreeBie can recover from power failures, we powered FreeBie from a stable power supply during a BLE connection. Then we turn off the power supply until FreeBie runs out of power. Then the power supply is resumed again, triggering the connection recovery. A snapshot of this process is presented in Figure 9. We clearly see that FreeBie can recover before the 32 s ST is reached.

► ***Checkpoint and Restoration Overhead:*** First we measured code size for both FreeBie applications, split per process. The results are presented in Table 4. Our firmware application is small, therefore little is gained during a network-only power cycle by not restoring the application. During an application-only cycle, however, where the network process does not have to be restored,

	Smartwatch			Firmware update		
Process	Data	BSS	Text	Data	BSS	Text
Application	0	2368		0	12	
Network	376	5668		320	5168	
OS	292	2392		244	2452	
Total	668	10 428	230 120	564	7632	204 797

Table 4: Code size of the FreeBie applications (in bytes).

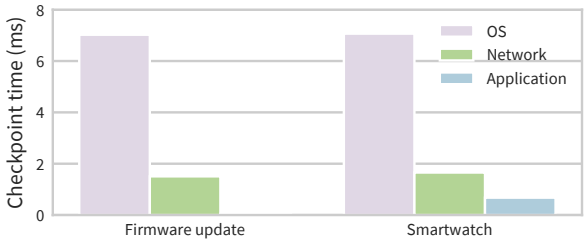

Figure 10: Average checkpoint time of each process for the two considered applications.

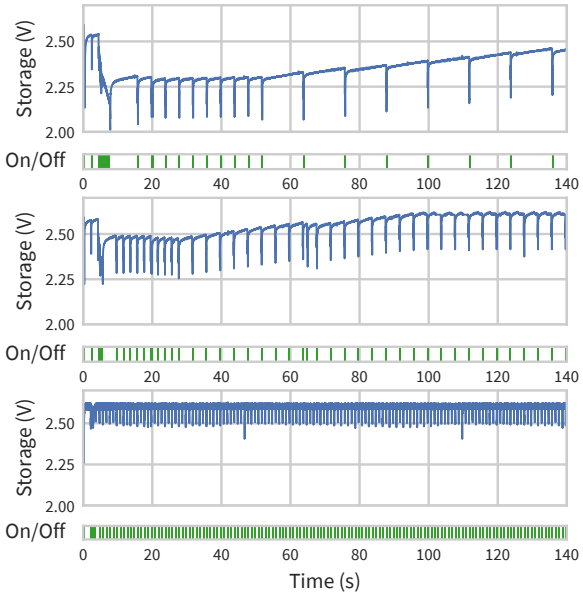

Figure 11: FreeBie smartwatch operation at three luminosities (top to bottom figure: 300 lx, 600 lx, 10 klx). Connection parameters for each luminosity are given in Table 3.

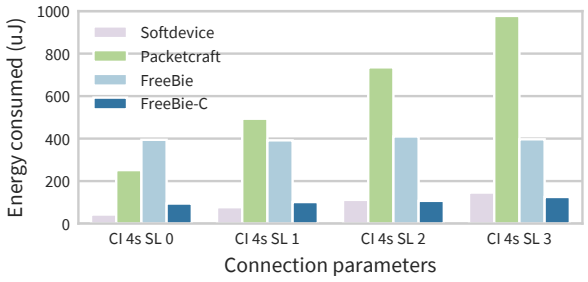

Figure 12: Consumed energy during one Connection Interval (CI), as shown in Figure 5, for four different Slave Latency (SL) values. FreeBie is compared against Packetcraft [77], SoftDevice [71] and a modified version of FreeBie (FreeBie-C) with external memory overhead excluded.

on average restoration is 21.30% faster compared to a naive checkpointing implementation where everything is restored. Due to the reduced overhead, with a more significant application—such as the smartwatch—not only the application-only cycles are 21.39% faster, but also the network-only cycles (by 7.76%) compared to the naive implementation. The average checkpointing times are depicted in Figure 10.

► ***Smartwatch Evaluation:*** Figure 11 shows a 2.5 min long trace of smartwatch operation at each luminosity (except 200 lx, as the screen consumed to much power) from the initial connection establishment. Note that the operating times of FreeBie are very short, shown as a very thin green bar. Nonetheless we see that at each luminosity, FreeBie works despite long power-off intervals. The more energy is available—the smaller the off intervals—the bigger the responsiveness. Zooming in, the execution starts when the storage capacitor voltage reaches 2.6 V. When the system switches on, the voltage drops sharply due to the inrush current and the workload of initialisation but recovers afterwards.

After one round of advertising, FreeBie is connected with the Android BLE host. Note that since the connection was established, FreeBie was turned on continuously for a relatively long time (see 'On/Off' plots underneath the storage plot) and the voltage also dropped significantly. This is caused by Android's initial connection parameters preventing FreeBie from turning off. When the requested connection parameters are applied, FreeBie can start operating intermittently and turn off. For both 300 lx and 600 lx after 50 s and 25 s respectively, when all ATT services were configured and both BLE peripheral and BLE host started sending empty packets, SL is applied which further increases the off time.

► ***Firmware Update Evaluation:*** Figure 13 shows the execution of firmware update at 600 lx. Comparing this figure with Figure 11 (center) (smartwatch evaluation at the same luminosity) FreeBie starts more favourably due to the different initial connection parameters set by the host (as defined in Section 5.1), hence it can keep a higher storage voltage from the start. As expected, in an *application-only cycle* (A), no network process is restored or checkpointed. In *network-only cycle* (B) no application process is executed nor restored. In the *combined cycle* (C) after the network process execution, the system detects the application process pending execution in the near future. Hence the system sleeps until the scheduled time of the application execution, then dynamically loads the application and executes it, after which no processes are scheduled in the near future so the system checkpoints and turns off.

► ***Power Consumption:*** We characterise FreeBie network-only cycles and compare the power consumption of FreeBie to (i) Packetcraft with low-frequency clock enabled and logging disabled, and

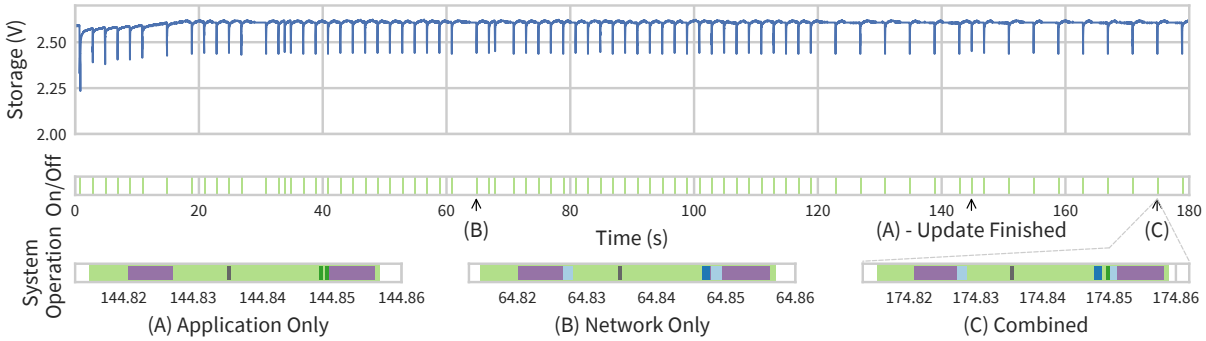

Figure 13: Evaluation of firmware update at 600 lx. Colour scheme: OS restore and checkpoint; on time; application restore and checkpoint; network restore and checkpoint; synchronisation point; network activity.

to (ii) the proprietary Nordic Semiconductor's BLE stack, i.e., SoftDevice [71], at one CI value and four values of SL, i.e., 0, 1, 2, and 3. The results are given in Figure 12.

Thanks to FreeBie's low power consumption when the processor domain is switched off (0.8352 µW) it is **9.5 times more efficient** compared to sleep mode (8.0172 µW with SoftDevice). FreeBie benefits from long connection intervals. Compared to Packetcraft, with a SL of 0 FreeBie's overhead of additional power consumption due to checkpointing and restoration is larger than the power saved by switching the MCU off. However, as the SL increases FreeBie starts to outperform Packetcraft. At a SL of 1 FreeBie already consumes less power compared to Packetcraft. Already with a SL of 3 *we are 2.46 times more energy efficient than the default Packetcraft stack.*

On the other hand, due to FRAM store and restore overhead and FreeBie's requirement to synchronise with the external RTC, FreeBie is not able to compete with SoftDevice's power consumption. Simply, utilising external FRAM consumes large amounts of power and is also slower than even a lower-clocked MCUs's with on-chip FRAM [101]. Ideally a MCU with on-board FRAM or MRAM and a ultra-low power RTC, e.g. the upcoming Ambiq Apollo 4 Blue [4], would remove this overhead almost completely. Therefore to make this comparison we have removed the external memory overhead from FreeBie traces denoting it as FreeBie-C. We see that FreeBie-C outperforms SoftDevice starting at a SL of 2.

Finally, we report the power consumption at each part of the network cycle, including checkpointing and restoration, as shown in Figure 5: FRAM read (restore checkpoint) (Ⓐ) is 10.26 mW, MCU sleep current (Ⓑ) is 2.41 mW, RX current (Ⓒ) is 18.86 mW, TX current (Ⓓ) is 19.35 mW, and FRAM write (checkpoint) (Ⓔ) is 12.03 mW.

► ***Long-term Execution:*** Figure 14 shows 24-hour operation of the FreeBie smartwatch. We see that FreeBie is able to sustain a connection despite power interrupts for extended period of time (see 'On/Off' trace representing FreeBie activity, in particular between 11:00 and 17:00). Moreover, FreeBie is able to sustain a connection in varying energy availability: during the whole experiment the BLE link only had to reconnect seven times. If FreeBie receives more than 300 lx (minimum viable luminosity with the FreeBie LCD powered on) FreeBie is almost always on, only disconnecting when insufficient energy is provided for extended time. Note, if a full day operation of FreeBie is required then increasing the surface area of the solar panels would decrease the minimum viable luminosity threshold, see Figure 14 (top)—increasing the on time of the smartwatch.

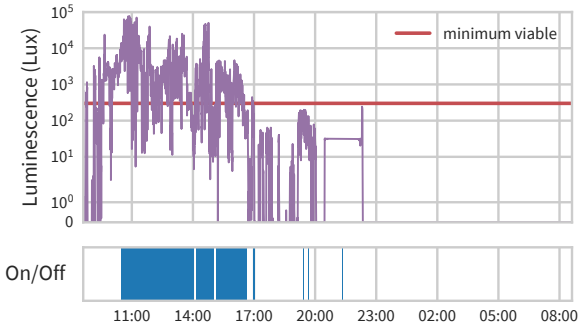

Figure 14: 24 hour-long operation of the FreeBie smartwatch.

6 DISCUSSION AND FUTURE WORK

► ***Hardware Improvements:*** As shown in Figure 12, external FRAM and RTC limit the benefits of our architecture (both in terms of price, size and energy consumption). The next step is a FreeBie version build with next-generation System on Chip (SoC) such as [4] with on-chip MRAM, reducing FreeBie cost/size. More energy-efficient harvesters, such as [76], as part of a complete SoC would make FreeBie not only more efficient but also *potentially cheaper than battery-based systems.*

► ***Battery-free Host:*** In our architecture only the end device is battery-free and intermittently-powered. The next research goal is an intermittently-powered host. The main research challenge would be integrating a synchronisation mechanism such as [41] into a fully battery-free architecture and efficient wake-up scheduling for end devices.

► ***Delay-tolerant Networks:*** One might propose a delay-tolerant network as a solution to the wireless link intermittency problem [55, 64]. Sadly, considering point-to-point links, protocols such

	[25]	[107]	[66]	[92]	[34]	[23]	[113]	**FreeBie**[2]
Size	25 (⌀) × 5.5 mm	4 × 4 cm	20 mm (⌀) [7]	35 × 53 mm	1 × ≈1.5 cm[4]	≈70 × 55 mm [5]	–	2.54 × 2.54 mm
Capactitor size	0.2 F	Unreported	50 mF [8]	200 μF	N/A[3]	1 F [6]	N/A	15 mF
Radio chipset	CYBLE [26]	X-less radio	nRF51822 [73]	CC2650 [100]	Custom	CC2650 [100]	Custom	nRF52840 [74]
Minimum luminosity	100 lx	N/A	N/A	150 lx	N/A	Not reported	N/A	200 lx
Backscatter-based	No	No	No	No	Yes	No	Yes	No
Battery-free	Yes[1]	Yes	Yes	Yes	Yes[3]	Yes	No [3]	Yes
Intermittent	No[1]	Yes	No	No	No	Yes	No	Yes
Advertising only	No	No	No	No	Yes	–	Yes	No
Resumes connections	No	No	No	No	No	No	No	Yes

[1] Supercapacitor; [2] This work; [3] Actual implementation was continuously powered; [4] Modulator only, logic driven by signal generator;
[5] Size inferred from [23, Figure 7]: comparable in size to Arduino Uno board; [6] Maximum value of capacitor bank taken;
[7] Excluding Powercast receiver [82]; [8] Unreported in the paper; a smallest value from Powercast receiver assumed.

Table 5: Comparison of relevant state-of-the-art BLE platforms.

as BLE have strict timing requirements and do not allow any delay beyond what is specified by the standard.

7 RELATED WORK

► ***Semi Battery-Free Wireless Networking:*** Augmenting battery-based IoT with backscatter tags was advocated in [79]. Such systems include [40, 51, 52, 85]. All these networks still need (i) a battery and (ii) a carrier generation source. A separate class of nodes are based on wake-up radios [81]. Wake-up radios still consume power when listening (which increases with receiver sensitivity [81, Figure 12]) and require the same infrastructure investment as backscatter-based radios. An example of battery-free sensors based on proprietary low-power wake-up radio technology is [35].

► ***Battery-Free Wireless Networking:*** Battery-free networks include backscatter-based LoRa [57] and LTE [22]. An alternative approach focuses on active radios and includes [3, 42, 86]. Yet another approach is to perform transformations of already existing protocols to have them failure-resilient [84]. Therein however it was assumed that a node with a power-off had its all memory flushed and needs to initialise from zero. The mathematical analysis of the channel capacity of a intermittent communication point-to-point link is given in [60]. Another way of providing energy to battery-free systems is based on wireless power transfer, recent examples include [53, 70].

Initial studies of duty-cycled bi-directional communication on intermittent power for IEEE 802.15.4-compliant (i.e., non-Bluetooth) CC2420 radio [99] has been proposed in [108]. The same work also proposed to use low-power timing circuit to wake up system to exchange data with a neighbour [108, Figure 3]. Idea of custom protocol state preservation in FRAM has been presented in [19].

► ***Battery-Free Bluetooth:*** All of the battery-free BLE nodes we are aware of do not operate intermittently when considering the BLE protocol itself. In each of such nodes one connection-less beacon transmission can be sent within a single capacitor charge from harvested energy; the recent examples of non-backscatter versions of such a systems are [20, 37, 44, 56, 87, 97] (academia) and [25, 33] (industry). Another (but less popular) approach for battery-free BLE is based on providing power wirelessly to the BLE nodes [66]. Except for our work no studies on intermittenly-powered BLE are presented beyond general calls for such system. A battery-free BLE node of similar hardware architecture to ours, i.e., an RTC-driven MCU with external FRAM, has been presented in [92]. The fundamental difference, however, is that [92] does not allow for state retention of the intermediate state of the BLE protocol (and other applications). Commercial implementations of battery-free BLE include [107]. Refer to Table 5 where a comparison of battery-free BLE platforms is given.

► ***Federated Energy Storage:*** A federated storage power supply architecture [49] 'splits' a large central storage capacitor into smaller capacitors, each powering individual peripherals (such as the radio or a temperature sensor). The federated storage aims at improving system availability. Alternatively, [23] proposed a centralized architecture where one configurable capacitor array serves the entire system, allowing for adding or removing individual capacitors from this array.

► ***Intermittently-Powered Systems Software Frameworks:*** Software supporting intermittently-powered operation have already been comparatively presented in Table 1. Naturally, the list of such systems is only partial and we refer to extensive surveys presented in [11, Table 1], [109, Table 1], [29, Table 2].

8 CONCLUSIONS

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

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their useful comments and Aaron Schulman for shepherding our paper. We would also like to thank John Hendriks for his TU Delft MSc thesis work on a related project, and Nowi B.V. for technical feedback and support. This research was supported by the Netherlands Organisation for Scientific Research (NWO), partly funded by the Dutch Ministry of Economic Affairs, through TTW Perspective program ZERO (P15-06) within Project P1.

REFERENCES

[1] TU Delft Sustainable Systems Lab. 2021. FreeBie Source Code Repository: Hardware and Software Artifacts. https://github.com/TUDSSL/FreeBie. Last accessed: Apr. 26, 2022.

[2] Henko Aantjes, Amjad Y. Majid, Przemysław Pawełczak, Jethro Tan, Aaron Parks, and Joshua R. Smith. 2017. Fast downstream to many (computational) RFIDs. In *Proc. INFOCOM*. IEEE, Atlanta, GA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2017.8056987.

[3] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the MithræUm of Circus Maximus. In *Proc. SenSys*. ACM, Virtual Event, 368–381. https://doi.org/10.1145/3384419.3430722.

[4] Ambiq Micro, Inc. 2018. Apollo4 Blue Ultra-Low Power Microcontroller. https://ambiq.com/apollo4-blue/. Last accessed: Sep. 8, 2021.

[5] Ambiq Micro, Inc. 2021. Artasie AM1815 Real-Time Clock. https://ambiq.com/artasie-am1815. Last accessed: Sep. 8, 2021.

[6] Apache Software Foundation. 2021. Apache Mynewt Operating System Bluetooth Stack Source Code Repository. https://github.com/apache/mynewt-nimble. Last accessed: Aug. 5, 2021.

[7] Konstantinos Arakadakis, Pavlos Charalampidis, Antonis Makrogiannakis, and Alexandros Fragkiadakis. 2022. Firmware Over-the-Air Programming Techniques for IoT networks - A Survey. *ACM Comput. Surv.* 54, 9 (Oct. 2022), 1–36. https://doi.org/10.1145/3472292.

[8] Rauf Arif. 2021. 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. Last accessed: Jul. 6, 2021.

[9] Arm Limited. 2020. GNU Arm Embedded Toolchain (version 9.3.1 (9-2020-q2-update)). https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads. Last accessed: Aug. 19, 2021.

[10] Arm Limited. 2021. Mbed Cordio BLE Solution Official Website. https://os.mbed.com/docs/mbed-cordio. Last accessed: Aug. 5, 2021.

[11] Abu Bakar, Alexander G. Ross, Kasım Sinan Yıldırım, and Josiah Hester. 2021. REHASH: A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 3 (Sept. 2021), 87:1–87:42. https://doi.org/10.1145/3478077.

[12] Jan Bauwens, Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, and Eli De Poorter. 2020. Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles. *IEEE Commun. Mag.* 58, 2 (Feb. 2020), 35–41. https://doi.org/10.1109/MCOM.001.1900125.

[13] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Alert Notification Service. https://www.bluetooth.com/specifications/specs/alert-notification-service-1-0/. Last accessed: Sept. 14, 2021.

[14] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Current Time Service. https://www.bluetooth.com/specifications/specs/current-time-service-1-1/. Last accessed: Sept. 14, 2021.

[15] Bluetooth Special Interest Group, Inc. 2021. Bluetooth Specifications List. https://www.bluetooth.com/specifications/specs. Last accessed: Aug. 8, 2021.

[16] Bosch Sensortec. 2021. BMA400 Triaxial Ultra-low Power Acceleration Sensor. https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/bma400. Last accessed: Sep. 9, 2021.

[17] Rodney Brooks. 2021. The Battery Revolution Is Just Getting Started. IEEE Spectrum, https://spectrum.ieee.org/energy/batteries-storage/the-battery-revolution-is-just-getting-started.

[18] Bradford Campbell, Joshua Adkins, and Prabal Dutta. 2016. Cinamin: A Perpetual and Nearly Invisible BLE Beacon. In *Proc. NextMote Workshop (EWSN 2016 Workshop)*. ACM, Graz, Austria, 331–332. https://www.ewsn.org/file-repository/ewsn2016/331_332_campbell.pdf.

[19] Bradford Campbell, Meghan Clark, Samuel DeBruin, Branden Ghena, Neal Jackson, Ye-Sheng Kuo, and Prabal Dutta. 2016. Perpetual Sensing for the Built Environment. https://doi.org/10.1109/mprv.2016.66. *Pervasive Computing* 15, 4 (Oct.–Dec. 2016), 45–55.

[20] Carlo Signer. 2017. *Batteryless Bluetooth Low Energy Communication*. Bachelor's Thesis. ETHZ, Switzerland. https://pub.tik.ee.ethz.ch/students/2017-FS/BA-2017-03.pdf.

[21] Ricardo C. Carrano, Diego Passos, Luiz C. S. Magalhães, and Célio V. N. Albuquerque. 2014. Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks. *IEEE Commun. Surv. Tutorials* 16, 1 (First Quarter 2014), 181–194. https://doi.org/10.1109/SURV.2013.052213.00116.

[22] Zicheng Chi, Xin Liua, Wei Wang, Yao Yao, and Ting Zhu. 2020. Leveraging Ambient LTE Traffic for Ubiquitous Passive Communication. In *Proc. SIGCOMM*. ACM, Virtual Event, 172–185. https://doi.org/10.1145/3387514.3405861.

[23] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proc. ASPLOS*. ACM, Williamsburg, VA, USA, 767–781. https://doi.org/10.1145/3173162.3173210.

[24] Taiyo Yuden Corp. 2021. EYSKBNZWB BLE Wireless Module. https://www.yuden.co.jp/eu/product/category/module/bluetooth/EYSKBNZWB.html. Last accessed: Sep. 9, 2021.

[25] Cypress Semiconductor Corp. 2020. 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. Last accessed: Aug. 4, 2021.

[26] Cypress Semiconductor Corp. 2021. CYBLE-022001-00 BLE Module. https://www.cypress.com/documentation/datasheets/cyble-022001-00-ez-ble-creator-module. Last accessed: Aug. 10, 2021.

[27] Al Danial. 2021. cloc - Count Lines of Code Source Code Repository. https://github.com/AlDanial/cloc. Last accessed: Aug. 10, 2021.

[28] Jasper de Winkel, Carlo Delle Donne, Kasım Sinan Yıldırım, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In *Proc. ASPLOS*. ACM, Lausanne, Switzerland, 53–67. https://doi.org/10.1145/3373376.3378464.

[29] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. 2020. Battery-Free Game Boy. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3 (Sept. 2020), 111:1–111:34. https://doi.org/10.1145/3411839.

[30] Maurizio Di Paolo Emilio and Roy Anirban. 2021. Macro Environmental Effect of Micro Energy Harvesting. https://www.powerelectronicsnews.com/macro-environmental-effect-of-micro-energy-harvesting. Last accessed: Jul. 27, 2021.

[31] Kristina Edström (Executive Publisher). 2020. 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. Last accessed: Jul. 7, 2021.

[32] Samsung Electronics. 2021. Galaxy Watch4 Smartwatch. https://www.samsung.com/us/watches/galaxy-watch4/. Last accessed: Dec. 14, 2021.

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

[34] Joshua F. Ensworth and Matthew S. Reynolds. 2015. Every Smart Phone is a Backscatter Reader: Modulated Backscatter Compatibility with Bluetooth 4.0 Low Energy (BLE) Devices. In *Proc. RFID*. IEEE, San Diego, CA, USA, 78–85. https://doi.org/10.1109/RFID.2015.7113076.

[35] Everactive. 2021. Batteryless Eversensors. https://everactive.com/batteryless-technology. Last accessed: Aug. 5, 2021.

[36] Duarte Fernandes, André G. Ferreira, Reza Abrishambaf, José Mendes, and Jorge Cabral. 2018. Survey and Taxonomy of Transmissions Power Control Mechanisms for Wireless Body Area Networks. *IEEE Commun. Surv. Tutorials* 20, 2 (Second Quarter 2018), 1292–1328. https://doi.org/10.1109/COMST.2017.2782666.

[37] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications. In *Proc. BuildSys*. ACM, Shenzen, China, 168–171. https://doi.org/10.1145/3276774.3282822.

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

[39] Fujitsu Semiconductor Limited. 2018. 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. Last accessed: Jun. 13, 2021.

[40] Ander Galisteo, Ambuj Varshney, and Domenico Giustiniano. 2020. Two to Tango: Hybrid Light and Backscatter Networks for Next Billion Devices. In *Proc. MobiSys*. ACM, Toronto, ON, Canada, 80–93. https://doi.org/10.1145/3386901.3388918.

[41] Kai Geissdoerfer, Mikołaj Chwalisz, and Marco Zimmerling. 2019. Shepherd: a Portable Testbed for the Batteryless IoT. In *Proc. SenSys*. ACM, New York, NY, USA, 83–95. https://doi.org/10.1145/3356250.3360042.

[42] Kai Geissdoerfer and Marco Zimmerling. 2021. Bootstrapping Battery-free Wireless Networks: Efficient Neighbor Discovery and Synchronization in the Face of Intermittency. In *Proc. NSDI*. USENIX, Virtual Event, 439–455. https://www.usenix.org/system/files/nsdi21-geissdoerfer.pdf.

[43] Shyamnath Gollakota, Matthew S. Reynolds, Joshua R. Smith, and David J. Wetherall. 2014. The Emergence of RF-Powered Computing. *IEEE Computer* 47, 1 (Jan. 2014), 32–39. https://doi.org/10.1109/MC.2013.404.

[44] Andres Gomez. 2020. Demo Abstract: On-Demand Communication with the Batteryless MiroCard. In *Proc. SenSys*. ACM, Virtual Event, 629–630. https://doi.org/10.1145/3384419.3430440.

[45] Google, LLC. 2019. Google Pixel 3a. https://support.google.com/pixelphone/answer/7158570. Last accessed: May 19, 2022.

[46] Google, LLC. 2021. Android 11 Mobile Operating System (with July 5 2021 security update). https://www.android.com/android-11. Last accessed: Aug. 8, 2021.

[47] Google, LLC. 2021. OpenThread Source Code Repository. https://github.com/openthread/openthread. Last accessed: Aug. 10, 2021.

[48] 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. 2021. Research Infrastructure to Power the Internet of Things. https://www.tyndall.ie/contentfiles/EnABLES_Research_Infrastructure_Position_Paper.pdf. Last accessed: Aug. 3,

2021.
[49] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. In *Proc. SenSys* (Nov. 1–4). ACM, Seoul, South Korea, 5–16. https://doi.org/10.1145/2809695.2809707.
[50] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proc. SenSys*. ACM, Delft, The Netherlands, 21:1–21:6. https://doi.org/10.1145/3131672.3131699.
[51] Pan Hu, Pengyu Zhang, Mohammad Rostami, and Deepak Ganesan. 2016. Braidio: An Integrated Active-Passive Radio for Mobile Devices with Asymmetric Energy Budgets. In *Proc. SIGCOMM*. ACM, Florianopolis, Brazil, 384–397. https://doi.org/10.1145/2934872.2934902.
[52] Ivar in 't Veen, Qingzhi Liu, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. 2016. BLISP: Enhancing Backscatter Radio with Active Radio for Computational RFIDs. In *Proc. RFID*. IEEE, Orlando, FL, USA, 1–4. https://doi.org/10.1109/RFID.2016.7488010.
[53] Vikram Iyer, Elyas Bayati, Rajalakshmi Nandakumar, Arka Majumdar, and Shyam Gollakota. 2017. Charging a Smartphone Across a Room Using Lasers. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4 (Dec. 2017), 143:1–143:21. https://doi.org/10.1145/3161163.
[54] Vikram Iyer, Maruchi Kim, Shirley Xue, Anran Wang, and Shyamnath Gollakota. 2020. Airdropping Sensor Networks from Drones and Insects. In *Proc. MobiCom*. ACM, London, United Kingdom, 813–826. https://doi.org/10.1145/3372224.3419981.
[55] Sushant Jain, Michael Demmer, Rabin Patra, and Kevin Fall. 2005. Using Redundancy to Cope with Failures in a Delay Tolerant Network. In *Proc. SIGCOMM*. ACM, Philadelphia, PA, USA, 109–120. https://doi.org/10.1145/1080091.1080106.
[56] Kang Eun Jeon, James She, Jason Xue, Sang-Ha Kim, and Soochang Park. 2019. LuXbeacon—A Batteryless Beacon for Green IoT: Design, Modeling, and Field Tests. *IEEE Internet Things J.* 6, 3 (June 2019), 5001–5012. https://doi.org/10.1109/JIOT.2019.2894798.
[57] Mohamad Katanbaf, Anthony Weinand, and Vamsi Talla. 2021. Simplifying Backscatter Deployment: Full-Duplex LoRa Backscatter. In *Proc. NSDI*. USENIX, Virtual Event, 955–972. https://www.usenix.org/system/files/nsdi21spring-katanbaf.pdf.
[58] Giannis Kazdaridis, Nikos Sidiropoulos, Ioannis Zografopoulos, Polychronis Symeonidis, and Thanasis Korakis. 2020. Nano-things: Pushing Sleep Current Consumption to the Limits in IoT Platforms. In *Proc. IoT*. ACM, Malmö, Sweden, 1–8. https://doi.org/10.1145/3410992.3410998.
[59] Keithley Instruments, LLC. 2021. 2450 SourceMeter Source Measurement Unit Instrument. https://download.tek.com/datasheet/1KW-60904-2_2450_Datasheet_072021.pdf. Last accessed: Sep. 11, 2021.
[60] Mostafa Khoshnevisan and J. Nicholas Laneman. 2017. Intermittent Communication. *IEEE Trans. Inf. Theory* 63, 7 (July 2017), 4089–4102. https://doi.org/10.1109/TIT.2017.2692239.
[61] Daeyong Kim, Junick Ahn, Jun Shin, and Hojung Cha. 2021. Ray Tracing-based Light Energy Prediction for Indoor Batteryless Sensors. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1 (March 2021), 17:1–17:27. https://doi.org/10.1145/3448086.
[62] Koninklijke Philips N.V. 2021. Hue Smart Light Bulb White Ambiance E27. https://www.philips-hue.com/en-gb/p/hue-white-ambiance-1-pack-e27/8718699673147. Last accessed: Aug. 19, 2021.
[63] Vito Kortbeek, Kasım Sinan Yıldırım, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. 2020. Time-sensitive Intermittent Computing Meets Legacy Software. In *Proc. ASPLOS*. ACM, Lausanne, Switzerland, 85–99. https://doi.org/10.1145/3373376.3378476.
[64] Yong Li, Pan Hui, Depeng Jin, and Sheng Chen. 2015. Delay-Tolerant Network Protocol Testing and Evaluation. https://doi.org/10.1109/MCOM.2015.7010543. *IEEE Communications Magazine* 53, 1 (Jan. 2015), 258–266.
[65] Lightricity Limited. 2021. EXL2-1V50 Solar Panel Module. https://lightricity.co.uk/excelllight-exl2-1v50-1. Last accessed: Sep. 9, 2021.
[66] Qingzhi Liu, Wieger IJntema, Anass Drif, Przemysław Pawełczak, Marco Zuniga, and Kasım Sinan Yıldırım. 2021. Perpetual Bluetooth Communications for the IoT. *IEEE Sens. J.* 21, 1 (Jan. 2021), 829–837. https://doi.org/10.1109/JSEN.2020.3012814.
[67] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proc. SNAPL*. Schloss Dagstuhl, Alisomar, CA, USA, 8:1–8:14. https://drops.dagstuhl.de/opus/volltexte/2017/7131/pdf/LIPIcs-SNAPL-2017-8.pdf.
[68] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasım Sinan Yıldırım, Brandon Lucia, and Przemysław Pawełczak. 2020. Dynamic Task-based Intermittent Execution for Energy-harvesting Devices. *ACM Trans. Sens. Netw.* 16, 1 (Feb. 2020), 5:1–5:24. https://doi.org/10.1145/3360285.
[69] Neeru Mittal, Alazne Ojanguren, Markus Niederberger, and Erlantz Lizundia. 2021. Degradation Behavior, Biocompatibility, Electrochemical Performance, and Circularity Potential of Transient Batteries. *Advanced Science* 8, 2004814 (May 2021), 1–26. https://doi.org/10.1002/ADVS.202004814.
[70] Noor Mohammed, Rui Wang, Robert W. Jackson, Yeonsik Noh, Jeremy Gummeson, and Sunghoon Ivan Lee. 2021. ShaZam: Charge-Free Wearable Devices via Intra-Body Power Transfer from Everyday Objects. *ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 2 (June 2021), 75:1–75:25. https://doi.org/10.1145/3463505.
[71] Nordic Semiconductor ASA. 2019. S140 BLE protocol stack (SoftDevice) for the nRF52811, nRF52820, nRF52833 and nRF52840 SoCs. https://www.nordicsemi.com/Products/Development-software/s140. Last accessed: Sep. 9, 2021.
[72] Nordic Semiconductor ASA. 2020. nRF52840 DK BLE, Bluetooth Mesh, Near Field Communication (NFC), Thread and Zigbee Single Board Development Kit for the nRF52840 SoC. https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK. Last accessed: Sep. 9, 2021.
[73] Nordic Semiconductor ASA. 2021. nRF51822 BLE and 2.4 GHz SoC. https://infocenter.nordicsemi.com/topic/struct_nrf51/struct/nrf51822.html. Last accessed: Sep. 9, 2021.
[74] Nordic Semiconductor ASA. 2021. nRF52840 Multiprotocol Bluetooth 5.2 SoC supporting BLE, Bluetooth mesh, NFC, Thread and Zigbee. https://www.nordicsemi.com/Products/nRF52840. Last accessed: Dec. 5, 2021.
[75] Nordic Semiconductor ASA. 2021. Online Power Profiler for Bluetooth LE. https://devzone.nordicsemi.com/power/w/opp/2/online-power-profiler-for-bluetooth-le. Last accessed: Dec. 5, 2021.
[76] NOWI B.V. 2021. NH2D0245 Energy Harvesting Power Management Integrated Circuit. https://www.nowi-energy.com/products-nh2. Last accessed: Sep. 9, 2021.
[77] Packetcraft, Inc. 2021. Packetcraft Protocol Software Source Code Repository. https://github.com/packetcraft-inc/stacks. Last accessed: Aug. 5, 2021.
[78] Arielle Pardes. 2020. The WIRED Guide to the Internet of Things. WIRED, https://www.wired.com/story/wired-guide-internet-of-things. Last accessed: Jul. 6, 2021.
[79] Carlos Perez-Penichet, Fredrick Hermans, Ambuj Varshney, and Thiemo Voigt. 2016. Augmenting IoT Networks with Backscatter-Enabled Passive Sensor Tags. In *Proc. HotWireless*. ACM, New York City, NY, USA, 23–27. https://doi.org/10.1145/2980115.2980132.
[80] Matthai Philipose, Joshua R. Smith, Bing Jiang, Alexander Mamishev, Sumit Roy, and Kishor Sundara-Rajan. 2005. Battery-Free Wireless Identification and Sensing. *IEEE Pervasive Comput.* 4, 1 (Jan.–Mar. 2005), 37–45. https://doi.org/10.1109/MPRV.2005.7.
[81] Rajeev Piyare, Amy L. Murphy, Csaba Kiraly, Pietro Tosato, and Davide Brunell. 2017. Ultra Low Power Wake-Up Radios: A Hardware and Networking Survey. *IEEE Commun. Surv. Tutorials* 19, 4 (Fourth Quarter 2017), 2117–2157. https://doi.org/10.1109/COMST.2017.2728092.
[82] Powercast Corp. 2018. P2110 Powerharvester Evaluation Board. https://www.powercastco.com/products/development-kits/#P2110-EVB. Last accessed: Aug. 10, 2021.
[83] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. https://doi.org/10.1145/1950365.1950386. In *Proc. ASPLOS*. ACM, Newport Beach, CA, USA, 159–170.
[84] David Richardson, Arshad Jhumka, and Luca Mottola. 2021. Protocol Transformation for Transiently Powered Wireless Sensor Networks. In *Proc. SAC*. ACM, Virtual Event, 1112–1121. https://doi.org/10.1145/3412841.3441985.
[85] Mohammad Rostami, Jeremy Gummeson, Ali Kiaghadi, and Deepak Ganesan. 2018. Polymorphic Radios: A New Design Paradigm for Ultra-low Power Communication. In *Proc. SIGCOMM*. ACM, Budapest, Hungary, 446–460. https://doi.org/10.1145/3230543.3230571.
[86] Michel Rotteluthner, Thomas C. Schmidt, and Matthias Wählisch. 2021. Sense Your Power: The ECO Approach to Energy Awareness for IoT Devices. *ACM Trans. Embed. Comput. Syst.* 20, 3 (March 2021), 24:1–14:23. https://dl.acm.org/doi/10.1145/3441643.
[87] Nurani Saoda and Bradford Campbell. 2019. No Batteries Needed: Providing Physical Context with Energy-Harvesting Beacons. In *Proc. ENSSys*. ACM, New York, NY, USA, 15–21. https://doi.org/10.1145/3362053.3363489.
[88] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. 2019. The Computing Landscape of the 21st Century. In *Proc. HotMobile*. ACM, Santa Cruz, CA, USA, 45–50. https://doi.org/10.1145/3301293.3302357.
[89] Seiko Instruments Inc. 2021. CPX3225A752D Chip-type Electric Double Layer Capacitor. https://www.sii.co.jp/en/me/datasheets/chip-capacitor/cpx3225a752d. Last accessed: Sep. 9, 2021.
[90] Sharp Corporation. 2016. LS013B7DH03 1.28" TFT-LCD Module. https://www.sharpsde.com/fileadmin/products/Displays/Specs/LS013B7DH03_25Apr16_Spec_LD-28410A.pdf. Last accessed: Sep. 8, 2021.
[91] Esther Shein. 2021. A Battery-Free Internet of Things. *Commun. ACM* 64, 7 (2021), 16–18. https://doi.org/10.1145/3464937.
[92] Lukas Sigrist, Rehan Ahmed, Andres Gomez, and Lothar Thiele. 2020. Harvesting-Aware Optimal Communication Scheme for Infrastructure-Less Sensing. *ACM Trans. Internet Things* 1, 4 (Oct. 2020), 22:1–22:26. https://doi.org/10.1145/3395928.

[93] Patrice Simon, Yury Gogotsi, and Bruce Dunn. 2014. Where Do Batteries End and Supercapacitors Begin? *Science* 343, 6176 (March 2014), 1210–1211. https://doi.org/10.1126/science.1249625.
[94] Sivert T. Sliper, Oktay Cetinkaya, Alex S. Weddell, Bashir Al-Hashimi, and Geoff V. Merrett. 2020. Energy-driven Computing. *Phil. Trans. R. Soc. A.* 378, 2164 (Feb. 2020), 1–18. https://doi.org/10.1098/rsta.2019.0158.
[95] STMicroelectronics N.V. 2018. X-NUCLEO-LPM01A 1.8 V to 3. V Programmable Power Supply Source (version 2.0). https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html. Last accessed: Sep. 11, 2021.
[96] Jethro Tan, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith. 2016. Wisent: Robust Downstream Communication and Storage for Computational RFIDs. In *Proc. INFOCOM*. IEEE, San Francisco, CA, USA, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524574.
[97] Pietro Tedeschi, Kang Eun Jeon, James She, Simon Wong, Spiridon Bakiras, and Roberto Di Pietro. 2021. Privacy-Preserving and Sustainable Contact Tracing Using Batteryless BLE Beacons. https://arxiv.org/pdf/2103.06221.pdf.
[98] Texas Instruments, Inc. 2012. SN74AUP2G79 Low-Power Dual Positive-Edge-Triggered D-Type Flip-Flop. https://www.ti.com/product/SN74AUP2G79. Last accessed: Sep. 9, 2021.
[99] Texas Instruments, Inc. 2013. CC2420 Single-Chip 2.4GHz IEEE 802.15.4 Compliant and ZigBee Ready RF Transceiver. https://www.ti.com/lit/ds/symlink/cc2420.pdf. Last accessed: May 19, 2022.
[100] Texas Instruments, Inc. 2016. CC2650 32-bit Arm Cortex-M3 Multiprotocol 2.4 GHz wireless MCU with 128 kB Flash. https://www.ti.com/product/CC2650. Last accessed: Aug. 10, 2021.
[101] Texas Instruments, Inc. 2017. MSP430FR59xx Mixed-Signal Microcontrollers (Rev. F). http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf. Last accessed: Sep. 13, 2021.
[102] Texas Instruments, Inc. 2018. OPT3004 Digital Ambient Light Sensor with Increased angular IR Rejection. https://www.ti.com/product/OPT3004. Last accessed: Sep. 9, 2021.
[103] Texas Instruments, Inc. 2018. TPL5111 Ultra Low Power System Timer (35 nA) for Power Gating in Duty Cycled Applications. https://www.ti.com/product/TPL5111. Last accessed: Sep. 9, 2021.
[104] Texas Instruments, Inc. 2021. BQ25570 Ultra Low Power Harvester power Management IC with Boost Charger and Nanopower Buck Converter. https://www.ti.com/product/BQ25570. Last accessed: Sep. 9, 2021.
[105] Uni-Trend Technology Co. Limited. 2020. UT383 Mini Light Meter. https://www.uni-trend.com/html/product/Environmental/Environmental_Tester/Mini/UT383.html. Last accessed: Aug. 19, 2021.
[106] Vishay Siliconix. 2020. SIP32432 Ultra Low Leakage and Quiescent Current and Load Switch with Reverse Blocking. https://www.vishay.com/docs/66597/sip32431.pdf. Last accessed: Sep. 9, 2021.
[107] Wiliot. 2021. Wiliot Battery Free IoT Pixel BLE Tags. https://www.wiliot.com/product/iot-pixel. Last accessed: Jul. 22, 2021.
[108] Lohit Yerva, Brad Campbell, Apoorva Bansal, Thomas Schmid, and Prabal Dutta. 2012. Grafting Energy-Harvesting Leaves onto the Sensornet Tree. https://doi.org/10.1145/2185677.2185733. In *Proc. IPSN*. ACM, Beijing, China, 197–208.
[109] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Pawełczak, and Josiah Hester. 2018. InK: Reactive Kernel for Tiny Batteryless Sensors. In *Proc. SenSys*. ACM, Shenzhen, China, 41–53. https://doi.org/10.1145/3274783.3274837.
[110] G. Pascal Zachary. 2016. The Search for a Better Battery. IEEE Spectrum, https://spectrum.ieee.org/at-work/innovation/the-search-for-a-better-battery. Last accessed: Jul. 7, 2021.
[111] Zephyr Project. 2021. Zephyr Real-Time Operating System Source Code Repository. https://github.com/zephyrproject-rtos/zephyr. Last accessed: Aug. 5, 2021.
[112] Maolin Zhang, Si Chen, Jia Zhao, and Wei Gong. 2021. Commodity-Level BLE Backscatter. In *Proc. MobiSys*. ACM, Virtual Event, 402–414. https://doi.org/10.1145/3458864.3466865.
[113] Pengyu Zhang, Mohammad Rostami, Pan Hu, and Deepak Ganesan. 2016. Enabling Practical Backscatter Communication for On-body Sensors. In *Proc. SIGCOMM*. ACM, Florianopolis , Brazil, 370–381. https://doi.org/10.1145/2934872.2934901.