Seminar Paper: Exploring Bluetooth Low Energy in Intermittently-Powered Wireless Communication Systems - An Introduction

Marvin März
marvin.maerz@fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

ABSTRACT

Bluetooth technology has become ubiquitous in modern devices, with Bluetooth-Low-Energy (BLE) standing out for it's efficiency. A recent breakthrough in research by de Winkel et al. [1] present an architecture for battery-free intermittently-powered wireless communication systems, able to handle power-failures without compromising the BLE specifications. This seminar paper provides a fundamental understanding of this protocol and presents the methodology used by de Winkel et al. in their case study called FreeBie. It examines and compares their approach with previous efforts to achieve similar goals in intermittently-powered environments.

1 INTRODUCTION

In the realm of wireless communication systems, Bluetooth Low Energy (BLE) stands as a pivotal technology, revolutionizing connectivity in, among others, Internet of Things devices. Its significance stems from its low power consumption, widespread support [2] and capacity for secure bi-directional communication.

The conventional reliance on batteries in such Bluetooth systems presents challenges in terms of maintenance, environmental impact and the inevitable issue of aging and disposal. Additional issues arise, regarding weight- or size-optimized BLE nodes, where batteries are frequent limiting factors. Addressing these concerns, the shift towards intermittently-powered systems employing capacitors instead of batteries emerges as a viable solution. They provide a lot of extended functionality but also come with unique challenges of their own. Previous solutions struggled with the multifaceted demands of intermittent wireless communication systems, a complex and largely unsolved challenge. The work by de Winkel et al. [1] marks a transformative breakthrough, offering solutions that exceed prior limitations.

Firstly, they present a novel architecture, introducing a reimagined checkpointing system tailored specifically for intermittent wireless systems. This innovation serves as the core contribution, mitigating the challenges posed by intermittent power and enabling seamless functionality even in energy-constrained environments. Moreover, their implementation and case study, *FreeBie* [open-source code: 3], enables BLE to operate intermittently and bi-directionally. Notably, it sustains established BLE connections even after multiple power failures, paving the way for new battery-free IoT devices and applications.

To enable continuous communication however, BLE is an inherently real-time dependent protocol with little margin for missed deadlines and schedules[, as described in Section 2.1]. In consequence, the **key challenge** is outlined as follows: To adapt such a protocol to intermittently-powered environments, by *checkpointing and restoring* (naturally) volatile connections and system state efficiently in the presence of power failures. Additionally taking into account required system characteristics like *data* and *environmental consistency*, *concurrency* and *energy efficiency*, following a particular *control flow*.

To provide insight into the work done by de Winkel et al., this seminar paper first highlights BLE specifications and complexity, then succeedingly presents and discusses their adapted architecture.

2 BACKGROUND AND PREVIOUS WORK

2.1 BLE Functionality

Bluetooth Low Energy (Bluetooth LE, BLE) was introduced in the Bluetooth 4.0 core specification [4] in 2010 and has since received major improvements in succeeding Bluetooth versions [5, Vol 1, Part C]. It differs from Bluetooth Classic in almost every way: It uses a different protocol stack, i.e. architecture, and different hardware which enables it to be used in ultra-low-power applications, e.g. in IoT devices [6]. As a result, it offers considerably lower power consumption than Classic Bluetooth and is able to operate on very small energy sources like coin-cell batteries at the cost of throughput [2, 7]. It is simply meant for more bursty and lower bandwith data transfers, as most commonly needed in embedded systems, like IoT sensors, relying on the principle to turn on the radio for as short and as rarely as possible to receive and transmit data. The transmission range can be individually configured by application developers to range from only a few meters to over 1 kilometer by using methods for data recovery like Forward Error Correction (FEC) and trade-offs to bandwidth [6].

In consequence, Bluetooth Classic and Bluetooth Low Energy are not compatible with each other, as they make different demands to their respective host or end devices. But luckily, most modern consumer devices support both Classic and Low Energy variants, making for a very broad range of usability and compatibility [6, 2].

BLE operates on the Industrial Scientific Medical (ISM) band, in the 2.4 GHz frequency spectrum. As usual, multiple Bluetooth devices can work in parallel at the same time, so it is important to distinguish between them. That is done by subdividing the spectrum into 40 unique Radio Frequency (RF) channels with a 2 MHz spacing. Data is transmitted via electromagnetic waves within these channels, by using the Gaussian Frequency Shift Keying (GFSK) modulation method [2]. Essentially, an interpretation, i.e. a *key* like "0" and "1", is assigned to two distinct frequencies in each channel. By *shifting* the frequency of the carrier signal according to this keying, binary data can be transmitted and received between devices. Three BLE RF channels are used by end devices to advertise their presence as a connectable device, giving them the name *advertising*

channels, and by host devices to scan for new connections. Data is sent over the remaining 37 *data channels* which are used for bi-directional communication. To avoid collision, interference and noise in the spectrum, as it is also shared between other wireless devices and protocols like WLAN, BLE uses an adaptive frequecy hopping algorithm. The host provides the end device with a specific map of data channels to be used for each connection time-slot, i.e. connection event, and the peripheral follows suit.

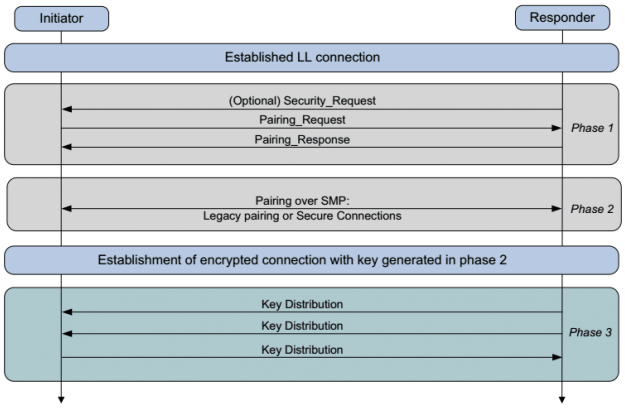

Figure 1: BLE Pairing Flowchart [see 8, Figure 1]

Transmissions can be done either in broadcast style or connection-based, which de Winkel et al. focused on since it had not been done properly before. Connections in BLE are established in multiple phases, during which the roles of both devices change. As described above, end devices announce their presence on the three advertising channels. If a host, the *initiator*, wants to connect to the device, it sends a *Pairing Request* to the advertiser, who may send a *Pairing Response*. Both messages contain information about each devices features and capabilities, as detailed in [8]. Before establishing a connection and transmitting data over the data channels, both devices get to generate an encryption key [9] which is used to securely identify packages [2] and encrypt the connection. On the encrypted link, host and end devices can share their security keys and so establish a secure data transfer, as seen in Figure 1. During this connection establishment period, different connection parameters that dictate various timings, can be negotiated. de Winkel et al. used this to their advantage in adapting their implementation to severely constrained intermittently-powered systems, as described in Section 3.3,

2.2 Motivation And Previous Work

Of course, other ventures in the field of intermittency and wireless communication have been made, but as de Winkel et al. noted, most of them were "infeasable" for the task. This is made clearly in the research overview that Figure 2 provides. The main challenge they happened upon, was that already existing restoration and checkpointing methods were not feasible for wireless communication. Those methods would disturb the protocol's meticulous timing, which in turn would not enable sustained connections. And so there existed no way of recovering a communication after a

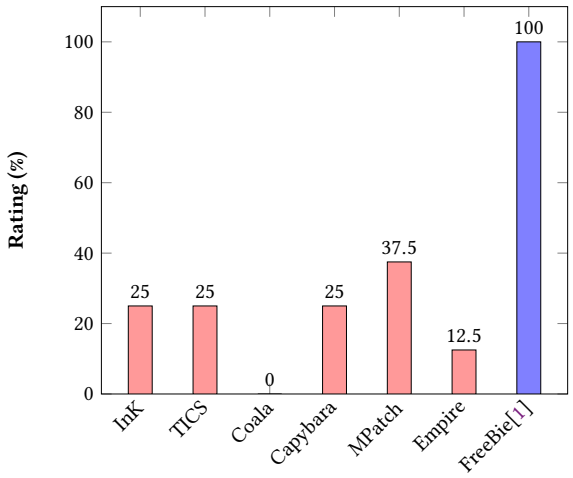

Figure 2: Overview of intermittent BLE implementations [see 1, Table 1]. The more "desired" wireless communication protocol features those systems address, the higher they rank in this score.

power-failure of the intermittently-powered end device. The following reconnection would require many packets to be sent. In the case of FreeBie, the intermittent solution by de Winkel et al., each reconnection required about 70 packets, taking around 40s at the lowest energy supply levels. Other methods like duty-cycling or transmission power control were also recognized as hindering, because of energy consumption problems or only marginal energy savings with too high of an overhead. In already energy-constrained intermittent systems, this huge overhead only drains energy faster, while other applications or network tasks must wait. This results in worse performance and a shorter operating span of the wireless device, neither of which are desirable traits. de Winkel et al. recognized that already proposed methods for checkpointing and restoring the system state are not feasible for wireless networking systems, because they would break the protocol's timing and real-time requirements.

3 FREEBIE: INTERMITTENT ADAPTATION AND IMPLEMENTATION

de Winkel et al. identified two types of frameworks that would be best suited for storing and resuming system operation in terms of memory. Task-based frameworks use tasks and variables to create a state-machine. The problem was that there existed no automated way to implement this efficiently, meaning it would have needed to be done *by hand* on a codebase comprised of around 400 000 lines of code.

A different approach to tasks is a checkpointing-framework where a function is introduced to the original protocol stack that saves the system state until the function (i.e. the checkpoint) is called. A big problem shared by both task-based and checkpoint-based approaches is that they inevitably disrupt the protocols timing because of time-consuming store and restore operations. To

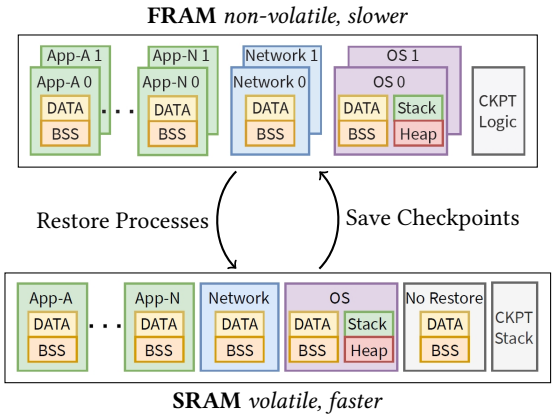

Figure 3: Memory structure [1, Figure 4], process checkpointing and restoration. Checkpoints are double-buffered in non-volatile memory.

nonetheless enable unobstructed communication within intermittently-powered end-devices, de Winkel et al. proposed three major framework changes that mitigate previously encountered problems and shortcomings, which will be addressed in the following Sections.

The main components of the intermittently-powered BLE node, from now on called FreeBie, are its nRF52840 BLE ARM-based microcontroller unit (MCU) [10], the MB85RS4MT fast non-volatile Ferroelectric Random Access Memory (FRAM) [11], and the low-power and high resolution AM1815 Real-Time Clock (RTC) [12]. To power the system, energy is generated by two EXL2-1V50 solar panels [13], harvested by the BQ25570 energy harvester [14] and stored in two parallel 7.5 mF capacitors [15]. A Google Pixel 3a [16] is chosen as the battery-based Android BLE host, dictating some technical specifications and energy requirements of the FreeBie node. The Packetcraft [17] BLE stack which implements all standard Bluetooth layers, was picked as the base of the FreeBie implementation.

3.1 Time-Deterministic Checkpoint (TDC) and Real-time Restoration

The system was adapted to an intermittently powered environment by using Time-Deterministic Checkpoints (TDC) to save the system state in non-volatile memory from which it can be restored following a power failure. In essence, the checkpoint that saves the whole system state can be triggered by the termination of wireless protocol processes. This results in a relatively minimal implementation effort, because no code instrumentation, fixed checkpoint timers or voltage monitors are needed to determine the checkpointing time. Processes and their respective memory need to be separated into three groups: (i) network, (ii) OS, (iii) application. For memory structure, the Packetcraft source code was split by the developers into code directories each representing one process, so that the linker can allocate static memory in different areas during compilation, according to the memory map presented in Figure 3. Processes are also either classified as non-real-time or real-time and restored as such by the scheduler. OS and network processes are, due to their nature, always marked as requiring real-time operation and restoration, whereas application processes can be *either* non-real-time or real-time.

The scheduler schedules the process checkpoints and restoration, and is responsible for deciding whether to place the system into sleep mode or turn off the processor power domain completely, using the virtualisation layer (see Section 3.2). The function **PalSysSleep()** is usually repeatedly called to set the system in sleep mode, awaiting the next process. That function was extended to allow the system to checkpoint and turn off the power to the processor power domain. The checkpointing procedure executed before a power-off is as follows: (1) determine the next power-on time by the virtualisation layer and (2) the real-time processes that then need to be restored, (3) checkpoint the latest process configuration and lastly (4) checkpoint the OS. In the OS checkpoint, processor registers, stack, heap, scheduler and many other important components of the system are saved. Thus application and network process checkpoints must always be committed with an OS checkpoint in order to ensure memory integrity. Each checkpoint is double-buffered in non-volatile FRAM (see Figure 3) to protect the system against faulty states at recovery. Lastly the processor power domain is shut down.

Checkpoint sizes and thus restoration times can vary, mainly due to different sizes of the OS checkpoint, in which all utilized portions of the stack and heap are also committed. So to keep the timing deterministic, a margin is added on top of the varying restoration times and the normalized restoration constant is given as $T_{restore}$ with a value of 10 ms.

The OS scheduler function **wsfOsDispatcher()** was altered in a major way, precisely to distinguish between non-real-time and real-time processes. Non-real-time processes are restored dynamically prior to their execution by the scheduler, whereas real-time processes are always restored at system boot in advance to the scheduler resuming operation. Under the condition that a checkpoint is present, the system restores OS and network checkpoints, as well as those real-time application processes determined prior to power-off. After synchronising to the external time source and compensating for the power-off time, the device resumes normal operation. That means it goes to sleep mode and then executes application or networking processes as dictated by the scheduler. If no checkpoint is available (i.e. during the first boot of the device), the system synchronises to the external time source, the external RTC, and starts operation.

3.2 Virtualisation Layer

To resume operation at a predetermined time, the system is split into two distinguished power domains: (i) processor power domain and (ii) "always-on" ultra-low-power domain to which the external RTC belongs. So when the next process event is scheduled to start later than T_{minOff}, most of the power-consuming system components can be turned off completely, saving a lot of energy. T_{minOff} was determined to be in a break even point between the additional energy cost of checkpointing operations and its obvious energy savings, at a value of 20 ms.

Protocol timing and peripheral state would be disrupted by standard intermittent operation. So to support time-deterministic

restoration as outlined in Section 3.1, de Winkel et al. proposed an abstraction layer that masks the time and peripherals of the intermittent end-device to the host. This is done by employing an external low-power Real Time Clock (RTC) which is always kept on, separately from the processor power domain, and can wake up the system at a predetermined alarm time T_{wakeUp}. It is calculated as follows: $T_{wakeUp} = T_{sync} - T_{startUp} - T_{restore}$, where T_{sync} denotes the *known* time of the next synchronisation with the external RTC and $T_{restore}$ = 10 ms (see Section 3.1. The start up duration $T_{startUp}$ was experimentally found to be a constant value of 10 ms. With those times known, the alarm of the external RTC (belonging to the always-on domain of the system) is set to T_{wakeUp} and the processor power domain can safely power down, because the RTC can control the processor power domain via an external power switch and wake the system up.

Syncing the MCU's RTC with the external RTC is necessary because at power-off, all data from the internal RTC registers is lost. Since T_{sync} is known and saved during checkpointing, it can be used as a compensation value to mask the effects of intermittent operation. Therefore all reads by applications to the internal RTC registers through the virtualisation API will get the compensated and corrected time instead of the raw value.

3.3 Dynamic Network Handling (DNH)

Dynamic Network Handling is necessary for two areas of the system: (i) network recovery and (ii) dynamic network adaptation. Network recovery is needed after a unforseen power failure or when the system was not able to turn back on at a predetermined time. If the Supervision Timeout (ST), the connection timeout parameter set to its maximum of 32s, allows it, missed connection events are then skipped and the network process is scheduled for the next connection event. Lost packages during the power failure will be attempted to be retransmitted by the network stack.

Dynamic network adaptation is responsible for improving the energy-efficiency and performance of the device. It monitors the amount of available energy in the capacitors and tries to adapt, i.e. re-negotiate, the connection parameters, namely Connection Interval (CI) and Slave Latency (SL) accordingly. Connection Interval is denoted as time between the start of two consecutive connection events, in the FreeBie node a value between 1s and 4s. Whereas Slave Latency is an integer number between 0 and 3, defining the amount of connection events which the end device is allowed to ignore [2]. When the system experiences high amounts of energy, CI and SL are requested to be reduced, effectively increasing responsiveness, throughput and overall performance. Complementary, in the case of low energy, the system tries to increase both parameters in order to preserve energy. This comes at the cost of throughput but it enables the device to operate for longer by allowing it to stay in sleep or power-off mode for extended periods of time (determined by the scheduler and virtualisation layer as explained in previous sections 3.1, 3.2).

3.4 FreeBie Evaluation

de Winkel et al. evaluated their FreeBie node with two applications and use-cases: First, they developed a fully functional battery-free smart-watch which was powered by the FreeBie node. Secondly, they also developed a firmware updating application, a first in intermittent wireless communication. To simulate energy levels, especially during a 24 hour benchmark, they first collected sample luminosity values from a sensor outside. The BLE node was then sealed in a light controlled box, where the only light source was a controllable LED light on the ceiling, as a means of imitating sun and weather conditions. de Winkel et al. then continued to evaluate their system's performance under various light intensities and long-time simulations, probing different parts of their architecture. For example, they examined the node's ability to retain a connection, after the main capacitors were empty, i.e. simulating a power-failure. They found that the node did indeed not lose the established connection, if it was powered on again within the ST time window. In comparison to another reference BLE Stack, SoftDevice, their system was up to 9.5 times more energy efficient in idle due to its ability to switch the processor power domain off completely, opposite to the other device which just went into sleep mode.

4 DISCUSSION

As FreeBie evaluation results show, the new intermittently-powered BLE node managed to retain already established connections for extended periods of time, even after multiple power failures. A lot of its longevity features can be attributed to its capability of re-negotiating connection parameters which are more suitable to the current energy state of the device. This, in combination with the time-deterministic checkpointing system and masking of its time and peripherals to the host device, allowed it to have better energy consumption and connection metrics in comparison to the unmodified Packetcraft [17] BLE stack, on which it is based.

However, de Winkel et al. also identified some areas where future improvement to their architecture would be possible to further optimise the energy consumption, cost and size of their BLE node: Using a MCU with *on-board* FRAM (or MRAM) storage and *integrated* RTC, should significantly reduce external memory access and synchronisation overhead. They were able to support this claim, by looking at data where this overhead was removed and could verify that energy consumption was significantly lower [1, Figure 12]. They identified the Ambiq Apollo4 Blue [18] to potentially provide these features, but sadly, this product was then yet upcoming. It has since been, at the time of this paper, released meaning further improvements and iterations to this architecture could be imagined.

5 CONCLUSION

In conclusion, this paper has provided an introduction to Bluetooth Low Energy (BLE) functionality, highlighting challenges in previous intermittent implementations. The unveiling of a fully functional, intermittently-powered BLE node, sets a precedent by shifting from connection-less beacon broadcasts to active connections. This new architecture not only overcomes power failure interruptions but also effectively preserves and restores connections and system state. The evaluation of this case study underscores its significant contribution to wireless intermittent communication, laying a foundation for future innovation.

REFERENCES

[1] Jasper de Winkel, Haozhe Tang, and Przemysław Pawełczak. 2022. Intermittently-powered bluetooth that works. In *Proceedings of the 20th Annual International*

Conference on Mobile Systems, Applications and Services (MobiSys '22). Association for Computing Machinery, Portland, Oregon, 287–301. ISBN: 9781450391856. DOI: 10.1145/3498361.3538934.

[2] Carles Gomez, Joaquim Oller, and Josep Paradells. 2012. Overview and evaluation of bluetooth low energy: an emerging low-power wireless technology. *Sensors*, 12, 9, 11734–11753. DOI: 10.3390/s120911734.

[3] TU Delft Sustainable Systems Lab. 2021. Freebie source code repository: hardware and software artifacts. https://github.com/TUDSSL/FreeBie. Last accessed: Jan. 11, 2024. (2021).

[4] Bluetooth Special Interest Group. 2010. Bluetooth 4.0 Specification List. https://www.bluetooth.com/specifications/specs/core-specification-4-0/. Last accessed: Jan. 17, 2024. (2010).

[5] Bluetooth Special Interest Group. 2023. Bluetooth 5.4 Core Specification. https://www.bluetooth.com/specifications/core54-html/. Last accessed: Jan. 17, 2024. (2023).

[6] Embedded Centric. [n. d.] Introduction to bluetooth low energy / bluetooth 5. https://embeddedcentric.com/introduction-to-bluetooth-low-energy-blueto oth-5/. Last accessed: Jan. 4, 2024. ().

[7] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino, and Domenico Formica. 2017. Performance evaluation of bluetooth low energy: a systematic review. *Sensors*, 17, 12. DOI: 10.3390/s17122898.

[8] Bluetooth Special Interest Group. 2016. Bluetooth pairing part 1 – pairing feature exchange. https://www.bluetooth.com/blog/bluetooth-pairing-part-1-pairing-feature-exchange/. Last accessed: Jan. 17, 2024. (2016).

[9] Bluetooth Special Interest Group. 2016. Bluetooth pairing part 2 key generation methods. https://www.bluetooth.com/blog/bluetooth-pairing-part-2-key-generation-methods/. Last accessed: Jan. 17, 2024. (2016).

[10] Nordic Semiconductor ASA. 2024. Nrf52840 multiprotocol bluetooth 5.2 soc supporting ble, bluetooth mesh, nfc, thread and zigbee. https://www.nordicsemi.com/Products/nRF52840. Last accessed: Jan. 15, 2024. (2024).

[11] Fujitsu Semiconductor Limited. 2018. Mb85rs4mt 4 mb fram memory with spi interface. https://www.fujitsu.com/uk/Images/MB85RS4MT.pdf. Last accessed: Jan. 15, 2024. (2018).

[12] Inc. Ambiq Micro. 2021. Artasie am1815 real-time clock. https://ambiq.com/artasie-am1815. Last accessed: Jan. 15, 2024. (2021).

[13] Lightricity Limited. 2021. Exl2-1v50 solar panel module. https://lightricity.co.uk/excelllight-exl2-1v50-1. Last accessed: Jan. 15, 2024. (2021).

[14] Inc Texas Instruments. 2024. Bq25570 ultra low power harvester power management ic with boost charger and nanopower buck converter. https://www.ti.com/product/BQ25570. Last accessed: Jan. 16, 2024. (2024).

[15] Quarktwin Technology Ltd. 2024. Seiko instruments cpx3225a752d chip-type electric double layer capacitor. https://www.quarktwin.com/product-detail/seiko-instruments-cpx3225a752d/2197598. Last accessed: Jan. 16, 2024. (2024).

[16] LLC Google. 2019. https://support.google.com/pixelphone/answer/7158570. Last accessed: Jan. 16, 2024. (2019).

[17] Inc Packetcraft. 2024. Packetcraft protocol software source code repository. https://github.com/packetcraft-inc/stacks. Last accessed: Jan. 16, 2024. (2024).

[18] Inc Ambiq Micro. 2024. Apollo4 blue ultra-low power microcontroller. https://ambiq.com/apollo4-blue/. Last accessed: Jan. 16, 2024. (2024).

[19] M. Honkanen, A. Lappetelainen, and K. Kivekas. 2004. Low end extension for bluetooth. In *Proceedings. 2004 IEEE Radio and Wireless Conference (IEEE Cat. No.04TH8746)*, 199–202. DOI: 10.1109/RAWCON.2004.1389107.

[20] Nthatisi Hlapisi. [n. d.] Understanding bluetooth le pairing - step by step. https://www.allaboutcircuits.com/technical-articles/understanding-bluetooth-le-pairingstep-by-step/. Last accessed: Jan. 4, 2024. ().