Delft University of Technology
Master of Science Thesis in Embedded Systems

Robust Bluetooth Low Power Communication

Benjamin Joey Hendriks

Robust Bluetooth Low Power Communication

Master of Science Thesis in Embedded Systems

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

Benjamin Joey Hendriks
B.J.hendriks@student.tudelft.nl
Johnhendriks.jr@gmail.com

09-02-2022

Author
Benjamin Joey Hendriks (B.J.hendriks@student.tudelft.nl)
(Johnhendriks.jr@gmail.com)

Title
Robust Bluetooth Low Power Communication

MSc Presentation Date
23-02-2022

Abstract

This work presents the first intermittent connection mode for Bluetooth Low Energy (BLE) on an energy harvesting sensor that maintains a connection during power failures. This is achieved by modifying existing open-source BLE software that runs on solar radiation harvesting BLE hardware architecture designed by [13]. The intermittent operation is achieved by turning off the microcontroller (MCU) between data packets and turn it on right before the exchange of packets. With an external RTC the MCU is switched on and off. Also the external RTC is always powered on during the connection, such that the time can be restored on the MCU if it is turned on again. In order to further reduce the power consumption, an connection interval adaptation algorithm is implemented. This algorithm adapts the data rate to the stored energy of the device. The connection interval adaptation algorithm can reduce the data rate beyond what the current BLE specification allows for and thus reduces the power consumption even more.

The results from the experiments show that the implemented intermittent BLE software can sustain a BLE connection for more than an hour and can be more power efficient than the default non-intermittent BLE software at a low data rate of at least one packet every 10 s or more. Also, at the maximum BLE slave latency measured of one packet every 14 s the intermittent BLE software is almost two times more power efficient. Finally when the connection interval adaptation algorithm was enabled the connection time was further increased.

This work has shown what the implications of an intermittent BLE connection is and how further improvements can be made. The methodology used in this work to run an intermittent connection can be applied to many networking system and can be an inspiration for any future works.

“We absolutely must leave room for doubt or there is no progress and no learning. There is no learning without having to pose a question. And a question requires doubt. People search for certainty. But there is no certainty.” – Richard P. Feynman

Preface

This thesis is written for my master graduation of Embedded Systems track at TU Delft. This topic was of particular interest to me due to the importance of how technology impacts the planet. I hope that this thesis can contribute to a more clean and sustainable future especially with the rapidly upcoming technologies.

First I want to thank Jasper de Winkel and Przemysław Pawełczak for their amazing support. Despite the pandemic and other obstacles they really helped me stay motivated and were always available to contact if I had any questions. Secondly I want to thank my parents John Hendriks, Ellen Hendriks and sister Angela Hendriks for their support and interest throughout the thesis. Also I want to thank my dog Joep for my well deserved breaks and walks that I kept me going for the day. Finally I want to thank Nowi Energy company for their support during this thesis and the hardware they made available.

Benjamin Joey Hendriks

Delft, The Netherlands
9th February 2022

Contents

Preface **vii**

1 Introduction **1**

2 Related Work **2**
- 2.1 Bluetooth Low Energy 2
- 2.2 Timekeeping for Battery-free Platforms 3
- 2.3 Low-Power Devices Powered by Energy Harvesting 4
- 2.4 Intermittently-powered Systems 4
  - 2.4.1 Intermittently-powered Systems: Hardware 4
  - 2.4.2 Intermittently-powered Systems: Software 5

3 Bluetooth Low Energy **6**
- 3.1 The Bluetooth Low Energy Stack 6
  - 3.1.1 Physical Layer 6
  - 3.1.2 Link Layer 7
  - 3.1.3 Host Controller Interface 9
  - 3.1.4 Logical Link Control and Adaptation Protocol 9
  - 3.1.5 Attribute Protocol 10
  - 3.1.6 Generic Attribute Profile 10
  - 3.1.7 Security Manager Protocol 11
  - 3.1.8 Generic Access Profile 11
- 3.2 Intermittent BLE stack 11
- 3.3 Challenges 12
  - 3.3.1 Integrating Intermittent Software With Bluetooth Low Energy 13
  - 3.3.2 Networking Through Power Failures 13
  - 3.3.3 Improving Beyond Bluetooth Low Energy 13

4 Intermittently-Powered BLE Architecture **14**
- 4.1 Battery-free Intermittently-powered BLE Architecture 14
- 4.2 Assumptions 16

5 Implementation **17**
- 5.1 Hardware Design 17
  - 5.1.1 Power Management 17
  - 5.1.2 Microcontroller 18
  - 5.1.3 External Real-Time Clock 19

5.2 Software Design 19
5.2.1 Bluetooth Low Energy Software Stack 20
5.2.2 Checkpointing Software 21
5.2.3 Timekeeping 21
5.2.4 Connection Interval Adaptation Algorithm 25

6 Evaluation Results 29
6.1 Experiment Setup 29
6.2 Power Consumption Measurements 30
6.2.1 Key Insights from Power Consumption Measurements . . 34
6.3 Connection Time Measurements 35
6.3.1 Key Insights from Connection Time Measurements 36
6.4 Connection Interval Adaptation Algorithm Measurements 37
6.4.1 Key Insights from Connection Interval Adaptation Algorithm Measurements 39

7 Limitations and Future Work 43
7.1 Limitations 43
7.1.1 Maybe Long Term Deployment Need To Be Tested? . . . 43
7.1.2 Maybe The Setup Was Limited? 44
7.1.3 Maybe Other Hardware Platform Is Required? 44
7.1.4 Maybe Different Software Platform? 45
7.1.5 Maybe a Hybrid Intermittent Solution Is More Optimal? 45
7.1.6 Maybe BLE Protocol Needs Optimization? 45
7.2 Future Work 46

8 Conclusion 48

Chapter 1

Introduction

The Internet of Things (IoT) refers to billions of connected sensors [16]. Powering all these sensors with batteries will impact the environment and increase the maintenance cost. An answer to this problem is energy harvesting, which utilizes ambient energy to power these small sensors. As a consequence these energy harvesting sensors might operate intermittently i.e., only when enough power is available [14].

Generally-speaking data from energy harvesting IoT sensors is send sporadically, and only when there is enough power available to the sensor. If an energy harvesting IoT sensor would setup a connection, the connection usually lasts until there is no more power in the system [22] or preemptively disconnects before it runs out of power [5]. Losing and reestablishing a connection between power failures is time consuming and ineffective, therefore a method to sustain a communication despite the power interrupt is highly desired.

Bluetooth Low Energy (BLE) is a widely used communication protocol for low-powered sensors. While there are designs available that demonstrate battery-free BLE (i.e., where BLE is broadcasting information as soon as harvested energy becomes available [55, 60, 31, 17]), to the best of our knowledge there is no research conducted on actively maintaining a connection to recover from power failures. A few studies however did implement an active battery-free BLE connection, but the connection was terminated either as soon as energy was not available [22] or when no more power could be harvested [5]. The advantage of maintaining a connection mode for BLE is that devices can communicate peer to peer and in some scenarios a connection can be more power efficient than broadcasting. This could also for example be beneficial for wearable devices that require two-way communication such as a smartwatch or update firmware on hard to reach devices.

Therefore the goal of this thesis is to implement a connection mode for BLE on an energy harvesting sensor in order to maintain a connection during power failures. A custom circuit from [13] is used designed for intermittent BLE.

The structure of this thesis is as follows. Chapter 2 presents the related works. Chapter 3 provides the necessary background on BLE. Chapter 4 describes the architecture of the battery-free BLE sensor, while Chapter 5 describes the architecture implementation. In Chapter 6 the results of the evaluation of the battery-free BLE sensor are shown. Chapter 7 presents the discussion and the recommendations for the future work. Finally, Chapter 8 concludes the thesis.

Chapter 2

Related Work

This chapter will briefly review existing research results related to the thesis. In the following sections we will cover results on BLE, timekeeping, energy harvesting platforms and intermittently-powered platforms. These topics are related to the design of the energy-harvesting intermittently-powered BLE device presented in Chapter 4.

2.1 Bluetooth Low Energy

Most of the research on BLE with energy harvesting devices is centered around beacon applications. BLE beacons are devices that broadcast (advertise) data to any listening device. Work of [55] presented a BLE beacon with solar energy harvesting that focused on measuring power consumption using various transmitted packet sizes. In work [31] a solar harvesting BLE beacon is presented that is tested in different light intensities and also deployed in realistic scenarios. In this paper they chose a big super-capacitor in order to prolong operation time even when there was almost no energy to be harvested. Unfortunately, both [55, 31] lacked power saving techniques to prolong the lifetime of the BLE beacon.

The first work that focused on the quality of service and maximizing the lifetime of the BLE beacon was [17]. In this work the authors were using their own algorithm that turns off most sensors that were not used and changed the advertising interval depending on different levels of power. Another study [6] demonstrated a trade-off between cold booting and sleeping between two BLE advertisements. It shows that when a device can harvest very little power it is more power efficient to cold boot the device and reach higher advertising rates. While if there is plenty of power in the system it is more feasible to sleep between advertisements to achieve a higher advertising rate. However, this study is limited since the data to generate the model is made from theoretical values (datasheets and data from other studies) and there is no experiment to reproduce the results. In [22] the suitability of both advertising and connection mode is determined. Both modes were tested with various light conditions and the power consumption and operation time was measured. It concluded that connection mode is more power efficient in some light scenarios. This study is however limited due to the unrealistic light scenarios during the test (only programmed light levels from a lamp were used). Furthermore no efforts were made for power savings to extend the lifetime or maintain connection during power failures.

In [5] a speedometer is attached to a wheel of a bicycle that generates power (with a magnet and coil) for a BLE device that will setup a connection with a smartphone if enough power is harvested and will terminate the connection when the wheel stops moving. The speed data is send during the connection from the BLE device to the smartphone only when the values changes (notify). This study shows an application specific solution and found an optimal setup for the longest operation time.

Some implementations of battery-free BLE beacons are already on the market. The company Williot [64] has small BLE tags that use Radio Frequency (RF) signals from the environment to transmit specific product data where the tag is attached to. For instance a BLE tag that can measure temperature will be attached on a bottle of wine and will advertise the temperature of the wine periodically to any listening device. The requirement is that there should always be a central device to keep track of the advertising packets and there should always be enough RF signals to keep the tag advertising. Another battery-less BLE beacon that is on the market is from Cypress [54], which uses BLE beacons to send product information of specific items to smartphones or sensor data using solar harvesting.

2.2 Timekeeping for Battery-free Platforms

Having reliable timekeeping for energy-harvesting battery-free devices during power failures is a difficult task since internal timers do not capture power off time. Moreover, a high accuracy millisecond-level timekeeping is necessary for BLE. In [26] three remanence based timekeeping methods are proposed that determine the off time by measuring the decay in capacitive devices. The first method is Time and Remanence Decay in SRAM (TARDIS), which has a poor timing resolution but is able to determine a time indication between power failures from a few seconds up to multiple hours. The second method is called Custom Time And Remanence Decay (CusTARD) that uses a dedicated capacitor to measure the decay and has a higher timing resolution than TARDIS and lower power consumption, but cannot detect long power failures (up to about 40 seconds). The final method is a variation of CusTARD where a low power Real-Time Clock (RTC) is added that has the highest accuracy of timekeeping but also has the highest cost and long reset time (up to several seconds). Only the CusTARD solution with the added RTC might be accurate enough for BLE. In [10] an improved remanence timekeeping method is proposed called Cascaded Hierarchical Remanence Timekeeper (CHRT) and shows that it outperforms an external RTC in startup time and power consumption. The CHRT solution however only has a clock accuracy in the order of milliseconds, which is not sufficient for BLE.

2.3 Low-Power Devices Powered by Energy Harvesting

Thanks to energy harvesting batteries can be removed from low-power devices. This results in long-term and self-sustainable deployment for low-power devices. However, without batteries new implementations of hardware and software have to be made in order to operate in the more uncertain environment. A lot of research on energy harvesting for low-power devices is already done for the architecture [32], sensor nodes [58], harvesting the energy [63, 15] and networking [35].

2.4 Intermittently-powered Systems

When low-power devices rely on energy harvesting instead of batteries, the devices can only operate when enough energy is harvested. This is called intermittent operation. With intermittent operation the device can turn on and off depending on the energy that is harvested. This requires changes in the hardware and software since execution paths of programs can be interrupted due to power failures.

2.4.1 Intermittently-powered Systems: Hardware

Lots of research is done on designing architectures for low-power energy harvesting devices [9, 31, 23, 24]. The most notable differences compared to traditional low-powered devices include energy harvesters, energy storage and the different memory systems.

Energy Harvesters: The most common energy harvesting sources for low-power devices are solar radiation, RF transmissions, pressure, vibration and heat [53]. The choice of the harvester usually depends on the environment and the workload of the device. Solar radiation is known for having a high power density 10 to 100 mW/cm^2 outdoor while RF harvester could harvest up to 1 to 10 mW/cm^2 [53]. However, solar radiation is limited to daylight or artificial light while RF sources are available even during the night.

For a typical BLE device such as the Nordic nRF52840 [43] the power consumption during a radio transmission is about 18 mW (at 6 mA and 3 V). Therefore the most safe choice would be the solar radiation harvesting. This can harvest enough energy for the radio transmissions and is also easy to use for prototyping.

Energy Storage: The harvested energy needs to be buffered in order to have enough voltage and current to power most devices [53]. The energy storage determines how long the device can be turned on. Super-capacitors can charge and discharge energy much faster than regular batteries and are more durable. However, the energy density is much lower than batteries. This results that when energy is harvested it can be accessed much quicker but also depletes faster. Furthermore regular capacitors have less energy density than super-capacitors but are typically smaller and cheaper than super-capacitors [21].

For intermittently-powered BLE typically the most energy is required when the radio need to be turned on. This requires that most energy need to acquired in a short moment of time. Thus super-capacitors are the best choice here due to their fast charge and discharge capability compared to batteries and regular capacitors.

Memory Architectures: In order to keep the data consistent between power failures the data needs to be stored before the device is turned off and restored when the devices turn on. This may be required for data to be further processed or to retain a system state. In works such as [11, 66] non-volatile Ferroelectric Random Access Memory (FRAM) is used. Other non-volatile memory such as flash has limited write cycles in the order of ten thousands, while FRAM can endure trillions of write cycles [61]. Also FRAM has faster access and is generally more power efficient than flash. However, FRAM generally is not as efficient in power consumption and has slower access times than Static Random Access Memory. (SRAM) [36] meaning that changing the memory entirely to FRAM might reduce memory efficiency.

Since the power consumption needs to be as low as possible it is most feasible for intermittently-powered BLE to have FRAM to store its memory contents in before turning off. Since this memory is low power and quick to access the system will not be halted for the memory for to long. Also it is non-volatile and thus will remain the content after power failures.

2.4.2 Intermittently-powered Systems: Software

Due to the intermittent operation on energy harvesting low-power devices the execution of software can be interrupted at any random time and at any place in the code. This can result in lost or corrupted data. There are currently two methods for intermittent operation [36] (i) checkpointing [50, 4, 66, 30] and (ii) task-based programming [67, 25, 37, 8].

Checkpointing systems store the state of the system in non-volatile memory usually at regular intervals, specific areas in the code or when low power is detected. This way when the device will power on it can restore from the checkpoint and will execute from that point forward. The task-based system breaks the program into atomic tasks that are idempotent. These tasks are constructed in a graph-like structure where the data flow from the tasks is determined. During power failures the tasks can then be restarted when required depending on the constraints that are put on the task. Checkpoint systems are straightforward and it is not difficult to expand existing software with checkpoints. However, the drawback of checkpointing is the memory overhead due to big pieces of data have to be stored. On the other hand porting software to atomic and idempotent tasks gives more control of the system. However, the porting of already existing software is very complex and time consuming [36, 34].

Currently there is no intermittent BLE software stack and to modify or create one in a task-based system would be near impossible for this thesis. The checkpoint system is much more interesting to use for intermittently-powered BLE, mainly because checkpoints can be added to the existing software without changing the code drastically. Most of the code thus remains intact and do not need to be split op in different tasks. Furthermore when the software needs to be changed or different functionalities are added the checkpoint system remains roughly the same, while with task based systems a whole range of tasks might need to be adjusted.

Chapter 3

Bluetooth Low Energy

This chapter describes the necessary background information of BLE and the challenges of powering BLE intermittently. In Section 3.1 the BLE stack architecture is described. In Section 3.2 explains how BLE can be powered intermittently without the penalty of re-initializing a connection. Finally in Section 3.3 the challenges of designing an intermittently-powered BLE device is described.

3.1 The Bluetooth Low Energy Stack

The BLE protocol stack is composed of three parts: the *controller*, the *host* and the *application* [62]. Between the *controller* and *host* is the Host Controller Interface (HCI). The *controller* part is usually a physical device that facilitates the transmission and reception of the BLE signals and contains (i) Physical Layer (PHY), (ii) Link Layer (LL) and (iii) HCI. The *host* part is typically a software stack that specifies how BLE data is constructed and BLE devices interact. It contains (i) Logical Link Control and Adaptation Protocol (L2CAP), (ii) Attribute Protocol (ATT), (iii) Generic Attribute Profile (GATT), (iv) Security Manager Protocol (SMP) and (v) Generic Access Profile (GAP). The *application* part consists of the logic, user interface and data handling which all relate to what the application was implemented for. An example would be an app that can interact and interpret a heart rate value from a connected BLE sensor. In Figure 3.1 the BLE protocol stack is illustrated. Each layer of the BLE stack is described in Sections 3.1.1 to 3.1.8.

3.1.1 Physical Layer

The Physical Layer contains the hardware that is capable of sending and receiving BLE data. The BLE radio operates in the 2.4 GHz Industrial Scientific and Medical band. This band is divided into 40 radio frequency channels, each channel with a bandwith of 2 MHz. These 40 channels are divided in 3 advertising channels (channels 37, 38 and 39) and 37 data channels (channels 0 to 36). The radio uses Gaussian Frequency-Shift Keying modulation with a transmit output power between −20 to 10 dBm [7]. The maximum data-rate for Bluetooth 4.2 is 1 Mbit/s and for Bluetooth 5.2 it is 2 Mbit/s [20, volume 1, page 187].

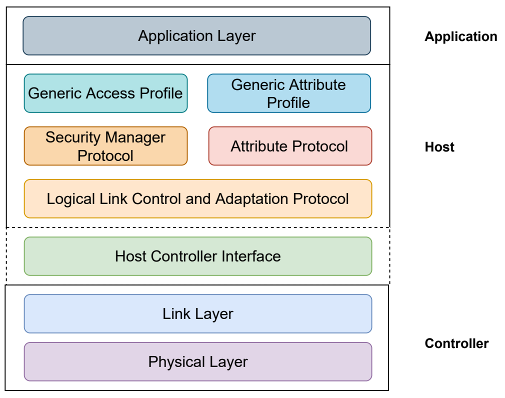

Application Layer
Application
Generic Access Profile
Generic Attribute Profile
Security Manager Protocol
Attribute Protocol
Host
Logical Link Control and Adaptation Protocol
Host Controller Interface
Link Layer
Physical Layer
Controller

Figure 3.1: **The BLE architecture [62].**

3.1.2 Link Layer

The Link Layer controls the Physical Layer and handles the state of the communication for BLE devices. There are three advertising channels used for broadcasting data and device discovery in order to establishing a connection. The broadcasted data is typically send on all three advertising channels to avoid interference with other traffic sources like Wi-Fi.

The data channels are used for bidirectional communication between connected devices. There are 37 data channels and data is exchanged on one channel each time. The host and end device constantly switch channels on which they communicate via adaptive channel hopping, reducing the interference from other signals. The basic formula defining the next hop is:

$$f_{n+1} = f_n + h \mod 37$$

Where h (denoting channel hop) is between 5–16, f_n is the current channel number and f_{n+1} the new channel number. The hop number h is randomly chosen at the start of the connection. The adaptive frequency hopping scheme makes it possible to let devices determine that channels with to much interference can be removed from the channel list. This will be explained in more detail in Section 5.2.4.

At the Link Layer there are two types packets: advertising packets, sent through the advertising channels, and data packets sent through the data channels. Both advertising packets and data packets have the same packet structure as shown in Figure 3.2. The maximum packet size can be up to 266 octets. The preamble field is usually one octet long but can be two octets long when communicating on 2 Mbit/s. The access address field is fixed for advertising packets with a constant value, while in data packets it contains the address of the receiving device. The header field contains specific information about each packet type, which is explained in more detail in [20, volume 6, page 2865]. The length field indicates the length of the packet. The payload field contains the actual data that has to be send and is specific for each data type but could contain, for example, a sensor value or information to establish a connection. The Cyclic Redundancy Check (CRC) field is a 24 bit error-detecting code [20, volume 6, page 2865].

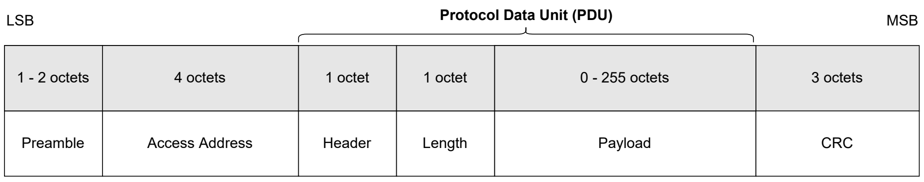


Figure 3.2: **BLE Link Layer packet [20, volume 6, page 2865].**

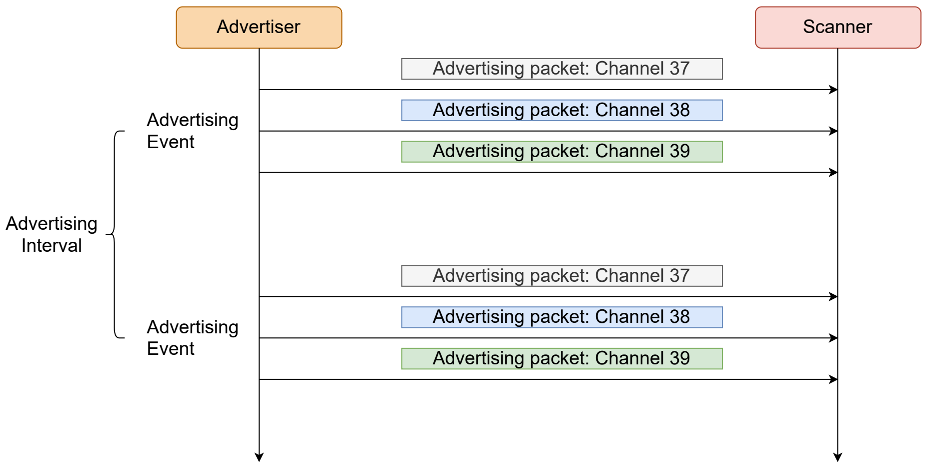

Advertiser
Scanner
Advertising packet: Channel 37
Advertising packet: Channel 38
Advertising packet: Channel 39
Advertising Event
Advertising Interval
Advertising packet: Channel 37
Advertising packet: Channel 38
Advertising packet: Channel 39
Advertising Event

Figure 3.3: **Unidirectional exchange of advertising packets from advertiser to scanner [27].**

There are 4 different communication states handled in the Link Layer (also knows as roles): advertiser, scanner, master and slave. These four roles can be divided into two pairs: (advertiser, scanner) and (master, slave). We will describe them below.

Advertiser and Scanner

The advertiser and scanner both send and receive data with advertising packets on the advertising channels. These advertising packets are send periodically with the predefined advertising interval. For BLE the advertising interval ranges from 20 ms to 10.24 s [20, volume 4, page 2483]. Typically the advertiser sends data and the scanner listens to one of the three advertising channels. In Figure 3.3 the exchange of advertising data from advertiser to scanner is shown. Note that the advertiser does not have any feedback from the scanner if the data is received or not.

The advertiser can have three properties: (i) connectable: the advertiser is able to connect to other devices (ii) scannable: the scanner is able to request additional data (iii) directed: a special type of packet for faster connecting where the target Bluetooth address is in the payload. An advertiser can have a combination of these properties or none [62]. When the advertiser is connectable the scanner can respond to an advertising packet from the advertiser, which then acts as a device discovery packet, with a connection request. If the advertiser is scannable, the scanner can send a scan request and the advertiser can then send scan response data. The scan response data contains more information about the advertiser, for example the device name.

Master and Slave

The master and slave both send and receive data with data packets on the data channels. Before a typical connection is made, a scanner will send a connection request on one of the three advertising channels as a response to the advertising packet. If the advertiser accepts this request, the two devices have a connection. The scanner that initiates the connection will be the master of the connection, while the advertiser that accepts the connection will be the slave of the connection. The master will manage the connection and determine which hop frequency is used. The slave follows the masters decisions. Each exchange of data in a connection is called a connection event and the time between these events is called the connection interval. Note that multiple data packets can be transferred during a connection event. The connection interval is initially set by the master. The slave can request a different connection interval, but the master could decide to reject this request. The connection interval ranges from 7.5 ms to 4 s with increments of 1.25 ms. In Figure 3.4 the connection establishment between master and slave is shown.

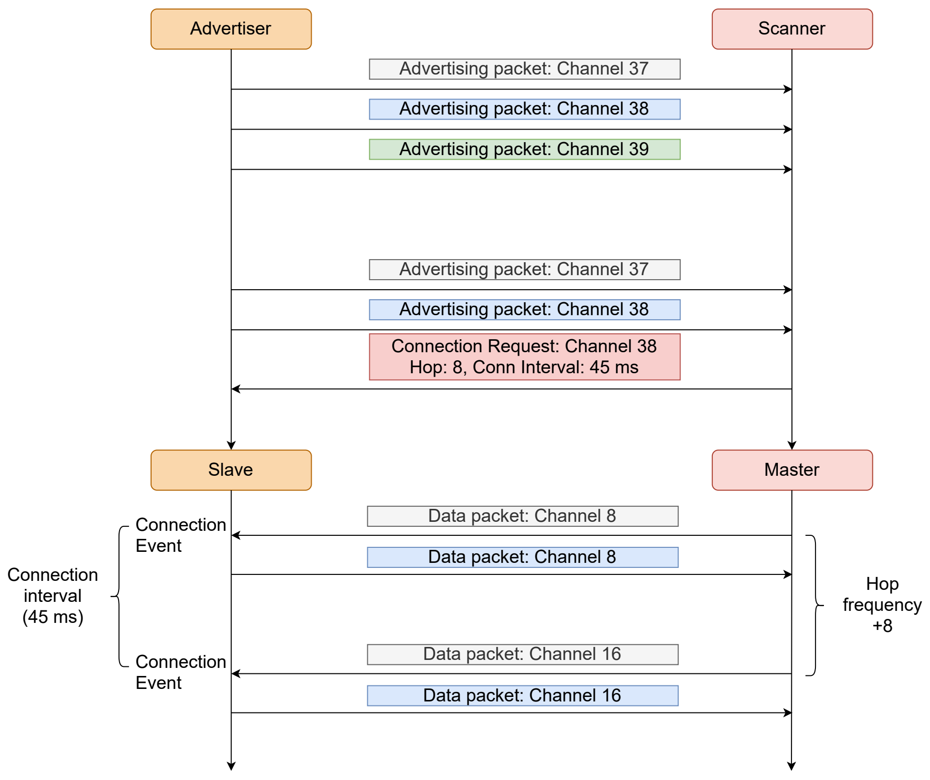
The connection can be terminated by both master and slave via a connection terminate packet or exceed the Connection Supervision Timeout. The Connection Supervision Timeout is the maximum time between two received data packets before the connection is terminated. This timeout ranges from 100 ms to 32 s. The Slave Latency is the number of skippable connection events from the slave when there is no data required to send. For example with a slave latency of two, the slave can skip two connection events if no data is available to send. If there is data to be sent it will not skip any connection event. The Slave Latency value ranges from 0–499, if it is within the Connection Supervision Timeout. The Connection Supervision Timeout and Slave Latency both are initially determined by the master via the connection request.

3.1.3 Host Controller Interface

The HCI facilitates the communication between the host layer and the controller layer from the BLE architecture. For systems where the host and controller are each executed on a different chip the communication between these layers is done via a physical medium, most commonly USB or UART. This was more common in older Bluetooth classic designs. With cheaper System on Chip designs the host and controller are more commonly integrated in one chip and the HCI becomes redundant.

3.1.4 Logical Link Control and Adaptation Protocol

The L2CAP combines data from ATT and SMP to create a BLE packet. The L2CAP fragments large data into chunks to fit them in the BLE packets. At the receiver side the L2CAP combines fragmented packets in order to construct the data for the upper layers.



Advertiser
Scanner
Advertising packet: Channel 37
Advertising packet: Channel 38
Advertising packet: Channel 39
Advertising packet: Channel 37
Advertising packet: Channel 38
Connection Request: Channel 38
Hop: 8, Conn Interval: 45 ms
Slave
Master
Connection Event
Data packet: Channel 8
Data packet: Channel 8
Connection interval (45 ms)
Hop frequency +8
Connection Event
Data packet: Channel 16
Data packet: Channel 16

Figure 3.4: **Example packet exchange between master and slave during initiating connection with connection interval of** 45 ms **and a hop-frequency of 8 [27].**

3.1.5 Attribute Protocol

The ATT is a protocol for facilitating the exchange of data between two BLE devices. This data is stored in a specific structure called an attribute. An attribute is stored on a server, which is the device that contains the data. The device that wants to access this data is called the client. Both master and slave can be a server and client simultaneously, meaning both can store and request data from each other. For example an attribute can store a the heart rate value in beats per minute on a server, and a smartphone (client) can request this data.

3.1.6 Generic Attribute Profile

GATT defines how data is stored and transferred. The data is stored in a hierarchy where characteristics are stored in services and services are stored in profiles. Each profile contains services which is a bundle of specific data called characteristics. The service is stored in an attribute which describes the kind of data stored. The characteristics is also an attribute which contain specific information of each type of data like the actual sensor value.

For example the Bluetooth SIG has predefined a Heart Rate profile that contains different services for heart rate monitors. One basic service is the Heart

Rate service. It is defined such that this Heart Rate service has a Heart Rate Measurement characteristic that stores the heart beat value in beats per minute, Body Sensor Location characteristic that stores the location of the monitor on the body, like wrist or foot, and Heart Rate Control Point characteristic which is used to enable a client to write control points to the server to control behavior [28].

3.1.7 Security Manager Protocol

SMP is a protocol that makes it possible for devices to pair and distribute different type of security keys. Some data values (characteristics) require encryption before they can be send. Typically if data needs to be encrypted, two devices pair by authenticating each other. Then the packets will be encrypted using a short term key. This short term key is configured by the temporary key and with two random values from each device. The temporary key can be configured in three ways (i) "just works", which is basically a zero value that is used when there is not interface available to authenticate (ii) "passkey entry", where via a pin-code the authentication is done (iii) "out of band", when both devices have authentication data that is acquired outside BLE for example via Near-Field Communication (NFC) authentication. Finally when pairing is done the long term keys are distributed to have faster authentication when both devices are reconnected later [27].

3.1.8 Generic Access Profile

GAP defines how BLE devices can discover and communicate with each other. For example with GAP it is possible for a smartphone to connect to a BLE sensor and have them interact with each other. For GAP four roles are defined: (i) broadcaster uses the LL advertiser role and transmits advertising packets, (ii) observer uses the LL scanner role which receives and collects data from advertising devices, (iii) central uses the LL master role and is the device that can connect to one or more devices, and (iv) peripheral uses the LL slave role that sends out advertising packets in order to receive a connection request from a central and connect to this central. Furthermore GAP describes modes for the roles, which is a state where the device execute a certain procedure. For example a broadcast mode in which a device will advertise packets. The procedure is a specific set of actions to reach a goal. This can be a specific set of data packets in order to update a connection interval. When a device is in a specific GAP mode there can be a procedure coupled to it and vice versa but it can also be standalone [62].

3.2 Intermittent BLE stack

To maintain a connection with the BLE stack with intermittent operation the LL needs to be integrated with the software stack supporting intermittent operation. The LL is the only layer in the BLE stack that operates in real-time. During a connection both the master and slave have to exchange packets in order to maintain the connection. It is allowed to skip some connection intervals and still maintain the connection, if it is within the Connection Supervision

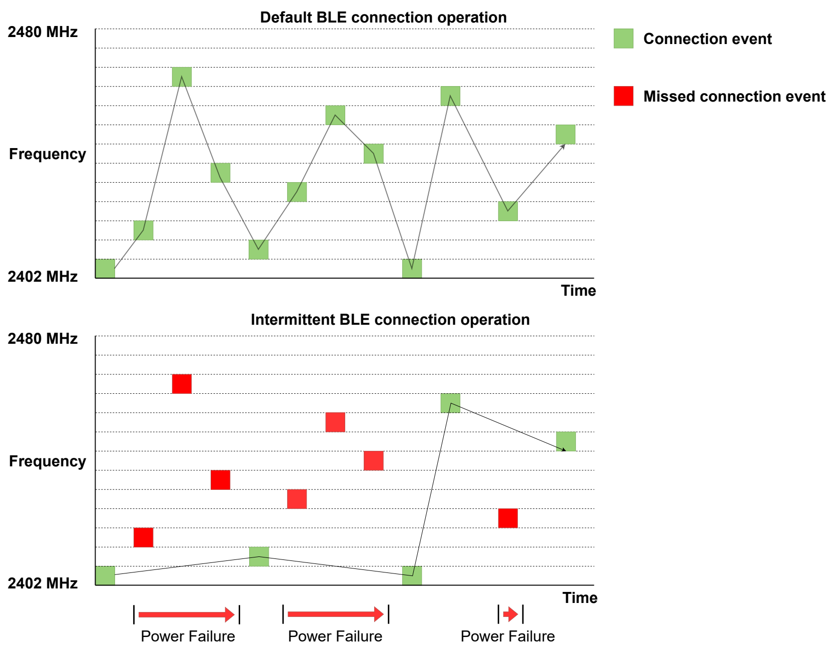


Figure 3.5: **Example depiction of the exchange of packets in the form of connection events between a master and slave. Top figure shows the default BLE connection operation, where each connection event has a successful exchange of packets. Bottom figure shows the intermittent BLE connection operation, where due to power failures some connection events are skipped. Note that the slave device must know how much time each power failure takes in order to configure the right radio frequency to continue the connection.**

Timeout. If due to a power failure a connection interval is missed, the slave has to know how many connection intervals are missed because of the adaptive frequency hopping that changes the channel frequency after each interval. It then must determine how many connection intervals have been skipped and, if it is within the allowed time, set the frequency accordingly. That is why timekeeping in battery-free BLE is necessary even when the system has power failures. Additionally when the slave is in a connected state it has to get back in the connected state without having to reestablish the connection with the master. In Figure 3.5 an example is shown where the importance of timekeeping is shown. The figure illustrates that if a device boots up from a power failure during a connection it has to know how long it was powered off. If this is not the case it becomes impossible to know what frequency the radio has to listen to and when it has to turn the radio on in order to continue the connection.

3.3 Challenges

Integrating BLE with software framework supporting intermittent operation to maintain a BLE connection on a BLE slave device during power failures has several challenges that needs to be addressed. In the following sections we list these challenges.

3.3.1 Integrating Intermittent Software With Bluetooth Low Energy

The first, implementation-related, challenge is to find and understand an open-source BLE software stack in order to implement the necessary features to overcome power failures. Especially access to the Link Layer in the BLE software stack is required due to access to the timing and frequency when recovering from power failures during connection. The BLE software stack has to be preferably bare-metal, since saving the state of a complex operating system will only add more complexity.

When a power failure has occurred it is important that the state of the connection can be recovered and that the peripherals are configured correct in order to continue to operate. The intermittent operation software has to be integrated in the BLE software stack but must not interfere with the critical sections that would hinder the regular (non-intermittent) BLE operations.

3.3.2 Networking Through Power Failures

The second challenge is to maintain a connection during power failures. A notion of time after recovering from a power failure is required to be able to communicate on the right frequency, and to know at which moment the slave device must turn on the radio to receive a packet from the master device. This exact timing requires a way to synchronize the clocks when the device is turned on again. Also the state of the connection has to be maintained, which means the right procedures have to be followed even during power failures in order, for example, to establish a connection.

3.3.3 Improving Beyond Bluetooth Low Energy

For the final challenge the amount of energy available to the system is an important aspect when BLE runs intermittent. For a connection it is essential to have good power management such that the connection can be maintained. Between connection events a BLE slave typically sleeps. Keeping the device in sleep mode can consume significant power if the interval is in the order of seconds. It might be more power efficient to turn off the device if long sleep periods are expected. However, saving the state and recovering from a power failure also adds overhead which should be considered as well.

BLE has a power saving technique called slave latency, that will skip certain radio events when no data is pending to be send. The slave latency is limited by the BLE specification to be not bigger than half of the supervision timeout. This way there must be at least two connection events before the supervision timeout is reached, which will reduce the likelihood of disconnecting. However, more power could be gained by turning the device off for longer periods of time than the slave latency if it is withing the supervision timeout. Furthermore the slave latency is a connection parameter which also limits the flexibility during a connection to change the time it has to be turned off, because new connection parameters have to be negotiated between the master and slave.

Chapter 4

Intermittently-Powered BLE Architecture

We now present the architecture of our battery-free intermittently-powered BLE system. Our architecture is based on the hardware design of [13]. It is important to note that the work of [13] originated from another MSc project of [59] on the same topic of battery-free intermittently-powered BLE system. The software architecture presented in this thesis has been developed independently from [59] and no interaction between the author of this MSc thesis and the author of [59] happened. In other words the hardware design is not part of this thesis, only the software implementation. The motivation of why this design was chosen and the architecture of the design is described in Section 4.1 while Section 4.2 describes the assumptions that are made for this project.

4.1 Battery-free Intermittently-powered BLE Architecture

To address the challenges listed in Section 3.3 as remarked above, we chose the hardware of [13] that allows to sustain bi-directional BLE communication despite power failures. In Figure 4.1 the high-level system design is shown. Note that the hardware of the external sensors are present in the hardware but are not used and are not further discusses for this thesis.

In the hardware architecture solar harvesting is used which is a reliable and power-dense source [53]. Also light is an easy power source to control, that is beneficial for experimenting. Finally it is easy to add more solar panels if this is required, which will prevent this solution from not being capable of harvesting enough power. The harvested solar energy is boosted by a DC-DC converter and stored in a super-capacitor, which powers the components.

The microcontroller (MCU) runs an open source BLE software stack with access to all BLE stack layers. Additionally the MCU has BLE hardware integrated, that is able to send and receive BLE packets. The MCU will also run dedicated checkpointing software to store and restore the internal state on FRAM. FRAM is a fast and durable (with many write cycles) non-volatile memory that is able to quickly store and restore data. Checkpointing is chosen over task-

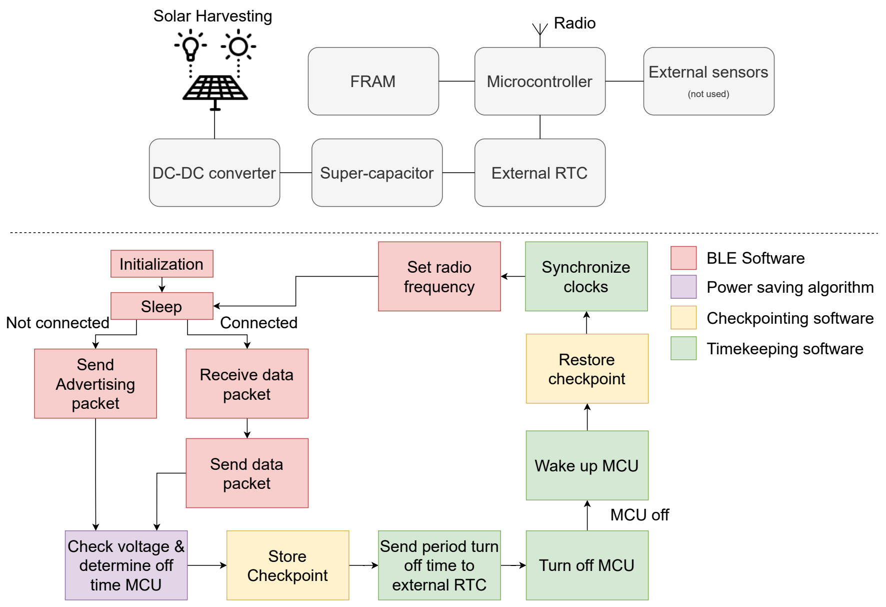


Figure 4.1: **High-level battery-free intermittently-powered BLE system design: (top) hardware (based on [13] and (bottom) software.**

based systems due to the lower complexity. The first challenge of integrating the intermittent software with BLE is addressed here because the checkpointing software is integrated with the BLE software and the whole system state will be stored in FRAM when the MCU will be turned off.

Having a high accuracy clock on BLE devices will reduce the receive window. The receive window is the time that the BLE radio is turned on while receiving data. This receive window is determined by both the master and slave clock accuracy. Ideally the radio should be turned off as much as possible, since the power consumption of the radio is high. This is why the external RTC is chosen over the remanence timekeeping, because it has a higher accuracy clock. The idea, originally proposed by [33], is that the external RTC is always powered on during operation and can switch the power of the MCU on or off. The external RTC is always on because it has a long startup time (order of seconds). The MCU will communicate with the external RTC and determines when to turn off and for how long. This way the MCU can be turned off during long connection events or sleep times and wake up in time to commit to the connection. The timekeeping will be performed by the external RTC and monitored by the MCU. This way the timing on the external RTC is used as a reference point to synchronize the internal clock of the MCU. The solution with the external RTC addresses the second challenge of networking through power failures.

In order to operate intermittently, two features will be implemented on the MCU: (i) the MCU will be turned off between connection intervals via the external RTC and (ii) the MCU must skip multiple connection intervals when low power is detected to be able to harvest more energy. The assumption is that the sleep function compared to turning off the MCU will consume considerably more energy. Note that here the connection interval will not be changed, but some connection intervals will be skipped to reduce the data rate. This is done because changing the connection parameters requires that the master and slave device negotiate on the new connection interval. This can take several connection intervals in order to be negotiated, which is to slow. By skipping connection intervals data rate is slowed down without the need for changing the connection interval. Also, BLE does support this feature itself which is called slave latency (described in Section 3.3.2). Skipping connection intervals will be different than the slave latency though, since the connection interval skips will be dynamically determined based on the energy in the system. The slave latency value is also a connection parameter value, and thus need to be negotiated between the master and slave device. By skipping multiple connection intervals the third challenge is addressed by improving beyond BLE.

The implementation of the intermittent BLE connection by turning off the MCU and skipping connection intervals to save power will add overhead because of storing a checkpoint, restore the checkpoint, synchronizing clocks and restoring the time. This is because the storing and restoring of the checkpoint needs to be done to the external FRAM. Furthermore, with the external RTC the time must synchronized and mapped to the internal clock on the MCU in order to make use of the external RTC time. The idea is that the external RTC will be the main clock of the system where all timing for the BLE stack will be based upon. To conclude, we preemptively turn off the MCU between connection intervals instead of waiting for the power to deplete. The advantage of this approach is that during the power off time the device can harvest energy. The disadvantage is that there is overhead added to each connection event.

4.2 Assumptions

In this thesis some assumptions are made in order to keep the project manageable. The above architecture will be implemented for a BLE slave device that will connect to a powered (i.e. non-battery free and non-intermittently-powered) master device. In addition there will be no fundamental changes applied to the BLE stack such that the slave device is not compatible with other BLE master devices. This means that there are no hacks made in packet structure or changes in certain protocols which need a specific modded BLE master to be able to connect. Furthermore, the external RTC is the component that powers the MCU and FRAM on and off. This mean the external RTC will be continuously powered during the connection. The device and software implementation is able to maintain the connection intermittent for at least one hour when powered. This is to ensure that the added intermittent software is stable.

Chapter 5

Implementation

This chapter recapitulates the description of the hardware design we have used in this thesis (and which was presented first in [13]) and describes the implementation of the software that maintains timekeeping and time synchronization to sustain a BLE connection on an energy harvesting device. Section 5.1 describes the hardware design. In Section 5.2 describes the software design.

5.1 Hardware Design

In Figure 5.1 the photo of the early fabricated board from [13], used by us for the software implementation in this project, is shown. In the following sections each component of the presented hardware will be described in more detail.

5.1.1 Power Management

The Nowi NH2D0245 energy harvesting Power Management Integrated Circuits (PMIC) [46] is a DC-DC converter that extracts the low power output from the solar panels [65] using a rectifier. It stores this energy in two Seiko Instruments CPH322A super-capacitors [29] of 11 mF each. The NH2D0245 only operates at 2.5 V and will therefore not boost anything from the solar panel when the voltage drops under this value. This means that if the voltage reaches below this point the whole system will shut down and most likely not recover, since the super-capacitors will not recharge if the DC-DC converter is disabled. The two super-capacitors are placed in parallel powering the Ambiq AM1805 external ultra-low power RTC [1], which is always powered on during operation. The external RTC switches the power of the EYSKBNZWB BLE module [68]. The EYSKBNZWB is build around the Nordic nRF52840 Bluetooth System-on-Chip [43] MCU. The MCU is connected with the Fujitsu MB85RS4MT 512 kB FRAM [18]. The FRAM is connected to the same power source as the MCU, meaning that if the MCU is shutdown by the external RTC it also shuts off the FRAM. Finally an over-voltage protection is added when the voltage from the super-capacitor is over 3.3 V to protect the internal hardware.

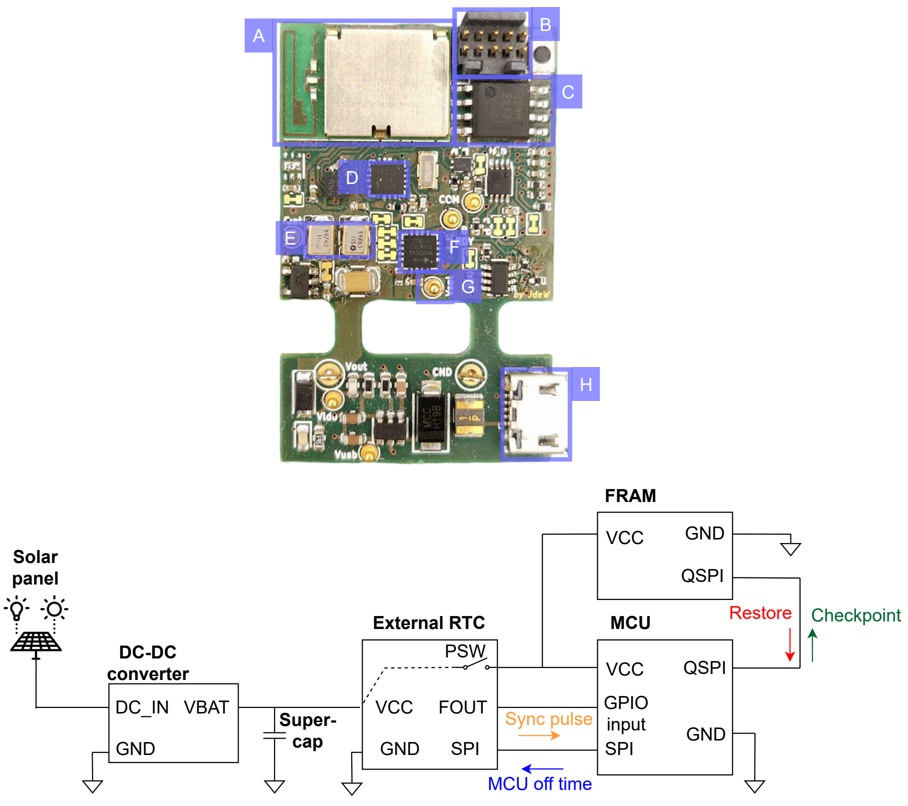


Figure 5.1: **Hardware overview of the system used in our implementation [13]: (top) Photo of the used board (version v1.4): (A) EYSKBN-ZWB BLE module, (B) SWD 10-pin debug, (C) Fujitsu MB85RS4MT** 512 kB **FRAM, (D) Ambiq AM1805 external ultra-low power RTC, (E) Seiko Instruments CPH322A** 11 mF **super-capacitor** [29]**, (F) Nowi NH2D0245 energy harvesting PMIC, (G) solar panel connector, (H) Micro USB power supply. (bottom) Schematic hardware overview of core elements of the considered hardware.**

5.1.2 Microcontroller

The nRF52840 has an 32–bit ARM Cortex-M4 64 MHz processor chosen for the wide variety of features and power efficiency. The nRF52840 will be further referred to as MCU. The MCU contains all the necessary peripherals such as timers and radio to send and receive BLE packets. External FRAM is added to the system and is connected via the Quad Serial Peripheral Interface (QSPI) to the MCU and is used to store and restore the checkpoints fast and reliable from the MCU. Furthermore the MCU is connected to the external RTC via Serial Peripheral Interface (SPI).

5.1.3 External Real-Time Clock

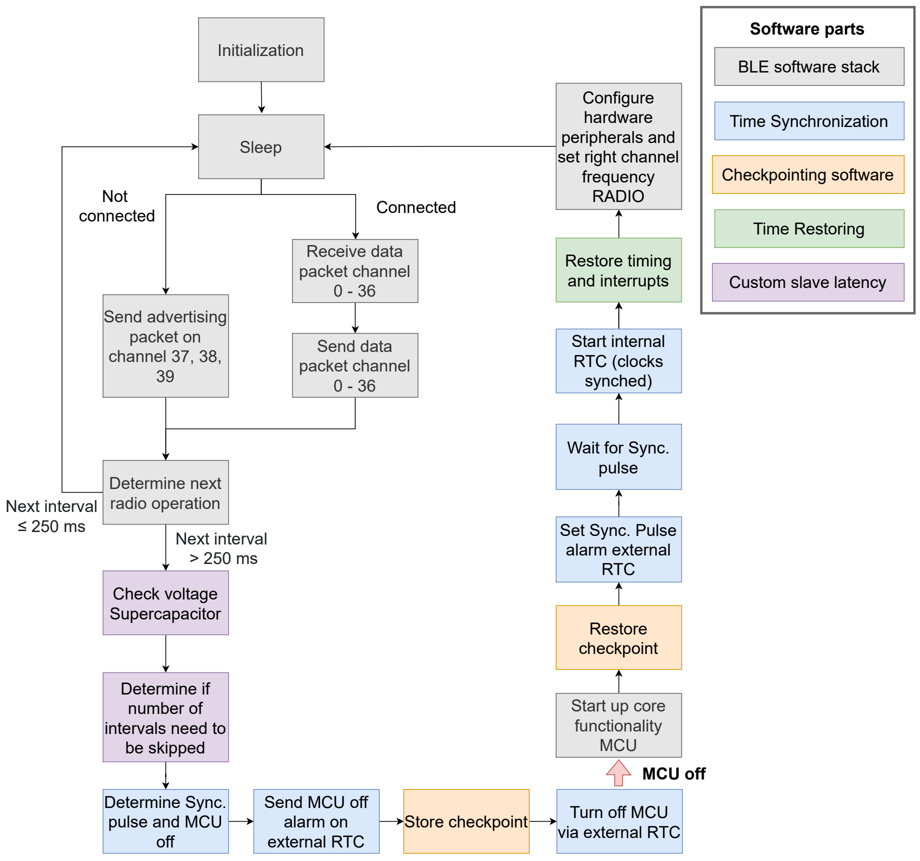
The Ambiq AM1805 external ultra-low power RTC [1] is used for switching the power of the MCU on/off, time synchronization and timekeeping. The VCC of both the MCU and FRAM are connected to an power switching output (PSW) of the external RTC. The MCU can instruct the external RTC to turn the MCU off (which will also turn off the FRAM) and to turn the MCU on at a given time value via the alarm feature. The alarm feature is configured such that it will switch the power switching output when the alarm value is reached. This means the MCU can determine how long it will be turned off. When the time value has passed, the external RTC will turn the MCU on again with the power switching output via the alarm feature. Also, an output of the external RTC is connected to the General Purpose Input/Output (GPIO) input of the MCU which will be switched on whenever the synchronization of the two clocks need to occur. This signal is called the synchronization pulse. More about the functionality of time synchronization will be described in Section 5.2.3.

5.2 Software Design

The software architecture for intermittently-powered networked systems is illustrated in Figure 5.2. The BLE advertising interval is fixed at 2 s and the connection interval at 1 s. These values are chosen because they are large enough to not deplete the super-capacitors rapidly. When the next connection interval or advertisement is bigger than 250 ms the proposed software architecture is executed. When this software is executed first the capacitor voltage will be checked in order to determine if there is enough power in the system or to skip a number of intervals in order to harvest more power. Then the MCU determines the next clock synchronization moment and for how long it has to be turned off. This period is send to the external RTC and used in its alarm feature to turn the MCU on again. Via the external RTC the VCC of the MCU is then turned off. With the external RTC alarm feature the MCU will turn on after this period is over. Then the MCU will boot and run its startup code. Afterwards the checkpoint is restored. The synchronization pulse is set via the alarm feature on the external RTC and when this signal is received the internal RTC[1] will start. Furthermore, the timing and the configuration of the peripherals are restored such that the radio is ready to be used. The MCU will then sleep until the connection interval or advertisement occurs. After that the next connection interval or advertisement is determined again. The proposed software system also includes intermittent advertisements. However, the focus of this thesis and the implementation will be on the intermittent connection, so no evaluation of advertisements are presented here.

In the software architecture presented on Figure 5.2 we can see the BLE software stack, which includes a light-weight Operating System (OS), which is described in Section 5.2.1. The contributions in this thesis are split per different software sections which are added to the OS: the Checkpointing software,

[1]On the nRF52840 the timer peripheral is called the Real-Time Counter instead of the abbreviation: Real-Time Clock (RTC) used in this thesis. However, to keep it simple we use the same abbreviation for both clocks, while we will precede it with "internal" and "external" to note the difference.



Initialization
Sleep
Not connected
Connected
Send advertising packet on channel 37, 38, 39
Receive data packet channel 0 - 36
Send data packet channel 0 - 36
Determine next radio operation
Next interval ≤ 250 ms
Next interval > 250 ms
Check voltage Supercapacitor
Determine if number of intervals need to be skipped
Determine Sync. pulse and MCU off
Send MCU off alarm on external RTC
Store checkpoint
Turn off MCU via external RTC
MCU off
Start up core functionality MCU
Restore checkpoint
Set Sync. Pulse alarm external RTC
Wait for Sync. pulse
Start internal RTC (clocks synched)
Restore timing and interrupts
Configure hardware peripherals and set right channel frequency RADIO
Software parts
BLE software stack
Time Synchronization
Checkpointing software
Time Restoring
Custom slave latency

Figure 5.2: **Software architecture of the intermittent BLE software. Here the different colors indicate different software parts.**

described in Section 5.2.2, Timekeeping, described in Section 5.2.3 and the Connection Interval Adaptation Algorithm, described in Section 5.2.4.

5.2.1 Bluetooth Low Energy Software Stack

The BLE software stack that is used for our implementation is the Packetcraft Protocol Software [47]. This BLE software stack is a newer version of the more elaborate documented ARM Mbed Cordio software stack [38]. The newer version still has all the same core functionalities, but contains BLE version 5.2 instead of 5.1. Packetcraft Protocol Software BLE software stack is chosen mainly for its low-complexity and full accessible code of all the BLE stack layers. Other considerations were: Zephyr [69], Apache NimBLE [2] and Nordic SDK [45]. All software stacks worked on the nRF52840 chip. However, Zephyr and Apache NimBLE implemented both a Real-Time Operating System (RTOS) with a lot of additional features. To be able to store and restore a RTOS would dramatically increase the complexity of the software. The Nordic SDK is less complex, but is closed source which does not give access to the link layer of the BLE stack.

The light-weight OS in Packetcraft Protocol Software is called Wireless Software Foundation (WSF) and it can be seen as a loop that checks for OS events, handles timing services and finally sleeps until an interrupt occurs [38]. The radio operations are interrupt driven and are served with the highest priority. The software can be configured with two different clocks: (i) high frequency 1 MHz clock that uses the TIMER peripheral or (ii) the RTC peripheral that uses a 32.768 kHz clock. The latter is the most power efficient and is used for this thesis.

The original Packetcraft Protocol BLE software stack, downloaded from [47], will be referred to as the default BLE software, while the developed intermittent software made for this thesis will be further referred to as the intermittent BLE software. Note that in the BLE software stack there are different BLE applications. These applications are slave device applications with different kind of BLE profiles. For example a fit application that can connect to a master device which can measure heart rate values or a uribeacon application that advertises URL links. The developed intermittent software is a modified fit application with custom software that removes the fit BLE services. For this work only a custom and empty service is created that will not be used. Furthermore for all applications (intermittent or default) of the software stack two improvements are implemented (i) enabling the DC/DC regulator in the MCU to reduce the power consumption of the radio and (ii) implement CMAKE (written by Jasper de Winkel, direct supervisor of this MSc thesis) which makes adding multiple libraries and building the code easier.

5.2.2 Checkpointing Software

The implemented checkpointing software is MPatch [12]. This software was first presented and used in [11] and then Jasper de Winkel made it compatible for the nRF52840 MCU for [13]. The nRF52840 compatible MPatch software made by Jasper de Winkel is used for this thesis. With MPatch certain consecutive memory regions can be marked as patches. In our design we use these patches to determine before compilation manually what data in the .bss (statically allocated variables) and .data (global and static local values) memory regions need to be checkpointed before the MCU turns off. The less data that needs to be checkpointed the faster the storing and restoring is. Due to the complexity of the BLE stack there have not been much work done on defining what memory regions are critical to checkpoint. This means that more data is checkpointed than is required in some instances, which results in longer checkpoints and more power consumption. The main contribution of this thesis is about placing the checkpoint functions on the correct location in the BLE software stack code.

5.2.3 Timekeeping

The external RTC is used to keep track of the time between power outages of the MCU and to synchronize with the internal RTC of the MCU. When the clocks are synchronized and the exact amount of clock ticks are determined the timing from the MCU can be restored. Synchronization of the clocks is required in order to accurately know how many clock ticks have occurred in the off time of the MCU. This accuracy is required to know when the radio should be active and receive during the connection. In addition having clock tick accuracy will keep the radio receive window of the slave device small, which will reduce the power consumption. The written synchronization software of the clocks, used in this thesis, is developed in collaboration with Jasper de Winkel and is also used in [13]. The time synchronization between the internal and external RTC and the restoration of timing will be described separately below.

Time Synchronisation

The MCU and the external RTC, which will be synchronized with the internal RTC after the MCU is booted up. Both the external RTC and the internal RTC in the MCU are using 32.768 kHz timers. To have both clocks synchronized for one clock tick on the external RTC one clock tick must also happen on the MCU. This can be achieved by starting the internal RTC on the MCU exactly when one clock tick on the external RTC occurs via an interrupt generated by the external RTC. Unfortunately on the external RTC chip Ambiq AM1805 [1] there are no registers to read the exact amount of clock ticks of the 32.768 kHz clock. Also there is no way to switch an output (or to generate) some interrupt on exactly one clock tick of the 32.768 kHz clock. The only way to read out or set the time of the clock is in years, months, date, hours, minutes, seconds and hundreds of a second. This means that hundreds (steps of 10 ms) is the most accurate time value that can be read or used for features such as an alarm, that can switch an output from the external RTC when a defined time value is reached (in steps of 10 ms). There are however time values which can be used to generate an interrupt that have exact amount of ticks of the 32.768 kHz clock and which is also divisible by 10 ms. For instance one is second has exactly 32768 ticks, which is 100 times 10 ms. When we divide this by 4 we got 250 ms with exactly 8192 ticks or 25 times 10 ms. This is the lowest amount of time that is divisible by 10 ms and has an exact amount of clock ticks from a 32.768 kHz clock. This means that in intervals of 250 ms the clock can be synchronized with the MCU. Since the external RTC starts with 0 ticks we know that on every 250 ms interval exactly 8192 clock ticks have occurred. Note that on every multiple of 250 ms on the external RTC a synchronization moment is possible for the MCU. This is why in order to have intermittent operation the next connection interval or advertisement needs to be bigger than 250 ms.

The MCU will use the alarm feature from the external RTC for two things: (i) to turn on the power from the MCU (ii) to send a signal (called the synchronization pulse) to the GPIO of the MCU that will generate an interrupt to start the internal 32.768 kHz RTC (which will synchronize the clocks). The alarm time for when the MCU should be turned on is determined by when the synchronization can happen. Thus the MCU must be turned on before the synchronization pulse. The synchronization pulse is the last synchronization moment before the connection interval (which is the last multiple of 250 ms before the connection interval). The time when the MCU is turned on and when the synchronization happens is fixed at 20 ms and is used for booting the MCU and restoring the code from the checkpoint. In Figure 5.3 the procedure of the communication between the external RTC and MCU for time synchronization is shown.

This solution however leads up to additional overhead between the synchronization pulse and the connection interval. If, for example, the connection interval is at 900 ms from the perspective of the external RTC than the last possible synchronization moment is at 750 ms (the last multiple of 250 ms before the connection interval). This leaves 150 ms of time between the synchronization

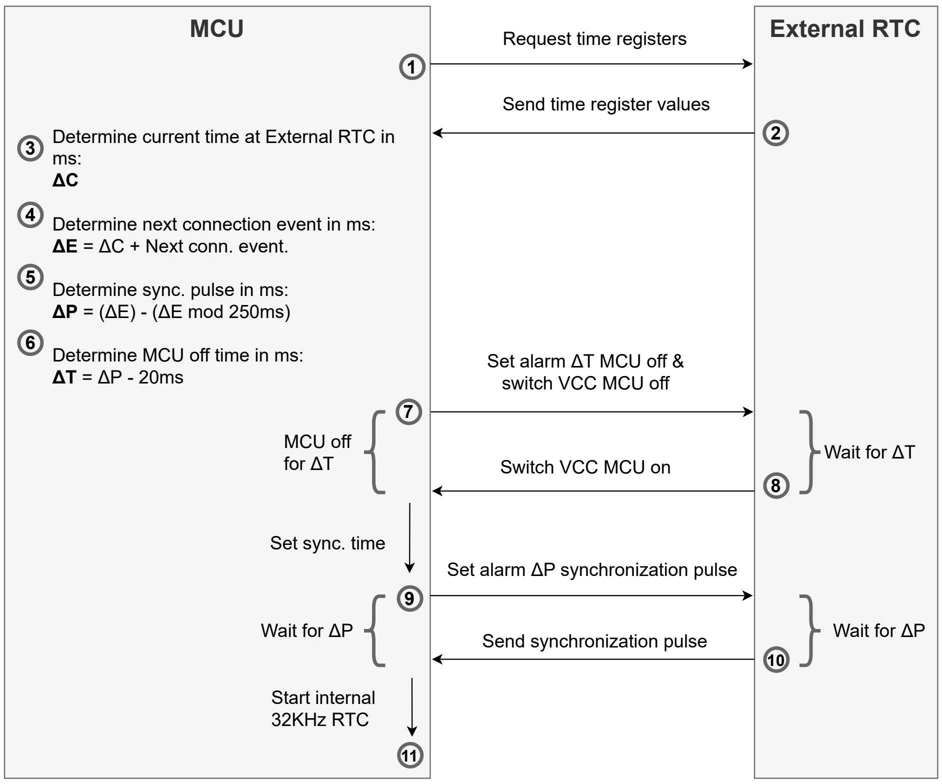

MCU
External RTC
Request time registers
Send time register values
Determine current time at External RTC in ms:
ΔC
Determine next connection event in ms:
ΔE = ΔC + Next conn. event.
Determine sync. pulse in ms:
ΔP = (ΔE) - (ΔE mod 250ms)
Determine MCU off time in ms:
ΔT = ΔP - 20ms
Set alarm ΔT MCU off & switch VCC MCU off
MCU off for ΔT
Switch VCC MCU on
Wait for ΔT
Set sync. time
Set alarm ΔP synchronization pulse
Wait for ΔP
Send synchronization pulse
Wait for ΔP
Start internal 32KHz RTC

Figure 5.3: **Procedure of time synchronization between external RTC and MCU. Here the ΔC is the current time of the external RTC, ΔE is the next connection interval time projected on the external RTC time, ΔP the synchronization moment which is the last** 250 ms **multiple before the connection interval and ΔT the wake up time for the MCU which is** 20 ms **before the synchronization moment such that the MCU can boot up and restore the checkpoint.**

pulse and the connection interval. During this overhead the MCU has to sleep most of the time to be as energy-efficient as possible. This overhead however can be reduced by moving the anchor point of the connection interval closer to the synchronization pulse or vice versa. The anchor point is the starting time of the connection interval and can be reduced via two methods: (i) changing the clock speed of the external RTC and (ii) changing the connection interval temporarily on the MCU. By changing the clock speed of the external RTC to be slightly faster than the internal RTC of the MCU the synchronization pulse will move slightly closer to the anchor point of the connection interval. Then also the receive window of the MCU radio has to be increased due to the more inaccurate clock. This method works but leaves the radio on for longer at every connection interval and can take up to 10 minutes in order to reduce the gap significantly, thus will not be further discussed here. However, changing the connection interval temporarily can reduce the overhead much faster. The MCU is programmed such that it can detect this overhead and if this overhead is above 150 ms it will change the connection interval from 1 s to 990 ms. This change in the connection parameters is done from the slave device, which is typically done from the master device. When the time between the synchronization pulse and connection interval is between 105 ms and 125 ms the connection interval is updated again. This is because it takes usually about 7 to 8 BLE connection intervals before the connection interval is actually changed. This way the time between the synchronization pulse and the connection interval will be around $25 - 55$ ms most of the time. However, it is required that the master device cannot does not change the anchor point of the connection interval when the connection interval is changed. When this method was tested with the nRF Connect app for Android [39] as a master device it became clear that the Android OS schedules the anchor point of the connection interval at a random time. This way we cannot predict where the anchor point will be when we change the connection interval. With the nRF Connect app for PC [40] as a master device this is predictable and is therefore used as the master application. Changing the connection interval is usually done by the master device but the slave device can request a connection parameter update via the L2CAP_CONNECTION_PARAMETER_UPDATE_REQ [20, volume 3, page 1069] command. This packet is send with the update request command along with the new connection parameters: the maximum connection interval, minimum connection interval, slave latency and supervision timeout. However, the master device has to accept these new connection parameters, which will be the case with our master device. The master will then send a LL_CONNECTION_UPDATE_IND [20, volume 6, page 3035] with the updated values and the connection parameters are changed at a specific connection interval indicated by the master. With the new feature of changing the connection interval slightly in order to reduce the time between the synchronization pulse and the connection interval, there is a bug that sometimes when the connection interval is changed at the start of the connection, the connection interval is not updated correctly and the connection fails. This is a bug that has not been resolved yet and due to the low likelihood of this problem it is not further investigated.

Time Restoration

Time for state restoration is required in order to accurately determine the next connection interval after the MCU is turned on. The BLE software will schedule the next connection interval in the RTC compare register[2] after the radio operations are done. When the MCU is turned off all the peripheral registers are cleared. Also the internal RTC counter value always starts at zero and cannot be set to another value.

In order to set the right value of the next connection interval after the MCU is turned on the external RTC is used as the main clock of the system. This can be done by using the synchronization pulse time and the internal RTC. We know that when the synchronization pulse occurs the external RTC clock has an exact amount of RTC ticks and the internal RTC clock will start when the synchronization pulse occurs. When the synchronization pulse time is converted to RTC ticks and then added with the internal RTC ticks the total amount of RTC ticks can be determined. Note that the external RTC has time value registers that need to be converted to 32.768 kHz RTC ticks. This way, even

[2]The compare register is a register where a RTC clock tick value can be set which is used to generate an interrupt or start an event in the software.

during power failures when the internal RTC ticks are reset to zero, the MCU can still determine the total elapsed time. Since the RTC counter value will always start at zero and cannot be set to another value a compensation value is used to convert the time on the MCU to the time on the external RTC. The compensation value is the last synchronization pulse time, where the external RTC time is converted to internal RTC ticks. Throughout the whole BLE software stack there is one set of function calls made to the internal RTC to read out the counter value, to set the compare register value or to read out the compare register value. By embedding the compensation value in this set of function calls to the internal RTC functions, from the BLE software stack perspective, the clocks have not been reset. If, for example, the counter value of the internal RTC then is required by the BLE software stack, the current internal RTC ticks are then added to the compensation value. This way when the MCU has been rebooted the time value that is then returned with the compensation value will resemble the actual elapsed time value instead of the internal RTC tick value that starts at zero after the reboot. Note that the internal RTC is a 24 bit counter and thus wraps around the timer.

In this implementation and the results constructed for this thesis there is an error made in setting the compensation value, where the internal RTC counter value is added to the synchronization pulse time to construct the compensation value. This was done in order to have the exact time stamp in RTC ticks in the function that sets the compensation value. However, if the current RTC time value is read by the BLE software stack with the compensation value, the RTC ticks that were added for the compensation value are then added twice (one time for constructing the compensation value and then added with the compensation value itself). Since these RTC ticks are one to at most two RTC ticks this has not been detected during testing and to the best of our knowledge did not interfere with the stability of the system.

The external RTC has an accurate 32.768 kHz crystal and the error value is measured at around 10 ppm. When we configure the MCU with the default 20 ppm clock accuracy that is also used for the internal RTC, the radio receive window will miss the data packet of the master device usually within a minute. However, when the MCU is configured at 50 ppm the receive window will not miss the data packet from the master device for more than an hour. This extra inaccuracy is probably the cause of the conversions that are done from the external RTC time to the internal RTC time and vice versa. Also, the interrupt of the synchronization pulse that starts the internal RTC does have a slight delay which could cause that slight inaccuracy. Finally note that for unknown reasons when two devices are connected with the intermittent BLE software the connection times out at around 4000 s. After setting the radio receive window to the maximum value we excluded the possibility that a clock drift will timeout the connection. Currently the cause of this problem is not know since it is time consuming to fix this problem. Therefore this bug is not solved for this thesis.

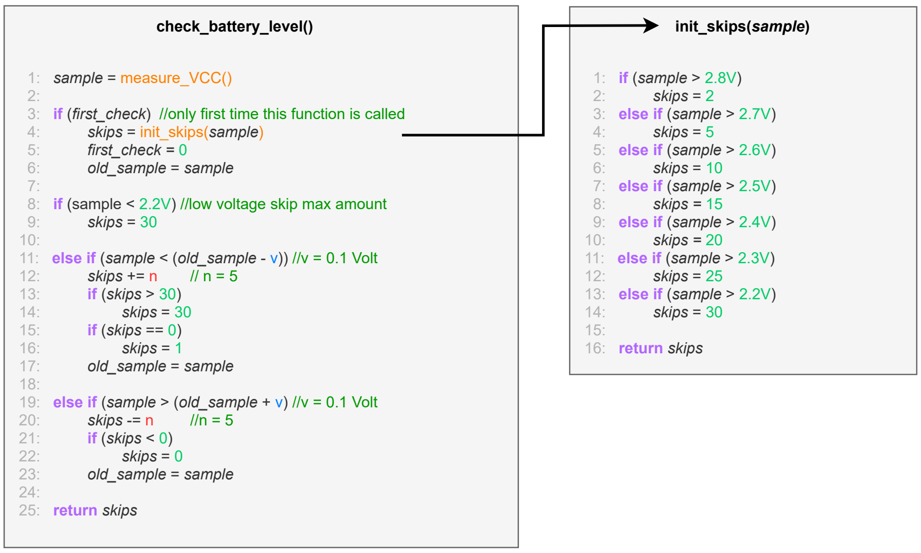
5.2.4 Connection Interval Adaptation Algorithm

The connection interval adaptation algorithm is implemented such that the data rate adapts to the VCC voltage level of the MCU. The assumption is that an intermittent operating BLE device should be aware of the energy budget that it has. By monitoring the VCC voltage of the MCU (which is connected to the super-capacitors) the MCU is aware of its power budget. This way the MCU can determine what data rate it want to operate depending on how much voltage is measured at the super-capacitors. The VCC is chosen, because the super-capacitor is connected to the VCC of the MCU. The MCU can measure the voltage with the Successive Approximation Analog-to-Digital Converter (SAADC) peripheral. The software for the connection interval skips is a modified version of the slave latency code that was present in the BLE software stack.

In Figure 5.4 the connection interval adaptation algorithm is presented. First the voltage of the super-capacitors is read (*sample*). The first time the algorithm is executed the starting point of the connection interval skips is determined (init_skips(*sample*)). The starting points in the init_skips function were determined by trial and error. Whenever the super-capacitors were fully charged via the USB connection, the super-capacitors were charged at 3 V. Since the voltage measurement of the super-capacitor is done after the connection interval for the intermittent BLE software, the measured voltage is typically around 2.9 V. At the start there are always at least two connection interval skips, this is because zero connection interval skips has a high power consumption and not to much power is lost at the start of the measurements. Furthermore, the MCU will not continue operation when the voltage is measured at around 1.7 V. That is why at 2.2 V the maximum number of connection interval skips are chosen to have a buffer for the system to harvest more energy. The other starting point values between 2.2 V and 2.8 V were spread equally in steps of 0.1 V (except at 2.8 V).

After the initialization the algorithm will always check the *sample* value to determine if the skips need to change by checking if the voltage has dropped or increased by 0.1 V (indicated in blue as v in Figure 5.4). This is done by checking the *old_sample* value, which is the value that is saved whenever the skips are changed. The *old_sample* value is checked, because the voltage over the capacitors increase or decrease gradually. By comparing it with this old value from when the connection interval skips were changed, we only change the skips whenever a bigger change occurs (in our implementation at least 0.1 V). This 0.1 V is determined by trial and error and was chosen due to the good balance between not changing the connection interval to often due to the fluctuation between measurements and still have an accurate enough measuring range to detect significant changes. After testing with values bigger than 0.1 V, the adaptation of the connection interval skips was slow. Especially in a low power setting where not much energy was harvested a lot power power was wasted since it had to drop a significant voltage level. Otherwise, when testing with smaller values than 0.1 V the connection interval skips would change to often and would sometimes keep changing between two values due to the different fluctuations in the measured super-capacitor voltage.

When the voltage of the super-capacitor has increased with 0.1 V, there will be subtracted five (indicated in red as n in Figure 5.4) from the total amount of connection interval skips with a minimum of zero connection interval skips. Also, when the voltage of the super-capacitor has decreased with 0.1 V, then there will be added five (indicated in red as n in Figure 5.4) to the total amount of connection interval skips with a maximum of 30 connection interval skips. The number five was chosen since there are six steps of 0.1 V between 2.2 V and 2.8 V and a total of 30 connection interval skips. Note at line 15 of Figure



```
sample = measure_VCC()

if (first_check)  //only first time this function is called
    skips = init_skips(sample)
    first_check = 0
    old_sample = sample

if (sample < 2.2V) //low voltage skip max amount
    skips = 30

else if (sample < (old_sample - v)) //v = 0.1 Volt
    skips += n       // n = 5
    if (skips > 30)
        skips = 30
    if (skips == 0)
        skips = 1
    old_sample = sample

else if (sample > (old_sample + v)) //v = 0.1 Volt
    skips -= n       //n = 5
    if (skips < 0)
        skips = 0
    old_sample = sample

return skips
```

```
if (sample > 2.8V)
    skips = 2
else if (sample > 2.7V)
    skips = 5
else if (sample > 2.6V)
    skips = 10
else if (sample > 2.5V)
    skips = 15
else if (sample > 2.4V)
    skips = 20
else if (sample > 2.3V)
    skips = 25
else if (sample > 2.2V)
    skips = 30

return skips
```

Figure 5.4: **Pseudo code for the connection interval skip adaptation algorithm. Here the number of skips (red) "n = 5" and the voltage of when the connection interval skips will change depending on the previous voltage level of when the connection interval skips were changed is at (blue) "v = 0.1 V". The starting point of connection interval skips in the function "init_skips()" is constructed depending on the values "n" and "v".**

5.4, if the connection interval skips are zero and is required to increase, the connection interval skips is only increased by one. This is done, because when there is a lot of energy in the system the measured voltage fluctuates heavily around 0.1 V, which is most likely due to the high data rate that has a high power consumption from the MCU. Adding this value of one does not make the connection fluctuate heavily between zero and five connection interval skips, but between zero and one connection interval skips which improves the data rate significantly.

The connection interval skips for the connection interval adaptation algorithm is different from the BLE slave latency, because it can change the number of connection interval skips depending on the voltage level while the BLE slave latency is a connection parameter that has to be changed with acceptance from the master device. In addition, the connection interval skips for the connection interval adaptation algorithm can skip up to 30 connection interval skips. This is more than the BLE slave latency, which ranges from zero to (connSupervisionTimeout / (connIntervalMax × 2)) - 1) (for the BLE 5.2 specification [20, volume 3, page 1071]). This is possible since the maximum super vision timeout for each connection can be configured to 32 s. If then the connection interval is set to 1 s, the BLE protocol will allow for 31 connection interval skips of 1 s to be skipped as long as the last second a data packet is exchanged between the master and slave. However, operating on these boundaries will reduce the reliability of the connection. This is why this configuration is typically not recommended. In order to improve the reliability of the connection a feature is added to the code that when the connection interval skips is bigger than 15 and a packet from the master device is not detected at the connection interval, the slave device will not skip any connection intervals until a packet from the master device is received. Note that the master device is not aware of the connection intervals skips and thus will send at every connection interval a packet to the slave device. The maximum connection interval skips is capped at 30, such that when at 30 connection interval skips a packet is missed the slave device has as least 2 more data packets that it can receive from the master device. This way some additional reliability in the connection is added. The advantages of the connection adaptation algorithm is that there is no connection update via the master device required in order to skip a different connection interval and more connection intervals can be skipped compared to what the BLE 5.2 specification slave latency allows. This way more power can be saved when the MCU is turned off. The disadvantages of the connection adaptation algorithm is that data packets are delayed, the data rate is low and a less reliable connection when data rate is very low (more than 15 connection interval skips).

There is however a conflict when during connection skips also the connection parameters need to be updated. If the master device decides to update the connection between skips the slave device must consider this change in order to have the correct timing of the connection intervals. This is done by turning on the MCU at the connection interval the connection parameter is changed. This way there do not have to be taken into account two connection intervals during the skips, which will reduce the complexity. After that the connection interval adaptation algorithm will determine the skips as explained before. Also a bug is detected from the default BLE software, where the first four connection intervals cannot be skipped when the skips are greater than 14. This is fixed by simply never skip the first four connection intervals.

Chapter 6

Evaluation Results

This chapter presents the results of maintaining a BLE connection, timekeeping during power failures and synchronizing time on a solar energy harvesting device. Section 6.1 describes the setup for the experiments. Section 6.2 presents the power consumption of the proposed software stack (further referred to as intermittent BLE software) and the power consummption during a connection will be compared with the non-intermittent default Packetcraft Protocol Software (further referred as default BLE software). Section 6.3 shows a theoretical model that is constructed from power measurements, and will present the connection interval skip value at which the intermittent BLE software is more power efficient than the default BLE software. Note that for the default BLE software the connection interval skips are configured via the BLE slave latency (described in 3.3.2), while for the intermittent BLE software the connection interval skips are done by actually skipping connection intervals (as described in 5.2.4). Furthermore, this section presents an experiment that will test the previous mentioned model, by comparing the connection time for different fixed connection interval skips for both the intermittent and default BLE software in a low light condition. Finally, Section 6.4 presents the experiments of the connection interval adaptation algorithm under various light conditions and will show how the connection interval skips will adjust to the light intensity.

6.1 Experiment Setup

In Figure 6.1 the experiment is shown. In this setup a solar panel is placed in a cardboard box with a controllable Philips Hue white ambient light source [48]. This way the solar panel is shielded from other light sources. The Saleae Logic pro 8 logic analyzer [51] is connected to the intermittent device and laptop to trace the status of the radio, the voltage over the super-capacitor (VBAT), the VCC of the MCU and the voltage of the solar panels (VSOL). The trace of the Saleae logic pro 8 is debugged on the laptop with the Logic 2 software [52] version 2.3.37. There are two nRF52 Development Kits added to the setup [42], one with nRF sniffer [41] version 3.1.0 software that will live capture Bluetooth packets with Wireshark version 3.4.8, and the second development kit acts as a programming tool and debugger for the intermittent device. Also the nRF52 Dongle [44] is connected to the laptop and is the master BLE device. This

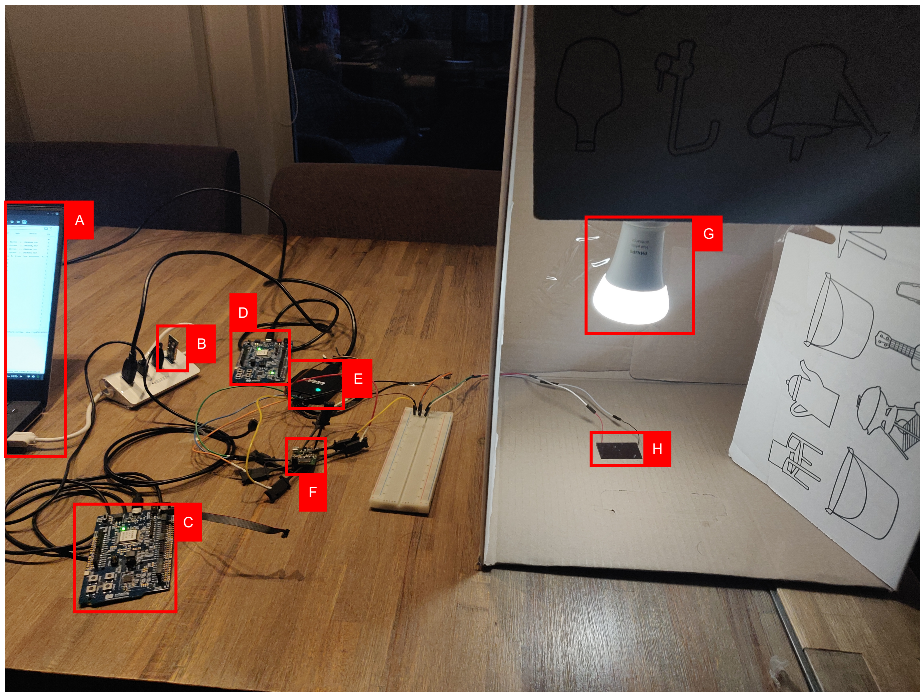


Figure 6.1: **Setup intermittent BLE measurements: (A) the laptop that is connected to the nRF52 Development Kits, nRF52 Dongle and the Saleae logic pro 8. Also it runs the Wireshark analyzing tool, nRF Connect App for PC and the Saleae Logic 2 software, (B) nRF52 Dongle, (C) nRF52 Development Kit (debugging and programming), (D) nRF52 Development Kit (capturing packets with nRF Sniffer), (E) Saleae Logic pro 8, (F) the intermittent BLE device, (G) Philips Hue ambient white lamp, (H) AM-1513CA solar panel.**

master device is configured and programmed via the nRF Connect App for PC [40] version 3.0.0 on the laptop.

For our experiments a custom application is used for both the intermittent software and the default BLE software. The custom application is a stripped down version of the heart rate application from the default BLE software and has one custom service that is not used for these measurements. This way only empty packets will be exchanged after the service discovery has been done. Furthermore all the software files are build with debug and UART mode off to further reduce the power consumption.

6.2 Power Consumption Measurements

The goal of this experiment is to compare the power consumption of the connection interval of the default BLE software with the intermittent BLE software to determine the overhead and when the intermittent software is more power efficient than the default software. In order to measure the power consumption the X-NUCLEO-LPM01A (STM32 power shield) [57] is used as a power

Default BLE Software			
Label	**Function**	**Time (ms)**	**Energy (µJ)**
A	Pre-processing	0.56	1.679
B	Radio: RX & TX	0.74	7.486
C	Post-processing	0.23	0.899
	Total	**1.53**	**10.064**
Intermittent BLE Software			
Label	**Function**	**Time (ms)**	**Energy (µJ)**
D	Booting	1.25	42.610
E	Startup code	1.85	25.606
F	Restoring checkpoint	7.75	91.520
G	Sleep: wait for sync. pulse	16.25	40.204
H	Restore timing & pre-processing	0.50	2.991
I	Sleep: wait for connection interval	35.00	81.738
J	Radio: RX & TX	0.81	7.930
K	Power saving algorithm	0.46	2.919
L	Checkpoint	8.00	86.071
M	Post-processing, turn off MCU	0.76	3.639
	Total	**72.63**	**385.228**

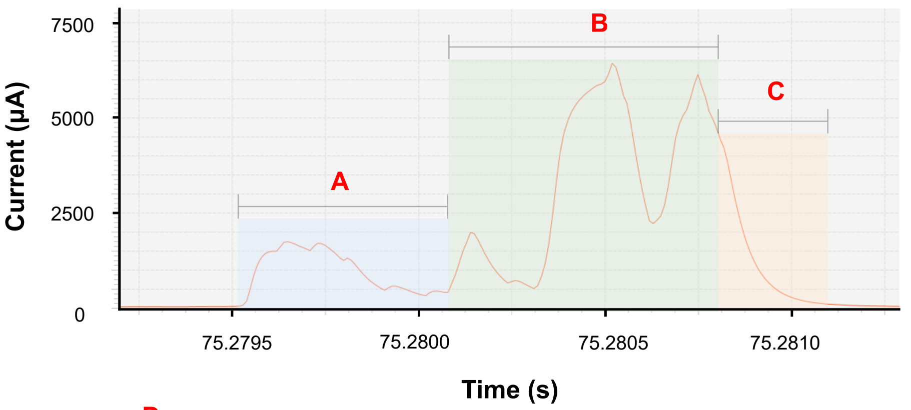
Table 6.1: **Power consumption and time of the labels showed in Figure 6.2 of the default BLE software and the intermittent BLE software.**

source instead of the solar panel and is configured to power the system at 3 V. With the X-NUCLEO-LPM01A the current in Ampére and power in µJ can be measured and monitored with the STM32 Cube Monitor [56] software tool. The advantage is that with the X-NUCLEO-LPM01A we can accurately measure the MCU-off current between connection intervals of the intermittent BLE software. However, the STM32 Cube Monitor tool has a very poor and inaccurate way of measuring small time values of µs, resulting in less accurate time measurements.

In this experiment the power consumption of fixed connection interval skip zero and 14 are measured for ten minutes. For the default BLE software the slave latency is configured on the master with nRF Connect App for PC software. For the intermittent BLE software the connection interval skips are programmed in the code. The power consumption of the minimum and maximum connection interval skips were measured such that with linear scaling all the connection interval skip values inbetween can be calculated.

In Figure 6.2 the power consumption of a connection interval is shown for both the default and intermittent BLE software at connection interval skip 0. In Table 6.1 the energy values are given for all events labeled in the figure. The results show that our implementation of the active part of the intermittent connection interval takes roughly 47 times longer in time and costs approximately 38 times more energy at zero connection interval skips. A note of caution is due here since this is not the power consumption of the whole connection interval that includes the sleep or off time.

The main advantage of the intermittent BLE software is that it turns off the MCU between connection intervals. This means that the longer the MCU is turned off the more power efficient the intermittent solution becomes. In Table



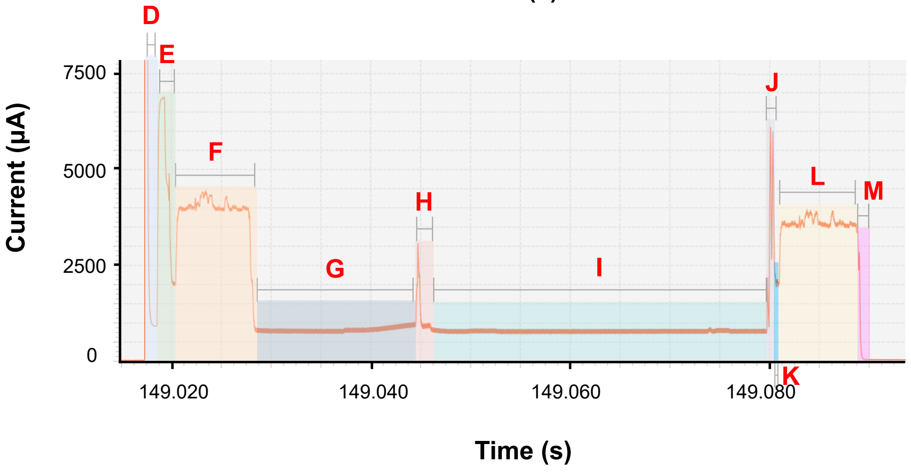


Figure 6.2: **Power consumption of an empty connection interval packet without the sleep/off time: (Top) default BLE software and (Bottom) Intermittent BLE software. Both measurements were configured with a connection interval of one second and zero connection interval skips. The time and energy of the labeled parts are defined in Table 6.1.**

6.2 the total power consumption is shown, that includes the sleep or off time, of connection interval skips zero and 14. It is interesting to observe that with 14 connection interval skips the intermittent BLE software is **almost two times more power efficient**. This is primarily due to difference between the default BLE software's sleep current at roughly 25 µA, and the MCU-off current of the intermittent BLE software at roughly 3 µA. Since the longer the MCU is not active, the more impact this difference will have and this will result in that the intermittent BLE software eventually will be more power efficient. However, at zero connection interval skips the default BLE software is almost five times more power efficient than the intermittent BLE software.

Knowing the power consumption at connection interval skips zero and 14, we can estimate the power consumption of the connection interval skip values inbetween. A model is constructed that shows the power consumption of the connection interval skip values ranging from zero to 14. From this model we can determine what number of connection interval skips the intermittent BLE

	Default BLE Software		**Intermittent BLE Software**	
Connection Interval Skips	**Time (s)**	**Energy (µJ)**	**Time (s)**	**Energy (µJ)**
0	1.000	86.022	1.001	417.108
14	15.019	1110.391	15.017	601.176

Table 6.2: **Comparison between the default and intermittent BLE software of the power consumption and time at zero and 14 connection interval skips of the whole connection interval. This includes the sleep time for the default BLE software and the off time for the intermittent BLE software.**

software is more power efficient than the default BLE software. A model is chosen over measuring every individual connection interval skip, mainly because each the measurement took a considerable amount of time (around 30 minutes) and this was not feasible for 28 measurements.

The connection interval skip values in-between zero and 14 in the model are determined by applying linear scaling between the measurement from zero connection interval skips and 14 connection interval skips for both the default and intermittent BLE software. In order to apply the linear scaling correctly, five random connection intervals per measurement are taken and then for each individual BLE software part the computation time and power consumption is determined. Five connection intervals for each measurement was chosen to determine the average value for each individual software part. Furthermore, the BLE software parts are the functions presented in Table 6.1, the sleep function between the connection intervals for default BLE software and the MCU turned off for the intermittent BLE software. Since not all software parts have the same power consumption per unit of time, the linear scaling is not applied to the total computation time and power consumption of one connection interval but to their individual software parts. Then the computation time and power consumption of all the software parts are added together and each connection interval skip value is determined. Note that not all software parts have been applied to linear scaling but only to a few. Most of the software parts, such as storing or restoring a checkpoint will have a fixed computation time and power consumption. But there are four software parts identified that will have a variable computation time and power consumption: (i) The radio receive window for both the default and intermittent BLE software that will increase when the number of connection interval skips increase, (ii) the time between the synchronization pulse and the connection interval for the intermittent BLE software that will vary depending on the anchor point of the connection interval, (iii) the sleep function between connection intervals from the default BLE software that will increase when more connection interval skips occur and (iv) the MCU-off time between connection intervals from the intermittent BLE software that will increase when more connection interval skips occur. The external RTC is always turned on, also for the default BLE software while the external RTC is not used. Since this is such a low powered device (100 nA) it is not deducted from the power consumption calculation due to the negligible impact. Finally to construct the total power consumption for each connection interval skip the software parts with fixed power consumption is added with corresponding variable

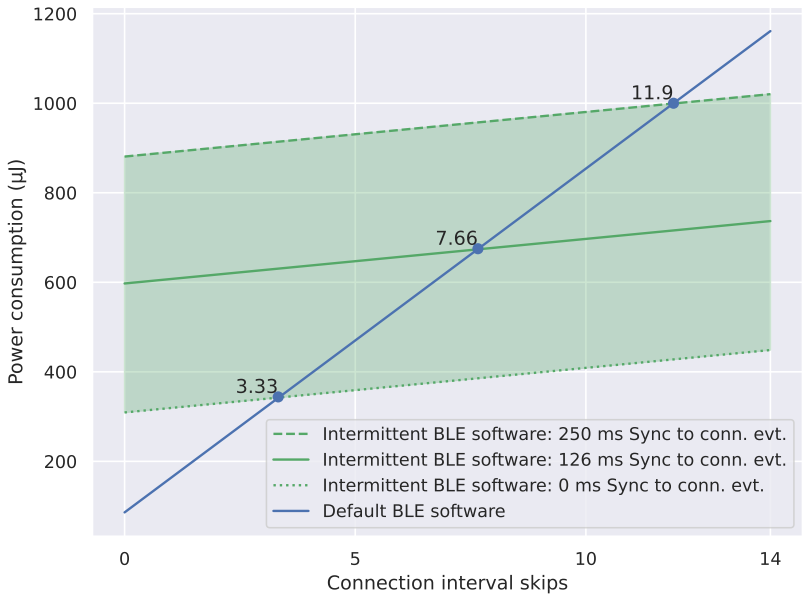


Figure 6.3: **Calculated power consumption for a connection interval of one second at different connection interval skips of the default and intermittent BLE software. For the intermittent BLE software the whole range of the overhead of the time between the synchronization pulse and the connection interval is added. The blue dots indicate the intersection between the power consumption of the default BLE software and the intermittent BLE software at the lowest, middle and highest overhead from the time between the synchronization pulse and the connection interval.**

power consumption part.

In Figure 6.3 the model of the power consumption for the default BLE software and the connection interval skips of the intermittent BLE software is shown for each connection interval skip between zero and 14. In addition the overhead of the time between synchronization pulse and connection interval is shown ranging between zero and 250 ms. This shows the variation of which the intermittent BLE software can run depending on this overhead. The figure shows that in the most ideal case where there is no time overhead between synchronization pulse and connection interval with 4 connection interval skips the intermittent BLE software is more power efficient than the default BLE software.

6.2.1 Key Insights from Power Consumption Measurements

The results of the power consumption measurements show that with 14 connection interval skips at one second connection interval the intermittent BLE software is almost two times more power efficient than the default BLE software while at zero connection interval skips the default BLE software is five times more power efficient. In addition, the overhead from the active parts of the software (not when the MCU is sleeping or turned off) for the intermittent BLE software takes roughly 47 times longer in time and costs approximately 38 times more energy at zero connection interval skips than the default BLE software. The biggest overhead is rebooting, run the startup code, storing and restoring of the checkpoints, and the idle time between the synchronization pulse and the connection interval. The overhead could be reduced by choosing an MCU that boots faster or more power efficient, optimize the startup code (if possible), operate in FRAM instead of SRAM to remove the overhead of storing and restoring to external memory, move the anchor point of the connection in order to reduce the time between the synchronization pulse and the connection interval, and finally, have an external RTC that can synchronize the clocks in smaller intervals than 250 ms in order to reduce the time between the synchronization pulse and the connection interval. With the model that is created it is determined that between four and 12 connection interval skips the intermittent BLE software is more power efficient depending on the time between the synchronization pulse and the connection interval. This is because the power saved from the intermittent BLE software by turning off the MCU is more than the added overhead of intermittent operation and thus will result in a lower power consumption than the default BLE software. It must be emphasised though that this result comes from the proposed theoretical model that only depends on the data from power measurements of the 14 connection interval skips and the zero connection interval skips for both the intermittent and default BLE software. In other words, more measurements need to be performed in order to verify its correctness.

6.3 Connection Time Measurements

The results of the power consumption model, given in previous section, shows that in theory the intermittent BLE software can be more power efficient than the default BLE software. In the next experiment presented here this model is tested in order to determine if the model is correct. This is done by measuring the total connection time up until the device runs out of energy of both the default and intermittent BLE software. The goal is to measure for connection interval skips 0, 5, 10 and 14 the total connection time for both the default and intermittent BLE software. Two measurements are done for each connection interval skips and the average is determined. During the experiment, at the start of each measurement, the voltage in the super-capacitor will be charged fully via the USB connection. Also, the power supply will remain constant and the solar panel is placed in a fixed low light setting for all measurements at 300 lx. The 300 lx value is chosen such that for most configurations the measurements will not have enough energy to last more than an hour. This is to determine the total connection time within reasonable time and because there is a bug that will terminate all connections at around 4000 s. Moreover, 300 lx is a typical indoor light level for most rooms in a house [3]. Before the connection starts the USB cable that powers the MCU is removed at the same time the master device will attempt to connect to the slave device, such that the super-capacitor will approximately have the same voltage at the start of each measurements.

In Figure 6.4 the average connection time for both the default and intermittent BLE software is shown at connection interval skips 0, 5, 10 and 14. It is interesting to observe that the connection time of the default BLE software does not change as much when the connection interval skips increased compared to the intermittent BLE software. At zero connection interval skips the intermittent BLE software has approximately one-tenth of the connection time of the default BLE software, while at 14 connection interval skips the intermittent BLE software has approximately double the connection time compared to the default BLE software. The data from these measurements can be compared with the model in Figure 6.3. Here we can see the same trend that first the default BLE software is more power efficient but when the connection interval skips increases the intermittent BLE software becomes more power efficient. From the measurements it is determined that between 5 and 10 connection interval skips the intermittent BLE software is more power efficient. This is also what can be made out of the model. These measurements confirm that the intermittent BLE software can be more power efficient when the amount of connection interval skips increases. However, one limitation of this experiment is that a small data set is used. The connection time for some measurements did vary significantly for some measurements. For example for the intermittent BLE software at 5 connection interval skips the connection times were measured at 814.478 s and 678.128 s which is 746.303 s average. The variable connection time from the measurements at the same connection interval skips can be caused by several factors, such as the voltage in the super-capacitor will never be exactly the same at the start of each connection. Also for the intermittent BLE software, the anchor point of the connection interval is different for each connection, which affects the time between the synchronization pulse and the connection interval. Finally, during the measurement of the intermittent BLE software at 14 connection interval skips one measurement failed early at around 1000 s when trying to change the connection interval. The cause of this problem is not found but most likely the master device terminated the connection, since the slave device was advertising afterwards.

6.3.1 Key Insights from Connection Time Measurements

The results of the connection time measurements show that a BLE connection is maintained for almost an hour at 300 lx and be almost two times more power efficient than the default BLE software at 14 connection interval skips and a connection interval of one second. The point where the intermittent BLE software is more power efficient is around 10 connection interval skips. This proves that the intermittent BLE software is more power efficient from a range of at least 10 to 14 connection interval skips at a connection interval of one second than the default BLE software. Also small factors can change the connection time such as, the voltage in the super-capacitor at the start of the measurement will never be exactly the same value, the solar panel can be slightly moved between measurements and temperature can influence

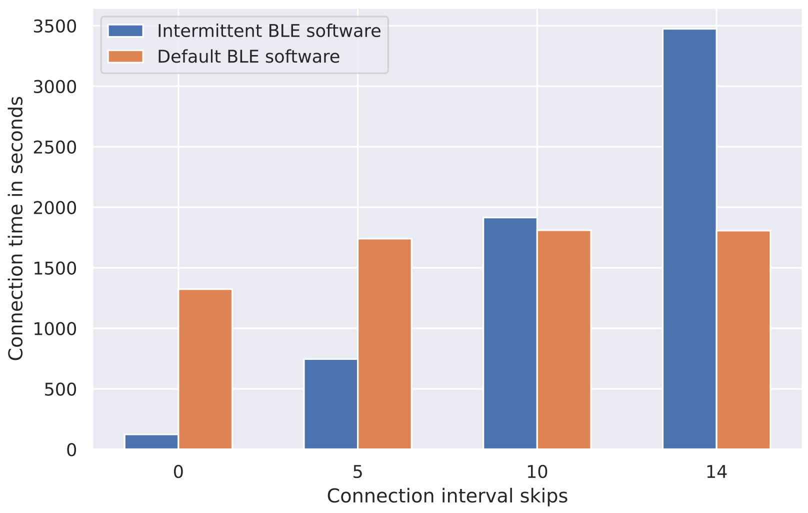


Figure 6.4: **Average connection time in seconds for default (orange) and intermittent (blue) BLE software measured at** 300 lx **for connection interval skips 0, 5, 10 and 14.**

6.4 Connection Interval Adaptation Algorithm Measurements

Finally, with two experiments the connection interval adaptation algorithm (shown in Figure 5.4) is tested, where the device is put under multiple light conditions in order to determine how the algorithm adapts the connection interval skips to the light intensity. The goal of these experiments are to determine the effectiveness of the connection interval adaptation algorithm and to determine if the connection interval adaptation algorithm can increase the connection time of the intermittent BLE software.

In the first experiment two measurements are done with the intermittent BLE software, one with the connection interval adaptation algorithm and the other without the connection interval adaptation algorithm (fixed at 14 connection interval skips). For both measurements, the solar panel is exposed to a fixed light scenario of 300 lx. The goal of the first experiment is to determine if the connection time increases with the connection interval adaptation algorithm compared to the fixed 14 connection interval skips that was presented in 6.3 and to determine the effect of the connection interval adaptation algorithm on the VCC voltage. The VCC voltage of the MCU is measured, because the connection interval adaptation algorithm will select the connection interval skips based on the VCC voltage of the MCU. The 14 connection interval skips are chosen, since this was the most power efficient intermittent BLE connection. Note that by measuring the VCC of the MCU, the voltage of the super-capacitor is measured.

In the second experiment also two measurements are done with the intermittent BLE software, one with the connection interval adaptation algorithm and the other without the connection interval adaptation algorithm (fixed at 14 connection interval skips). However here for both measurements the solar panel is exposed to a series of different light scenarios: first four minutes no light, then four minutes at 100%, then four minutes at 25%, four minutes at 50%, four minutes at 75%, four minutes at 100% and finally no light until the device will have its power depleted. The goal of the second experiment is to determine how effective the connection interval adaptation algorithm responds to increases and decreases in the voltage of the VCC. The extreme light values were chosen to see changes of the VCC faster in a smaller time window. The current setup is not very reactive to small light changes at low light strengths, such as 300 lx, due to the relative big super-capacitors and only one solar panel. The specific light scenario is chosen for two reasons: (i) to see how the connection interval adaptation algorithm reacts to sudden changes of the energy input (at the start no light and then suddenly 100%) and (ii) when the energy input gradually increases (the increasing steps from 0% to 100%).

For the setup of both experiments the solar panel is placed in a closed cardboard box and the light source is regulated in one of six different scenarios (depending on the measurement): no light (10 lx), 17% (300 lx), 25% (630 lx), 50% (2550 lx), 75% (3645 lx) and 100% (6565 lx) light intensity. Furthermore, the lamp can be set to a color scene and is configured in the "Concentrate" light scene on the Philips Hue Bluetooth app [49] version 1.34.0, since this was the brightest light scene that was available. The connection is established from the master device (nRF52 Dongle) on the nRF Connect App for PC with the connection interval of one second and zero connection interval skips. Also, the super-capacitors were fully charged during advertisement and when the connection request was send the USB power of the intermittent device was unplugged.

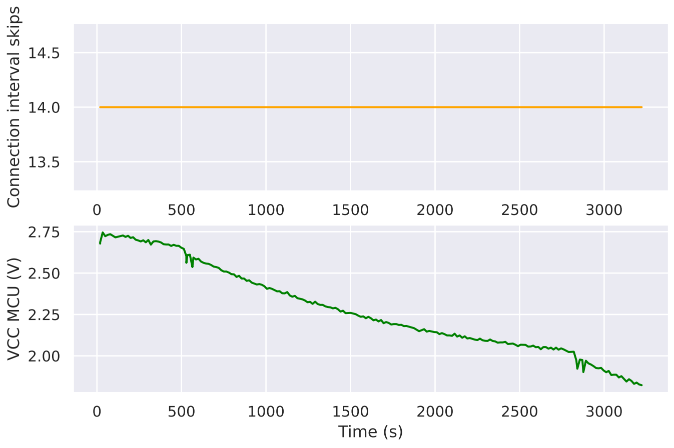
The results are constructed by mapping the light strength, connection interval skips and the VCC of the MCU to the same timeline. The connection interval skips and the measured voltage of the MCU are retrieved from the intermittent BLE software. This is possible due to the FRAM that is in the system that can store these values and can be debugged after the measurements are done. The light strength is configured manually in the Philips Hue Bluetooth app. With the logic analyser the voltage of the solar panel, the VCC of the MCU and the radio events are monitored. This way the light strength can be mapped to the connection interval skips (for the second measurement). Finally, with the Wireshark monitoring tool all packets of the connection are monitored and timestamped. With this data everything can be mapped to one timeline. Note that during the experiments the connection interval also was changed in order to reduce the time between the synchronization pulse and the connection interval. These connection interval skips are not included to the figures since they are not part of the algorithm and the state of the battery.

In Figure 6.5 the results of the first experiment is shown. The results show that the connection time with the connection interval adaptation algorithm has increased with roughly 600 s compared to the fixed 14 connection interval skips. However, the connection time with the connection adaptation algorithm most like could be increased, since the voltage of the VCC was at 2.4 V when the connection was timed out due to the bug described in Section 5.2.3. The VCC voltage decay with the connection interval adaptation algorithm does not decrease as much as without the connection interval adaptation algorithm. With the connection interval adaptation algorithm there is even an increase in the voltage of the VCC at around 2500 s. The voltage drops for the measurement with the connection interval adaptation algorithm at around 1300 s and 3000 s, are when the connection interval changes in order to reduce the gap between the synchronization pulse and the connection interval. This increase the power consumption, since the data rate is changed here for a short duration. When comparing both measurements the average connection interval skips with the connection interval adaptation algorithm is 17.98 compared to 14 for the fixed connection interval skips. This means that the data rate is lower when using the connection interval adaptation algorithm. However, the voltage in the VCC with the connection interval adaptation algorithm does not decline as much as with the fixed connection interval skips. This is because of the increase in connection interval skips, since the connection interval adaptation algorithm can skip beyond the 14 connection interval skips and thus save more power.

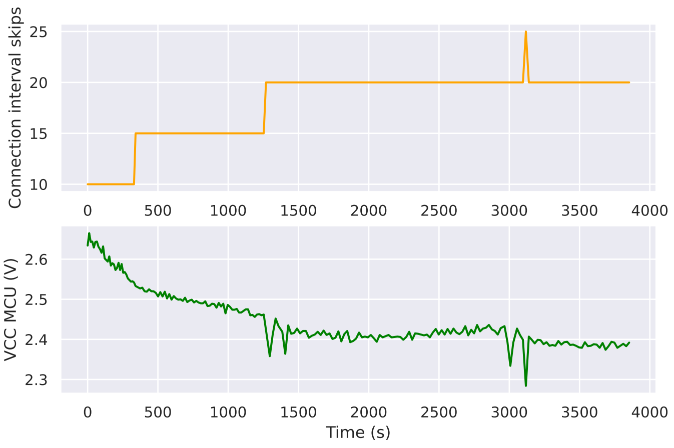
In Figure 6.6 the results of the second experiment is shown. The results show that with the same light scenario the connection interval adaptation algorithm increases the connection time. This is because of the increased connection interval skips beyond 14 than can be achieved with the connection interval adaptation algorithm. Also, when comparing the VCC voltage it shows that with the connection interval adaptation algorithm the voltage fluctuates more when the light is at 100%. This is because the data rate is high here (between zero and one connection interval skip) that does increase the power consumption significantly. Furthermore, at around 1000 s the VCC voltage of the connection adaptation algorithm is at almost 2.5 V compared to the fixed 14 connection interval skips at around 2.75 V. This is because the connection interval adaptation algorithm requires to have the voltage dropped in order to increase the connection interval skips, which takes time and in this case more energy. The average connection interval skips with the connection interval adaptation algorithm is 18.37, which is higher than the 14 connection interval skips mainly because of the 30 connection interval skips at the end of the measurement. However, for the connection interval adaptation algorithm the connection interval skips adapts to the VCC voltage of the MCU and thus utilizes the energy more effective by being able to be more active when the voltage is high in the super-capacitor.

6.4.1 Key Insights from Connection Interval Adaptation Algorithm Measurements

The results from both experiments show that the connection interval adaptation algorithm can increase the connection time of an intermittent BLE connection. This is primarily because the connection interval adaptation algorithm can skip more connection intervals than the current BLE 5.2 specification allows for at 15 connection intervals (for our implementation 14). When comparing the VCC of the connection adaptation algorithm measurements with the fixed connection interval skips measurements, it also shows that the connection adaptation algorithm can reduce the decay of the voltage in the VCC by increasing the connection interval skips. This will stabilize the voltage of the VCC, which will increase the connection time. However, the results also show that the data rate of the connection can decrease during low light settings due to the connection interval adaptation algorithm. Furthermore the connection interval adaptation algorithm can only change the connection interval skips depending on the increase or decrease of VCC voltage. This means that currently the connection



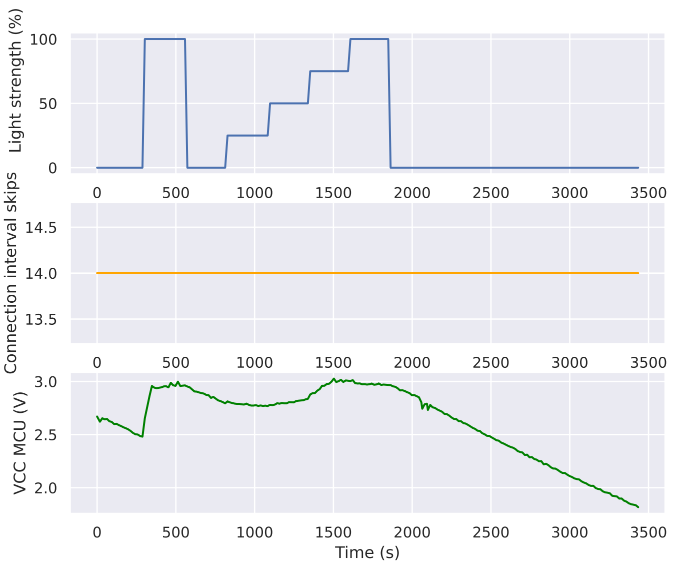
(a) Fixed 14 connection interval skips measurement.



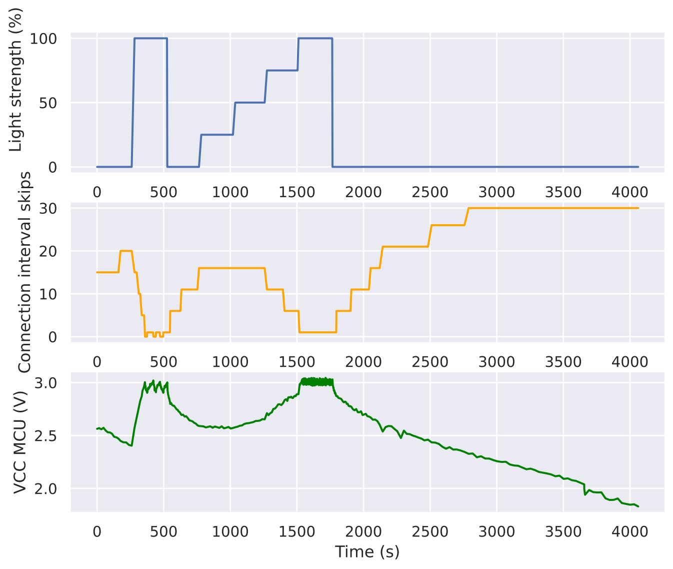
(b) Connection interval adaptation algorithm measurement.

Figure 6.5: **Experiment 1: The connection interval skips and the voltage of the VCC of the MCU at** 300 lx **during an intermittent BLE connection for (top) fixed at 14 of connection interval skips and (bottom) the connection interval adaptation algorithm. The connection parameters for both measurements is one second connection interval and zero connection interval skips.**

interval adaptation algorithm cannot preemptively reduce the connection interval skips if less energy is harvested, but only when the voltage drops. For this implementation the VCC voltages are mapped to the connection interval skips. More ideal would be that the connection interval adaptation algorithm could also implement the charging or discharging rate of the super-capacitor in order to change the connection interval skips more preemptively without having the voltage to drop to a certain level in order to have the maximum amount of skips. Moreover, the results must be interpreted with caution, since there are only four measurements compared here. More measurements must be done in order to fully back up the previous described claims. There was chosen for only a small data set, since these experiments take a considerable amount of time. Also, the light scenarios are not comparable with real life scenarios and the connection interval adaptation algorithm might not be suitable for every application. Finally, with these experiments the goal was to determine if the connection adaptation algorithm is effective and if the connection time could be further increased compared to the fixed connection interval skips, which is shown here.



(a) Fixed 14 connection interval skips measurement.



(b) Connection interval adaptation algorithm measurement.

Figure 6.6: **Experiment 2: The connection interval skips and the voltage of the VCC of the MCU at** 300 lx **during an intermittent BLE connection with a fixed light scenario (top) fixed at 14 of connection interval skips and (bottom) the connection interval adaptation algorithm. The connection parameters for both measurements is one second connection interval and zero connection interval skips.**

Chapter 7

Limitations and Future Work

This chapter describes the limitations and future work for this thesis. In Section 7.1 the limitation of this thesis is described. In Section 7.2 the recommendations for future work are presented.

7.1 Limitations

In the following subsections the limitations of this project are described.

7.1.1 Maybe Long Term Deployment Need To Be Tested?

The current implementation does not allow for long deployment testing due to a software bug in the intermittent BLE software (described in Section 5.2.3) that will timeout the connection at around 4000 s. This timeout does not occur with the default BLE software and is not related to a clock drifting issue, since the connection timed out even when tested with the maximum receive window. Trying to fix the bug was time consuming and the bug is not resolved yet. Most likely the cause of the bug is due to the intermittent BLE software a variable is reaching a limit that will cause the timeout.

Also, the Nowi NH2D0245 PMIC DC-DC converter does not operate under 2.5 V, meaning that when the voltage reaches under 2.5 V the harvested solar energy will not boost the voltage of the super-capacitor. In our experiments this limitation was not prominent. Note that during the algorithm measurements the voltage was also under 2.5 V. This is because the voltage is measured by the MCU after the radio operation is done, and since this requires significant amount of energy the voltage of the super-capacitor drops temporarily. This means that the voltage measured by the MCU is typically lower than when the voltage will be measured when the MCU is turned off. However, for long term deployment with a lower light intensity the device might never charge the super-capacitor if the voltage consistently drops under 2.5 V. This need to be addressed in future implementations by having a DC-DC converter that will boost at a lower voltage than 2.5 V, otherwise the device is not able to harvest enough energy.

7.1.2 Maybe The Setup Was Limited?

This work did not consider a battery powered device, nor does it make a comparison with a battery-powered device. The comparison between a battery powered and intermittent powered BLE device does not fit in the scope of this thesis. Adding a battery to the system instead of the super-capacitor would not change the way the software operates but only changes how much energy the system has. Also, a new revision of the hardware had to be made in order to connect a battery to the system. The most interesting comparison that could be made with a battery powered device, would be to compare the connection time. However, due to the connection timeout bug no long term deployment measurements can be done with battery powered. Hence this is why no batteries were used in this thesis.

In this work only one solar panel is used for the measurements. The main reason for this was to not harvest to much energy, such that the connection time for each measurement is shorted. With shorter connection times the measurements were more manageable and did not exceed the connection timeout bug value.

7.1.3 Maybe Other Hardware Platform Is Required?

The main hardware components that result in significant overhead are: (i) the external FRAM, where the checkpoint needs to be stored and restored, (ii) the external RTC, that adds idle time due to the synchronization pulse. The checkpointing in this implementation is not optimized since currently all data is stored from SRAM to FRAM and restored back. Most of this data is not used during the connection, such as the data used for advertising. Due to the complexity of the BLE software stack the process of determining what data should be checkpointed and what data does not need to be checkpointed (since the old data in FRAM is the same) would take a considerable amount of time. Also this would only optimize the current application and needs to be maintained for every new application. A more maintainable solution would be to operate completely in FRAM, which is not possible on the nRF52840 MCU. This way the overhead of transporting data from the SRAM and FRAM would be removed, which would be a considerable amount.

The overhead from the external RTC is that the synchronization time can happen in intervals of 250 ms and should always happen if the MCU is turned on before the connection event. This can result in considerable amount of idle time, where the MCU has to wait for the connection event after the synchronization. This overhead could be removed by having an external RTC that has the option of reading out the current clock tick value. Otherwise, an internal RTC on the MCU that can run separately from the MCU and has the capability of turning the MCU on or off like the external RTC does. To the best of our knowledge there is no ultra low power external RTC that has clock tick accurate registers nor a MCU with an embedded power switching RTC that is also capable of BLE communication.

7.1.4 Maybe Different Software Platform?

In this work the Packetcraft Protocol Software is used to implement intermittent software, however this was not the most power efficient BLE software stack. The closed source Nordic SDK is the most optimal BLE software for the nRF platform. On the current hardware the Nordic SDK with the maximum slave latency (in this thesis 14 connection interval skips) is not compared with the intermittent BLE software. This is not done because this was overlooked and realised at the end of the thesis. Therefore there was no time to test this. Other considerations of different MCU's and software stacks were not done, mainly because the checkpointing system was compatible in ARM environment and a lot of technical support was available from the partner NOWI for the nRF52840 chip.

7.1.5 Maybe a Hybrid Intermittent Solution Is More Optimal?

From the results it is concluded that the most optimal power efficient solution for the intermittent BLE software is to have a hybrid solution. This hybrid solution means turning off the MCU bewteen connection events at the data rates the intermittent BLE software is more power efficient and to put the MCU in sleep mode between connection events at data rates the default BLE software is more power efficient. The hybrid solution is most likely an improvement that is recommended for a future implementation, especially in combination with the connection interval adaptation algorithm where the data rate changes. However, currently the hybrid solution is not implemented in the intermittent BLE software since there was no time to implement this. In this thesis we found out where the intermittent BLE software is more power efficient than the default BLE software and how the intermittent BLE software could be improved.

7.1.6 Maybe BLE Protocol Needs Optimization?

A big limitation of the BLE standard is that there are a lot of optional features available that are not included by default in BLE devices. For this work one such feature was missing in the Packetcraft Protocol Software stack, which is the slave initiated anchor point placement [20, volume 6, page 3039]. This is a feature where the slave device can negotiate with the master device where the anchor point of the connection event starts. This way the idle time between the synchronization pulse and the connection event could be reduced without changing the connection interval. Furthermore, for Android devices the anchor point placement is always random and is decided by the Bluetooth driver. This means that for Android devices the current intermittent BLE software would not work optimally, since the time between the synchronization pulse and the connection event will always be random depending on how the Bluetooth driver of Android schedules the anchor point. Thus in order for intermittent BLE software to work properly, there has to be a BLE standard with guaranteed features that work on different platforms as well.

Furthermore, the results of the connection interval adaptation algorithm shows that increasing the connection interval skips beyond the current maximum slave latency value of 15 (for the BLE 5.2 specification [20, volume 3, page 1071]) can reduce the power consumption. This is beneficial for an intermittent device in order to harvest more power and continue operation. It is remarkable to see that in the BLE 4.0 specification the maximum slave latency value could be theoretically 31 [19] if the supervision timeout is 32 s at a connection interval of 1 s. Currently nothing is stopping us from having more than 15 connection event skips event if the current BLE software is version 5.2. Nonetheless, for intermittent BLE it would be advised that the older slave latency policy would be made available again, since our data has shown that this could improve the connection time.

7.2 Future Work

In future work we recommend that the hardware should be further optimized. Especially by executing the code in FRAM, which will remove the store and restore overhead. This will require a different MCU and changes in the checkpointing software. Also, more research must be done about how to synchronize the external RTC with the internal clock of the MCU. Ideally the synchronization pulse should be removed, but this requires a different external RTC than currently is implemented that is able to access the clock tick values. To the best of our knowledge no such external RTC is currently available that has access to the clock tick values, has power switching capabilities and is ultra low power. Hence, more research must be done in order to determine if these external RTC's exist and otherwise this thesis might be an inspiration for future work to develop this hardware.

Also, the software needs optimization in future work. Currently no hybrid solution is implemented that combines the intermittent BLE software with the default BLE software. In future work this hybrid solution should be implemented and tested in order to determine if this would increase the connection time. It is expected that the hybrid solution will be effective in combination with the connection interval adaptation algorithm, since the data rate can vary significantly depending on the VCC voltage of the MCU. Finally, the software bug should be resolved that makes the intermittent BLE software timeout at around 4000 s. This way long term deployment of the intermittent BLE software can be tested.

Furthermore, the current Packetcraft Protocol Software stack is not the most optimal software for this MCU. In future work having access to the most optimal software is required to have better performance for the intermittent BLE software. Also, then the intermittent BLE software can be tested against the state of the art non-intermittent BLE software at different data rates to test how power efficient the intermittent BLE software is. In addition it is recommended that the current architecture of intermittent BLE software will be used for exploring how other networking protocols can operate intermittent. Most of the principles that were used in this project, such as how the timing is synchronized and restored can be applied to other networking protocols like Zigbee, Z-Wave, Wi-Fi and LoRa.

In future work it is recommended that the connection interval adaptation algorithm would be revised. Currently the connection interval skips are mapped to a specific VCC voltage. This means that only when the voltage drops the connection interval skips increases. Thus some significant energy is wasted before the power consumption can be reduced. Therefore it would be better if the connection interval adaptation algorithm would have input from the energy source to determine the data rate preemptively. This could result in less VCC voltage drop before the right connection interval skip value is found.

Chapter 8

Conclusion

The goal of this thesis was to implement a BLE connection mode on an energy harvesting sensor with timekeeping and time synchronization in order to maintain a connection during power failures. This study has shown that running a BLE connection intermittently is possible by turning the MCU off between connection events and restoring the operation just in time for the next connection event. All the timing of the radio operations is mapped to an external RTC and is controlled by the MCU in order to have timekeeping and time synchronization. Furthermore a connection interval adaptation algorithm is implemented that manages the data rate depending on the measured voltage in the system by the MCU. Changing the data rate is done by skipping certain connection events, much like the BLE slave latency.

The results show that the implemented intermittent BLE software can sustain a BLE connection for more than an hour and can be more power efficient than the default non-intermittent BLE software at a low data rate of at least one packet every 10 s or more. Also, at the maximum BLE slave latency measured of one packet every 14 s the intermittent BLE software is almost two times more power efficient. Furthermore, with the implemented connection interval adaptation algorithm it is possible to skip more connection intervals than the current BLE 5.2 standard allows for. This results in a longer connection time when the connection interval adaptation algorithm is enabled.

The current implementation of the intermittent BLE software connection has some hardware and software implications that add significant overhead that still need to be addressed. For the hardware, mainly the overhead from storing and restoring all data to external non-volatile memory should be reconsidered. Our suggestion to remove the overhead of storing and restoring would be to operate entirely from non-volatile memory. Also, the external RTC currently can only synchronize in intervals of 250 ms, which results in long idle time when the MCU is turned on. In future work, more research has to be done on how a different external RTC could remove the requirement to be only able to synchronize in intervals of 250 ms. Furthermore, one software implication for intermittent BLE software is that due to a bug the connection always times out at around 4000 s. For long term deployment of intermittent BLE connections this must be resolved in future research.

To conclude this thesis is to the best of our knowledge the first work that shows that it can maintain a BLE connection intermittent. Also some effort has been made to improve the intermittent BLE connection by introducing a connection interval adaptation algorithm. This work has shown what the implications of an intermittent BLE connection is and how further improvements can be made. The methodology used in this work to run an intermittent connection can be applied to many networking system and can be an inspiration for any future works.

Acronyms

ATT Attribute Protocol. 6, 9, 10

BLE Bluetooth Low Energy. iii, 1–21, 24–40, 42–46, 48, 49

CHRT Cascaded Hierarchical Remanence Timekeeper. 3

CRC Cyclic Redundancy Check. 7

CusTARD Custom Time And Remanence Decay. 3

FRAM Ferroelectric Random Access Memory. 5, 14–19, 35, 38, 44, 46

GAP Generic Access Profile. 6, 11

GATT Generic Attribute Profile. 6, 10

GFSK Gaussian Frequency-Shift Keying. 6

GPIO General Purpose Input/Output. 19, 22

HCI Host Controller Interface. 6, 9

IoT Internet of Things. 1

ISM Industrial Scientific and Medical. 6

L2CAP Logical Link Control and Adaptation Protocol. 6, 9

LL Link Layer. 6, 11

MCU microcontroller. iii, 14–19, 21–29, 31–33, 35, 37–40, 42–46, 48

NFC Near-Field Communication. 11

OS Operating System. 19–21, 24

PHY Physical Layer. 6

PMIC Power Management Integrated Circuits. 17, 18, 43

QSPI Quad Serial Peripheral Interface. 18

RF Radio Frequency. 3, 4

RTC Real-Time Clock. iii, 3, 15–19, 21–25, 33, 35, 44, 46, 48

RTOS Real-Time Operating System. 20

SAADC Successive Approximation Analog-to-Digital Converter. 26

SMP Security Manager Protocol. 6, 9, 11

SPI Serial Peripheral Interface. 18

SRAM Static Random Access Memory.. 5, 35, 44

TARDIS Time and Remanence Decay in SRAM. 3

WSF Wireless Software Foundation. 20, 21

Bibliography

[1] Ambiq. AM18X5 Real-Time Clock with Power Management Family. `https://ambiq.com/wp-content/uploads/2020/10/Artasie-AM18X5-RTC-Datasheet.pdf`, 2019. Last accessed: Jun. 17, 2021.

[2] Apache. Apache NimBLE. `https://github.com/apache/mynewt-nimble`, 2021. Last accessed: Jun. 17, 2021.

[3] Archtoolbox. Recommended Lighting Levels in Buildings. `https://www.archtoolbox.com/materials-systems/electrical/recommended-lighting-levels-in-buildings.html`, 2021. Last accessed: Feb. 08, 2022.

[4] Naveed Anwar Bhatti and Luca Mottola. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In *Proc. IPSN*, page 209–219, Pittsburgh, USA, April 2017. ACM.

[5] Luca Buccolini, Paola Pierleoni, and Massimo Conti. Design and energetic analysis of a self-powered bluetooth low energy speed sensor. In *Proc. EEEIC*, pages 1–6, Florence, Italy, 2016.

[6] Bradford Campbell, Joshua Adkins, and Prabal Dutta. Cinamin: A perpetual and nearly invisible BLE beacon. In *Proc. EWSN*, page 331–332, Graz, Austria, 2016.

[7] Riccardo Cavallari, Flavia Martelli, Ramona Rosini, Chiara Buratti, and Roberto Verdone. A survey on wireless body area networks: Technologies and design challenges. *IEEE Communications Surveys Tutorials*, 16(3):1635–1657, 2014.

[8] Alexei Colin and Brandon Lucia. Chain: Tasks and channels for reliable intermittent programs. In *Proc. OOPSLA*, page 514–530, Amsterdam, Netherlands, October 2016. ACM.

[9] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proc. ASPLOS*, page 767–781, New York, USA, March 2018. ACM.

[10] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. Reliable timekeeping for intermittent computing. In *Proc. ASPLOS*, pages 53—-67, New York, USA, March 2020. ACM.

[11] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. Battery-free game boy. In *Proc. IMWUT*, volume 4, New York, USA, September 2020. ACM.

[12] Jasper de Winkel, Vito Kortbeek, Josiah Hester, and Przemysław Pawełczak. MPatch Checkpoint Library. `https://github.com/TUDSSL/ENGAGE/blob/master/software/libs/mpatch/README.md`, 2020. Last accessed: Jun. 17, 2021.

[13] Jasper de Winkel, Haozhe Tang, and Przemysław Pawełczak. Connected despite power failures: Intermittently-powered bluetooth that works, September 2016. submitted for publication.

[14] TU Delft. Intermittent computing to replace trillions of batteries. `https://www.tudelft.nl/en/stories/articles/intermittent-computing-to-replace-trillions-of-batteries`, 2021. Last accessed: Sep. 20, 2021.

[15] Abdul-Rahman El-Sayed, Kevin Tai, Mohammad Biglarbegian, and Shohel Mahmud. A survey on recent energy harvesting mechanisms. In *Proc. CCECE*, pages 1–5, May 2016.

[16] Ericsson. Cellular networks for massive IoT. `https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-networks-for-massive-iot--enabling-low-power-wide-area-applications`, 2021. Last accessed: Sep. 20, 2021.

[17] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. Pible: Battery-free mote for perpetual indoor BLE applications. *CoRR*, abs/1812.04717:1–4, 2018.

[18] Fujitsu. MB85RS4MT Memory FRAM. `https://www.fujitsu.com/uk/Images/MB85RS4MT.pdf`, 2018. Last accessed: Jun. 17, 2021.

[19] Core Specification Working Group. *Bluetooth Core Specification: Version 4.0*, volume 3, page 77. Core Specification Working Group, 2010.

[20] Core Specification Working Group. *Bluetooth Core Specification: Version 5.2*. Core Specification Working Group, 2019.

[21] Fredrik Häggström and Jerker Delsing. IoT energy storage - a forecast. *Energy Harvesting and Systems*, 5(3-4):43–51, 2018.

[22] Steven Lain Hearndon. An analysis of bluetooth low energy in the context of intermittently powered devices. Thesis, Clemson University, South Carolina, USA, 2016.

[23] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. Demo: A hardware platform for separating energy concerns in tiny, intermittently-powered sensors. In *Proc. SenSys*, page 447–448, Seoul, South Korea, November 2015. ACM.

[24] Josiah Hester and Jacob Sorber. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proc. SenSys*, pages 1–13, Delft, Netherlands, November 2017. ACM.

[25] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely execution on intermittently powered batteryless sensors. In *Proc. SenSys*, pages 17:1–17:13, Delft, Netherlands, November 2017. ACM.

[26] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P. Burleson, and Jacob Sorber. Persistent clocks for batteryless sensing devices. *ACM Trans. Emb. Comput. Syst.*, 15(4), August 2016.

[27] Robin Heydon. *Bluetooth Low Energy: The Developer's Handbook*, chapter 7–11. Pearson, 2012.

[28] Microchip Technology Inc. Attribute and Data Hierarchy. `https://microchipdeveloper.com/wireless:ble-gatt-data-organization`, 2021. Last accessed: Nov. 14, 2021.

[29] Seiko Instruments Inc. Chip Capacitors CPH3225A. `https://www.sii.co.jp/en/me/datasheets/chip-capacitor/cph3225a/`, 2021. Last accessed: Nov. 11, 2021.

[30] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. Quickrecall: A hw/sw approach for computing across power cycles in transiently powered computers. *J. Emerg. Technol. Comput. Syst.*, 12(1):8:1–8:19, July 2015.

[31] Kang Eun Jeon, James She, Jason Xue, Sang-Ha Kim, and Soochang Park. LuXbeacon—a batteryless beacon for green IoT: Design, modeling, and field tests. *IEEE Internet of Things Journal*, 6(3):5001–5012, June 2019.

[32] Pouya Kamalinejad, Chinmaya Mahapatra, Zhengguo Sheng, Shahriar Mirabbasi, Victor C. M. Leung, and Yong Liang Guan. Wireless energy harvesting for the internet of things. *IEEE Commun. Mag*, 53(6):102–108, June 2015.

[33] Giannis Kazdaridis, Nikos Sidiropoulos, Ioannis Zografopoulos, Polychronis Symeonidis, and Thanasis Korakis. Nano-things: Pushing sleep current consumption to the limits in IoT platforms. In *Proc. IoT*, Malmö, Sweden, 2020. ACM.

[34] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. In *Proc. ASPLOS*, pages 85—-99, Lausanne, Switzerland, 2020. ACM.

[35] Xiao Lu, Ping Wang, Dusit Niyato, Dong In Kim, and Zhu Han. Wireless networks with RF energy harvesting: A contemporary survey. *IEEE Commun. Surveys Tuts*, 17(2):757–789, 2nd Quart. 2015.

[36] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent computing: Challenges and opportunities. In *Proc. LIPIcs*, volume 71, pages 8:1–8:14, 2017.

[37] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent execution without checkpoints. *Proc. ACM Program. Lang.*, 1(OOPSLA):96:1–96:30, October 2017.

[38] ARM Mbed. Arm Mbed Cordio. https://os.mbed.com/docs/mbed-cordio/19.02/introduction/index.html, 2021. Last accessed: Jun. 17, 2021.

[39] Nordic. nRF Connect for Mobile. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=nl&gl=US, 2020. Last accessed: Jan. 29, 2021.

[40] Nordic. nRF Connect for Desktop. https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-desktop, 2021. Last accessed: Nov. 19, 2021.

[41] Nordic. nRF Sniffer for Bluetooth LE. https://www.nordicsemi.com/Products/Development-tools/nRF-Sniffer-for-Bluetooth-LE, 2021. Last accessed: Nov. 19, 2021.

[42] Nordic. nRF52 DK. https://infocenter.nordicsemi.com/pdf/nRF52_DK_User_Guide_v1.3.1.pdf, 2021. Last accessed: Nov. 19, 2021.

[43] Nordic. nRF52840. https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.6.pdf, 2021. Last accessed: Nov. 19, 2021.

[44] Nordic. nRF52840 Dongle. https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dongle, 2021. Last accessed: Nov. 19, 2021.

[45] Nordic. Software development kit for the nRF52 series and nRF51 series SoCs. https://www.nordicsemi.com/Products/Development-software/nrf5-sdk, 2021. Last accessed: Jun. 17, 2021.

[46] Nowi. NH2D0245-004 Energy Harvesting PMIC. https://media.digikey.com/pdf/Data%20Sheets/Nowi/NH2D0245_Datasheet.pdf, 2021. Last accessed: Jun. 17, 2021.

[47] Packetcraft. Packetcraft Protocol Software. https://github.com/packetcraft-inc/stacks, 2020. Last accessed: Jun. 17, 2021.

[48] Philips. Hue white ambiance 1-pack E27. https://www.philips-hue.com/nl-nl/p/hue-white-ambiance-1-pack-e27/8719514328167, 2021. Last accessed: Nov. 24, 2021.

[49] Google Play. Philips Hue Bluetooth. https://play.google.com/store/apps/details?id=com.signify.hue.blue, 2021. Last accessed: Nov. 24, 2021.

[50] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on RFID-scale devices. In *Proc. ASPLOS*, pages 159–170, Newport Beach, USA, March 2011. ACM.

[51] Saleae. Saleae Logic Pro 8 Analyzer. http://downloads.saleae.com/specs/Logic+Pro+8+Data+Sheet.pdf, 2018. Last accessed: Jun. 17, 2021.

[52] Saleae. Logic 2 Software Download. https://support.saleae.com/logic-software/sw-download, 2021. Last accessed: Nov. 24, 2021.

[53] Teodora Sanislav, George Dan Mois, Sherali Zeadally, and Silviu Corneliu Folea. Energy harvesting techniques for Internet of Things (IoT). *IEEE Access*, 9:39530–39549, March 2021.

[54] Cypress Semiconductor. Solar-powered BLE sensor beacon. `https://www.cypress.com/file/280631/download`, 2021. Last accessed: Jun. 17, 2021.

[55] Carlo Signer. Batteryless bluetooth low energy communication. Bachelor thesis, Eidgenössische Technische Hochschule Zürich, Zürich, Switzerland, 2017.

[56] STMicroelectronics. Software tool for power and ultra-low-power measurements. `https://www.st.com/resource/en/data_brief/stm32cubemonpwr.pdf`, 2021. Last accessed: Nov. 24, 2021.

[57] STMicroelectronics. STM32 Nucleo expansion board for power consumption measurement. `https://www.st.com/resource/en/data_brief/x-nucleo-lpm01a.pdf`, 2021. Last accessed: Nov. 24, 2021.

[58] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. *IEEE Commun. Surveys Tuts.*, 13(3):443–461, 3rd Quart. 2011.

[59] Philo Tang. Ultra-low-power architecture for wireless internet of things. Master thesis, Delft University of Technology, Delft, Netherlands, 2021.

[60] Pietro Tedeschi, Kang Eun Jeon, James She, Simon Wong, Spiridon Bakiras, and Roberto Di Pietro. Privacy-preserving and sustainable contact tracing using batteryless BLE beacons. *CoRR*, abs/2103.06221:1–11, March 2021.

[61] Priya Thanigai. FRAMs as alternatives to flash memory in embedded designs. `https://www.embedded.com/frams-as-alternatives-to-flash-memory-in-embedded-designs/`, 2012. Last accessed: Nov. 8, 2021.

[62] Kevin Townsend, Carles Cufí, and Robert Davidson. *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking.* O'Reilly, Sebastopol, USA, 2014.

[63] Alex S. Weddell, Michele Magno, Geoff V. Merrett, Davide Brunelli, Bashir M. Al-Hashimi, and Luca Benini. A survey of multi-source energy harvesting systems. In *Proc. DATE*, pages 905–908, 2013.

[64] Williot. Wiliot platform: IoT pixels. `https://wiliot.com/product/iot-pixel`, 2021. Last accessed: Jun. 17, 2021.

[65] Panasonic Electric Works. AM-1513CA Amorphous Silicon Solar Cell. `https://www.panasonic-electric-works.com/cps/rde/xbcr/pew_eu_en/ca_amorton_solar_cells_en.pdf`, 2020. Last accessed: Jun. 17, 2021.

[66] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *Proc. OSDI*, pages 17–32, Savannah, USA, November 2016. USENIX.

[67] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. Ink: Reactive kernel for tiny batteryless sensors. In *Proc. SenSys*, page 41–53, Shenzhen, China, November 2018. ACM.

[68] Taiyo Yuden. EYSKBNZWB Bluetooth Low Energy 5.0. `https://www.yuden.co.jp/jp/product/category/module/img/TY_BLE_EYSKBNZWB_DataReport_V1_0_20190227E.pdf`, 2019. Last accessed: Jun. 17, 2021.

[69] Zephyr. The Zephyr Project. `https://github.com/zephyrproject-rtos/zephyr`, 2021. Last accessed: Jun. 17, 2021.