Bachelor's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

Cost Efficient Wireless Sensor Network for Long Term Data Acquisition

Tomáš Reichl

Supervisor: Ing. Ladislav Sieger, CSc.
Study Program: Cybernetics and Robotics
May 2024

BACHELOR'S THESIS ASSIGNMENT

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

I. Personal and study details

Student's name: **Reichl Tomáš** Personal ID number: **499130**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Cost efficient wireless sensor network for long term data acquisition

Bachelor's thesis title in Czech:

Bezdrátová senzorová sí pro dlouhodobý sb r dat

Guidelines:

Research available technologies for low-power wireless sensor networks. Compare the suitability of these technologies for a mode of operation in which wireless sensors conserve battery power by reducing the wireless communication time frame while maintaining sufficient communication range for use in large buildings. Based on this research, choose a suitable development board and implement a laboratory prototype of a wireless sensor network for long-term temperature and humidity monitoring, focusing on a long battery life (which shall be experimentally evaluated) and low unit cost.

Bibliography / sources:

NIKOUKAR, Ali; RAZA, Saleem; POOLE, Angelina; GÜNE , Mesut; DEZFOULI, Behnam. Low-Power Wireless for the Internet of Things: Standards and Applications. IEEE Access. 2018, vol. 6, pp. 67893–67926. issn 2169-3536. Available from doi: 10.1109/ACCESS.2018.2879189.

KHALIFEH, Ala'; MAZUNGA, Felix; NECHIBVUTE, Action; NYAMBO, Benny Munyaradzi. Microcontroller Unit-Based Wireless Sensor Network Nodes: A Review. Sensors. 2022, vol. 22, no. 22, article no. 8937. issn 1424-8220. Available from doi: 10.3390/s22228937.

Name and workplace of bachelor's thesis supervisor:

Ing. Ladislav Sieger, CSc. Department of Physics FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **25.01.2024** Deadline for bachelor thesis submission: ___________

Assignment valid until:
by the end of summer semester 2024/2025

Ing. Ladislav Sieger, CSc. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt — Student's signature

©   VUT v Praze, Design: VUT v Praze, VIC

Acknowledgements

I would like to thank my supervisor, Ladislav Sieger, for offering me the opportunity to work on this project and for always being helpful and keeping me motivated.

I want to also thank Michael Brabec for sharing with me his experience with WSN design and helping me refine the basic concept, and Oxmi Kelruc for helping me with the ESP32 platform.

I would also like to express my immense gratitude to my family, who went out of their way to give me the support and space I needed.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských prací.

V Praze dne 24. května 2024

....
Tomáš Reichl

Abstract

The aim of this thesis is to evaluate the suitability of available wireless communication technologies for use in ultra-low power wireless sensor networks for long-term data acquisition. It focuses on environment monitoring applications, which tend to prioritize high reliability and energy efficiency over frequency of measurement.

The possibility of using inexpensive and wide-spread IoT chipset platforms is looked-into. Although often not being optimized for use in ultra-low power designs, their low-cost and popularity makes them a promising option, which may have the potential to make wireless sensor networks accessible to a wider range of applications.

A communication protocol based on the research of available technologies is then proposed and implemented on a development platform for evaluation. The goal is to create a prototype, on the basis of which a fully functional wireless sensor node could be developed if the proposed technologies prove to be viable.

Keywords: wireless sensor network, low-power, Bluetooth LE, ESP32-C3, FreeRTOS

Supervisor: Ing. Ladislav Sieger, CSc.

Abstrakt

Cílem této práce je porovnat vhodnost různých bezdrátových technologií pro použití v bezdrátových senzorových sítích určených pro dlouhodobý sběr dat. Zaměřuje se na aplikace monitorování prostředí, ve kterých se upřednostňuje velmi nízká spotřeba a vysoká spolehlivost na úkor množství a frekvence měřených dat.

Je zkoumána možnost využití levného a rozšířeného IoT hardware, který obvykle není optimalizovaný pro tuto aplikaci, ale díky své popularitě a nízké ceně se jedná o velmi atraktivní možnost, která má šanci zpřístupnit bezdrátové senzorové sítě pro větší škálu použití.

Na základě výzkumu je navržen komunikační protokol, který je implementován na vývojové platformě za účelem ověření jeho vlastnosti. Účelem je vytvořit prototyp, na jehož základě bude možné později založit plně funkční zařízení, osvědčí-li se navrhované postupy.

Klíčová slova: bezdrátová senzorová síť, low-power, Bluetooth LE, ESP32-C3, FreeRTOS

Překlad názvu: Bezdrátová senzorová síť pro dlouhodobý sběr dat

Contents

1 Introduction **1**

1.1 Motivation and goals 2

2 Research **5**

2.1 Comparison of WCPs 5

2.1.1 Cellular networks (LTE, 5G) . 5

2.1.2 Low power WANs (LoRa, LTE Cat-M1, NB-IoT) 6

2.1.3 Bluetooth 6

2.1.4 IEEE 802.11 (Wi-Fi) 7

2.1.5 IEEE 802.15.4 (Zigbee, Thread) 8

2.2 Hardware selection 9

2.2.1 Considered parameters 9

2.2.2 Viable hardware platforms . . 10

2.2.3 Hardware comparison 11

3 Hardware **15**

3.1 SoC 15

3.2 Development board 16

3.3 Device design 17

4 Application architecture **19**

4.1 Network architecture 19

4.2 Measurement data storage and transmission 23

4.3 Reliability 24

5 Firmware and software **25**

5.1 Node and server firmware 25

5.1.1 Server operation 25

5.1.2 Node operation 25

5.1.3 ESP-IDF framework 26

5.2 Bluetooth LE stack and implementation 27

5.2.1 Bluetooth-related terminology 27

5.2.2 Establishing connection (GAP) 28

5.2.3 Data transfer (GATT) 28

5.3 Data readout application 31

6 Experiments and results **33**

6.1 Current consumption testing . . . 33

6.2 Communication range testing . . 39

6.3 Testing at different TX power levels 41

7 Discussion, conclusion and future work **45**

A Bibliography **49**

Figures

3.1 Espressif ESP32-C3 block diagram 16

3.2 Seeed Studio XIAO ESP32-C3 board photo 17

3.3 WSN node schematic 17

3.4 WSN server and node 18

4.1 Example network communication diagram 21

5.1 Example readout application output plot 31

6.1 Current consumption of a node . 34

6.2 Current consumption of a node – communication cycle detail 35

6.3 Current consumption of a node – measurement cycles enabled 36

6.4 Current consumption of a server 37

6.5 Current consumption of a server – communication detail 38

6.6 Signal strength drop across a corridor 40

6.7 Signal strength drop across different floors 40

6.8 Signal strength drop across a corridor for different TX power levels 42

Tables

2.1 Comparison of wireless SoCs . . . 12

2.2 Estimated daily electric charge consumption of wireless SoCs 13

5.1 GATT services, characteristics and descriptors 29

6.1 Average current and electric charge consumption of nodes and server with default parameters 39

6.2 Comparison of average electric charge consumption at different TX power levels 41

6.3 Estimated daily electric charge consumption of a WSN node 43

Chapter 1

Introduction

Wireless sensor networks (WSN) have in recent years become an indispensable tool for many applications, including but not limited to environmental monitoring, portable medical devices and industrial sensors [1][2]. Compared to many traditional wired sensor systems, WSN nodes require less or no static infrastructure, can be powered with batteries for off-grid operation and may perform data processing in the node. WSNs have enabled new applications and are often being used in place of wired sensors due to their ease of installation.

Since WSNs may be deployed in many different ways with widely differing requirements, many wireless communication protocols (WCP) have been adapted or developed for use in WSNs. Many of these are also being used in consumer electronics, the Internet of Things (IoT) or mobile robotics [3][4].

When selecting the optimal WCP for a particular application, there are many different aspects that need to be considered. Some of the most important are:

- communication range,
- data rate and latency, and
- power consumption.

The microcontroller (MCU) platforms, on which WSN nodes are built, have significantly evolved in recent years. Thanks to advances in manufacturing and instruction set architecture (ISA) development, microcontrollers have been getting more power efficient, while the move from legacy 8-bit to modern 32-bit architectures like ARM or RISC-V has greatly expanded the features and improved the performance of the microprocessors.

Thanks to increased popularity of IoT, microcontrollers and wireless transceivers supporting one or more WCP are now often combined onto integrated modules or, in some cases, into one IC package or even a single silicon die. This usually improves cost

efficiency, requires less space in the finished product and has the potential to simplify development.

1.1 Motivation and goals

The focus of this thesis are wireless sensor networks used for long-term data acquisition. These are used in many applications which require a large amount of measurements gathered by multiple sensors over a long time period.

Due to the long term nature of these measurements in combination with the sensors being possibly located in hard to reach areas, it is crucial for the WSN nodes to have exceptional energy efficiency. This requirement influences not only the selection of utilized technologies – WCP, microcontroller and sensor type – to maximize energy efficiency, but also requires the mode of operation to be optimized. Unlike IoT, where all devices are usually permanently connected the network to allow accessing any device at any time, the communication in this WSN should only connect to each other at predetermined intervals. This will allow the nodes to turn off the radio for most of their operation, which should result in a significant efficiency increase, since wireless transmission is usually one of the highest power consuming operations of a WSN node.

Communication range is the second most important parameter to consider. In a usual application of this WSN, it is expected that the maximum distance between adjacent nodes should be on the magnitude of tens of meters, with possibly one or two walls blocking the line of sight between the two nodes.

Cost efficiency is another significant factor. Minimizing the unit cost of the nodes as well as the operational cost (cost of setup and installation as well as required maintenance) would make the finished product competitive against existing solutions. The unit cost needs to be addressed in the prototyping process by choosing cost-efficient components that, however, should not compromise on functionality and reliability.

The network topology is not strictly predetermined. There should be one central unit – a server – that collects measurement data from all nodes in the network and allows the user to read it out. Whether the nodes connect directly to the server in a star topology, or whether the nodes connect to each other in a mesh topology (or any other configuration) is not important, as long as the transmission is reliable and efficient.

Data rate is not especially crucial in this application, since the measurement data is not expected to be particularly large. However, the network may be configured to collect measurements over longer periods and then send them all at once, which could disqualify some WCPs with a particularly low data rate.

The goal of this thesis is to find suitable technologies and implement a prototype of the WSN described above. There should be a comparison of available WCPs and their suitability for the application. A suitable hardware platform that meets

the application's requirements should then be selected. Using a development board for the chosen hardware platform, a working prototype should then be implemented. This prototype should be used to prove, whether the selected WCP and hardware platform are, in fact, suitable for the application. However, the prototype itself is not required to be a fully functional WSN, only the basic functionality of the network needs to be implemented to allow for testing and evaluation of the chosen solution.

Chapter 2

Research

2.1 Comparison of WCPs

Nowadays, many different wireless technologies are used in WSNs and IoT. In this section, I will describe the most popular wireless communication protocols and compare, which one has the most advantageous properties for this application.

2.1.1 Cellular networks (LTE, 5G)

Cellular networks are wide area networks (WAN) that have been developed for wireless voice calls and simple text messaging. As their data transmitting capabilities were further developed for mobile Internet access, they also started to be used by other portable devices for data connectivity. Nowadays, LTE and 5G networks have achieved speeds on par with DSL broadband and are often used in fixed installations [5].

Cellular networks rely on signal coverage from fixed transceivers (cellular radio towers), which each serve many devices in a multi-kilometer radius. They usually operate on licensed frequency bands between approximately 800 and 2000 MHz. Access to cellular network is a paid service.

High data throughput and almost universal coverage of mobile networks have made them a popular option for IoT and sensor networks, which need to send a lot of data and/or require real-time remote access [4].

While the virtually unlimited range of WANs is advantageous for sensor networks since it effectively removes any limitation on maximum distance between nodes or from node to central, this is unnecessary for my application, where all sensor nodes will be concentrated on a small area. The relatively high power consumption of LTE and the dependence on a network operated by a third party with the associated monetary costs makes this option unsuitable for my WSN.

2.1.2 Low power WANs (LoRa, LTE Cat-M1, NB-IoT)

Since most IoT and WSN applications tend to prioritize energy efficiency over high data rate, new WAN technologies have been developed which utilize many power saving techniques. Some of these are based on existing mobile cell network technologies, including LTE Cat-M1 and NN-IoT, which are based on 4G LTE. Others, like LoRa or SigFox, have been developed independently and usually operate on unlicensed ISM (Industrial, Science, Medical) radio bands, most commonly on 868 / 915 MHz (depending on region) or 2.4 GHz [5][6].

Although the energy efficiency of Low Power WANs is better than that of conventional cellular networks, its other properties still make it not an ideal choice for my application.

2.1.3 Bluetooth

Bluetooth is a personal area network (PAN) operating on the 2.4 GHz ISM band. There are two versions of Bluetooth – Bluetooth Classic and Bluetooth Low Energy (Bluetooth LE / BLE). Bluetooth (Classic) was originally specified in the IEEE 802.15.1 standard. Currently, Bluetooth is maintained and developed by Bluetooth SIG [7].

Bluetooth Classic is the original Bluetooth protocol, which has been developed for use with consumer electronics. Every Bluetooth Classic device supports a 1 Mb/s physical layer (PHY) using FM modulation and may support optional 2 Mb/s or 3 Mb/s PHYs using PSK modulation.

Bluetooth LE was introduced in Bluetooth Core Specification version 4.0. It has a mandatory 1 Mb/s PHY and can support an optional 2 Mb/s PHY, both using FM modulation. To increase effective range and link reliability, devices can implement optional coded PHYs, which utilize forward correction codes with the 1 Mb/s PHY, resulting in 500 kb/s (for 2 coded bits per uncoded bit) or 125 kb/s (for 8 coded bits per uncoded bit) data rates. With increased transmit power of up to +20 dBm, the Bluetooth LE Long Range devices have a significantly increased range over Bluetooth Classic [8].

Bluetooth normally operates in point-to-point mode, where a master device connects directly to a slave device. However, since version 5.0, Bluetooth Mesh is an optional part of the standard. It is built upon Bluetooth LE using uncoded 1 Mb/s PHY [7][9].

Bluetooth Classic is not suitable for my application due to its relative low range and energy efficiency. Bluetooth LE is however much more optimized for use in WSNs and also has longer range. Thanks to the relative simplicity of the protocol and transceiver hardware which results in good energy efficiency even with increased transmit RF power [3], Bluetooth LE is suitable for my application.

2.1.4 IEEE 802.11 (Wi-Fi)

Wi-Fi is a local area network (LAN), which operates on the unlicensed ISM bands of 2.4, 5 or 6 GHz. It is the main wireless LAN technology used in consumer electronics for Internet access. Its ubiquity makes it one of the most common choices of WCP in consumer IoT.

Wi-Fi is defined by the IEEE 802.11 family of standards. The most up-to-date version, at the time of writing, is Wi-Fi 6, which is defined by the IEEE 802.11ax standard [10][11]. While newer versions of Wi-Fi tend to focus on increasing data throughput and maximum number of connected devices, they often also bring optimizations to energy efficiency [12].

Wi-Fi 4 and newer support multiple frequency bands – 2.4 GHz, 5 GHz and (introduced in Wi-Fi 6E) 6 GHz. The benefit of the higher frequency bands lies in greater data rates, which are a result of less congestion in this spectrum and wider channel width. However, communication range at these frequencies is lower compared to 2.4 GHz, since they are absorbed by solids more than lower frequency signals, and they also tend to be less energy efficient. This, in combination with lesser need for high data rates, results in most IoT applications only using the slower 2.4 GHz band.

Wi-Fi is usually operating in infrastructure mode, which means that devices (stations) connect to a wireless access point (base station) in a star topology. All communication thus goes through the access point and every device has to be in its range. There may be multiple access points in one network to extend it, either connected wirelessly (acting as repeaters) or wired (usually through IEEE 802.3 – Ethernet).

Alternatively, Wi-Fi can also work in ad-hoc mode, where stations connect directly to each other in mesh topology. The implementation is often vendor specific. The consumer standard Wi-Fi Direct uses this mode, although it only allows communication between two devices at a time and cannot connect multiple devices in a mesh network.

The usual range of 2.4 GHz Wi-Fi is tens of meters without obstacles, however, this significantly depends on transmit power, antenna gain and radiation pattern and if there are any obstacles in the signal path. Current consumption is similarly dependent on transmit power and other parameters as well as the specific transceiver's hardware design.

Wi-Fi meets many requirements of my WSN, although its energy efficiency is usually worse than that of Bluetooth Low Energy, which may be a better choice for my application.

2.1.5 IEEE 802.15.4 (Zigbee, Thread)

The IEEE 802.15.4 family of standards specifies the PHY and MAC for ultra-low power, low data rate WCPs [13]. There are many WCPs which utilize this specification, including Zigbee and Thread.

Zigbee is an IoT protocol, which is developed by Connectivity Standards Alliance. Unlike Bluetooth and Wi-Fi, Zigbee was specifically designed for use in embedded electronics, industrial control and IoT. It aims to be a low-power and low-cost communication technology with sufficient data-rate for many IoT and WSN applications.

Zigbee can utilize two of the PHYs defined by the IEEE 802.15.4 standard, one operating at 2.4 GHz ISM band and the other operating at 868 / 915 MHz ISM band. It is not required for Zigbee devices to implement both PHYs. Power requirements, communication range and data rate depend on which PHY is used. Data is transmitted in small packets with maximum data rate ranging between 48 kb/s and 1 Mb/s.

Zigbee supports star, tree and mesh topologies. There are multiple device roles in a Zigbee network. Each network contains a coordinator which creates and controls the network and manages its security, storing cryptographic keys. Routers extend the networking, acting as repeaters (they are not present in star topology networks). Finally, end devices connect to the network through a coordinator or a router.

Zigbee and other IEEE 802.15.4-based WCPs tend to be very energy efficient, matching or exceeding Bluetooth LE. Thanks to being developed specifically for low-power embedded electronics, Zigbee is a suitable WCP for my application.

To summarize, I have ruled out wide area networks from the selection, because many of their advantages over other WCPs are not applicable to my WSN, which would be mostly affected by the weaknesses of WAN technologies. This leaves only LAN or PAN protocols. From those, I have determined, that Bluetooth Low Energy and Zigbee (and possibly other protocols using the IEEE 802.15.4 standard) are the most suitable WCPs for my application.

In these comparisons, I have only considered open standards or very wide-spread technologies. There are other, proprietary WCPs, which may be suitable for my needs; however, these could bring other disadvantages in the form of smaller user base or lesser hardware selection and result in higher cost of the solution or possibly licensing fees.

2.2 Hardware selection

As was already stated, the aim of this thesis is to create a prototype that will proof the viability of the selected protocol and communication strategy for the specific use case of long-term monitoring WSN. There are several key differences from a finalized product, which include the following.

- The prototype will be based on a development board, while the finished product would use a custom PCB.
- There is no need to power the prototype from batteries – powering from USB or a laboratory power supply is sufficient for the development, while communication range tests and other operations, which could require the device to be moved away from static power supplies, can be powered with a USB power bank.
- Data storage and readout may be simplified (and less user-friendly) when compared to the finished product.

2.2.1 Considered parameters

There are several important parameters that need to be considered when selecting the microcontroller and wireless communication chip (or a combined wireless microcontroller or SoC), as well as the development board, on which the WSN prototype will be built.

Wireless connectivity. As I determined in section 2.1, the selected hardware should support either Bluetooth LE or Zigbee, or possibly both. Higher maximum supported RF power level is desirable, as it should allow for longer communication range. External antenna connector is preferred over a PCB trace antenna.

Sleep and RTC. The WSN nodes will operate in a way, where they will be inactive for very long periods of time, significantly more in fact, then actively communicating or measuring. Thus, both the microcontroller and the communication chip need to be able to enter sleep or powered-down mode, where their power consumption will be significantly reduced. However, since the node needs to perform periodic measurements and communicate with other devices in the network, there needs to be a real time clock (RTC) that can wake the microcontroller. This can either be a part of the microcontroller or added as an external component.

Power consumption. There are three power modes which are important for our application. As was already mentioned, the node will spend most of the time in a sleep mode. The microcontroller will then periodically wake up and take a measurement

using a connected sensor – active mode. Then, either after every measurement or after a longer period, the wireless communication will be enabled and the measured data will be transmitted through the network – radio mode. The total power consumption depends on the amount of time the node spends in each of these modes – this is not entirely possible to accurately predict, since it may depend on many different factors including the relative distance of nodes, the obstacles between them, distance of hops from node to server and, of course, the measurement and communication periods, which can differ between applications.

Firmware development and library support. Due to the relative complexity of implementing wireless communication, IoT applications are usually built upon a framework or a library, which implements parts of the WCP's layer stack and provides the user with an API to use in their application. Also, if the framework has good third party library support, it has the potential to greatly simplify development. In my application, this would be especially useful for implementing support for new sensors. Since many of the currently available frameworks are built specifically for one hardware platform, this will also be considered when choosing the microcontroller.

2.2.2 Viable hardware platforms

I have looked at a number of BLE and Zigbee wireless platforms and have selected the following 3 as viable options.

Texas Instruments SimpleLink CC13xx / CC26xx. These are low-power wireless microcontrollers. They are based on various ARM Cortex M-series CPU cores and a variety of wireless cores. All CC13xx microcontrollers include a sub-1 GHz radio with support for IEEE 802.15.4 (including sub-1 GHz Zigbee) and various other protocols. All CC26xx, but also some CC13xx microcontrollers, support Bluetooth LE. Additionally, some CC13xx and CC26xx microcontrollers also support 2.4 GHz Zigbee [14].

These microcontrollers can be programmed using Texas Instruments' SimpleLink Low Power F2 SDK. It is a C framework built on FreeRTOS and Texas Instruments' own ZigBee and Bluetooth stacks [15].

Espressif ESP32-C Series. This is a family of wireless SoCs based on single-core RISC-V microprocessors. The ESP32-C2 (ESP8684) and ESP32-C3 (ESP32-C3 and ESP8685) SoCs support Wi-Fi 4 and Bluetooth 5 LE [16][17], while the ESP32-C6 supports 2.4 GHz IEEE 802.15.4 (Zigbee and Thread) in addition to 2.4 GHz Wi-Fi 6 and Bluetooth 5 LE [12].

All microcontrollers in the ESP32 family are programmed using the Espressif IoT Development Framework (ESP-IDF) [18]. This SDK is based on a modified version of FreeRTOS and wireless stacks based on existing open source solutions – BlueDroid or

MyNewt NimBLE for Bluetooth and ZBOSS for Zigbee. In addition to low-power and low-cost oriented ESP32-C and ESP32-H Series SoCs, which use single core RISC-V CPUs, the framework also supports single- or dual-core Tensilica Xtensa processors used in the original ESP32 or the ESP32-S Series SoCs.

Nordic Semiconductor nRF52 Series. This is a family of Bluetooth LE SoCs based on single-core ARM Cortex-M4 CPUs. In addition to Bluetooth 5.4 LE, some SoCs in this series also support 2.4 GHz IEEE 802.15.4 (Zigbee and Thread) [19].

Nordic supports these SoCs through their nRF Connect SDK [20][21]. It is built on the Zephyr RTOS, Nordic Semiconductor's own Bluetooth LE stack and ZBOSS ZigBee stack. In addition to this, nRF52 Series is also supported by Apache MyNewt, an open source RTOS and framework which includes the NimBLE Bluetooth LE stack.

2.2.3 Hardware comparison

I have selected 5 candidates from the hardware platform selection presented in section 2.2.2. In table 2.1, I list datasheet values of the parameters which I determined to be significant for my application in section 2.2.1.

Both Texas Instruments MCUs are a great low power option, having the most energy efficient CPU from the selection. Nordic Semiconductor nRF52840 consumes more current, approximately 50 to 85 % more for equivalent operations (higher CPU current consumption would be partially caused by higher clock speed as compared to the TI MCUs). Both Espressif chips have far higher datasheet current consumption as compared to both other vendors.

When it comes to radio capabilities, both Espressif chips support much higher maximum TX power than their competitors – +21 dBm for ESP32-C3 or +20 dBm for ESP32-C6. Also, their RX sensitivity is on par or slightly better than the other 2.4 GHz capable SoCs. The higher TX power is one of the causes of the significantly higher current consumption which, at the highest supported TX power, is more than 10× that of the competitors.

To find out whether the lower energy efficiency of the Espressif chipsets would significantly affect the application, I calculated the daily electric charge consumption[1] for the following example scenario:

- measurements every 15 minutes, which require 10 seconds of CPU active time each,

[1] I use electric current and electric charge instead of power and energy in my calculations, because every component of the system is powered by 3.3 V outputted by a voltage regulator, whose efficiency might change depending on the current consumption of the entire system. Thus, in my opinion, using current and charge consumption is more appropriate in this application.

SoC	Bluetooth LE		ZigBee			Current consumption				
	TX max power (dBm)	**RX sensitivity[2] (dBm)**	**PHY freq. band**	**TX max power (dBm)**	**RX sensitivity (dBm)**	**CPU active (mA)**	**Deep sleep[3] (μA)**	**Radio RX (mA)**	**Radio TX 0 dBm (mA)**	**Radio TX max power (mA)**
Texas Instruments CC1312R [14]	—	—	sub-1 GHz	+12	-110	2.89[4]	2.78	5.8	8.0	24.9
Texas Instruments CC2652R [22]	+5	-97	2.4 GHz	+5	-99	3.39[4]	3.2	6.9	7.0	9.2
Espressif ESP32-C3 [17]	+21	-97	—	—	—	17[5]	5	∼ 60[6]	∼ 100[6]	∼ 290[6]
Espressif ESP32-C6 [12]	+20	-98.5	2.4 GHz	+20	-104	19[5]	7	60[7]	92[8]	277[8]
Nordic Semiconductor nRF52840 [19]	+8	-95	2.4 GHz	+8	-100	6.3[9]	3.16	10.1	10.8	16.4

Table 2.1: Comparison of wireless SoCs

[2] 1 Mb/s PHY, 0.1 % bit error rate (BER)
[3] With RTC and memory retention enabled
[4] 48 MHz CPU clock, running CoreMark, peripheral power consumption not included
[5] 80 MHz CPU clock, active, all peripheral clocks disabled
[6] No value for Bluetooth LE radio current provided – value is approximated from Wi-Fi current consumption and values from ESP32-C6 datasheet
[7] Datasheet value includes CPU idle current at 160 MHz with all peripheral clocks disabled, which was subtracted to get this value
[8] Datasheet value includes CPU active current at 160 MHz with all peripheral clocks enabled, which was subtracted to get this value
[9] 64 MHz CPU clock, running CoreMark from flash, not using DC/DC regulator

- communication every 3 hours, which requires 30 seconds of RX and 10 seconds of TX (at 0 dBm TX power to allow for an equivalent comparison) with CPU active for the whole time, and
- deep sleep in between.

This is a rough estimate made before the application architecture was known, so the chosen time durations are not representative of the actual mode of operation. Also, this only compares the SoC electric charge consumption and not that of the sensor and supporting components. However, since the power requirements of the final application greatly depend on the environment and how the mode of operation of the network is configured, this estimate was deemed sufficient to compare the SoCs to each other.

SoC	Estimated daily electric charge consumption (mAh)
Texas Instruments CC1312R	1.65
Texas Instruments CC2652R	1.90
Espressif ESP32-C3	12.38
Espressif ESP32-C6	12.97
Nordic Semiconductor nRF52840	3.23

Table 2.2: Estimated daily electric charge consumption of wireless SoCs

As can be seen in table 2.2, the electric charge (energy) consumption of Espressif chips is approximately 3× to 6× higher than that of other chips. However, if the SoCs were to be powered by a typical 18650 lithium-ion battery with 3000 mAh capacity, the Espressif chipsets would (without accounting for other components and power inefficiencies) still run for over 200 days. Given that current consumption of other parts of the node would be virtually identical regardless of which SoC is chosen, the difference between the SoCs is likely to be much smaller.

The Texas Instruments SoCs have the best energy efficiency from the selection. However, I have decided against using them. One reason was the cost of development boards – I could only find official development boards for these chipsets, which cost over 1000 CZK and would make the development fairly expensive since multiple boards are needed for testing. Also, compared to the software development options for the other chipsets, I have found the Texas Instruments' SimpleLink SDK to be fairly closed down with far less community material being available.

The Espressif chipsets are a very popular option for hobbyists interested in IoT and other wireless projects. Thus, over the years, a large community has formed, providing help in forums and releasing third party libraries for use with the ESP-IDF framework. Similarly, although arguably to a lesser extend, the Nordic Semiconductor's chipsets are also a popular option in this space.

In the end, I have decided to use the Espressif ESP32-C3 in my application's prototype for the following reasons. Despite the negative impact on its energy efficiency, the ESP32-C3's radio capabilities for Bluetooth were the best out of the selection – it has the highest maximum TX power and second best RX sensitivity. Also, not only the chipset and modules (which integrate SoC, memory, antenna or antenna connector and supporting components), but also the first party development boards are very inexpensive. As an example, the ESP32-C3-MINI-1U-H4 module costs around 50 CZK and the ESP32-C3-DevKitC-02 development board costs under 200 CZK.

I have considered the Espressif ESP32-C6 as a potentially better option, since it also has Zigbee in addition to BLE. However, because the selection and purchasing of development boards for this project was done in spring of 2023, only shortly after the ESP32-C6 became widely available on the market, almost no development boards for it were available at the time, and none of the few available ones supported external antenna connector. Also, at the time, the support for the chipset in the ESP-IDF framework was incomplete, making it impossible to evaluate all of its features.

The Nordic Semiconductor nRF52840 seems to also be a very good option. However, I found the Espressif chipsets to have more useful features for the prototyping process. Also, both the chipsets and development boards are more expensive than the Espressif ESP32-C family.

I have selected the Espressif ESP32-C3 SoC for its combination of good RF performance, well-documented SDK with great community support, low cost and sufficient energy efficiency. The SoC supports Bluetooth LE and Wi-Fi WCPs, I will be using Bluetooth LE due to its better energy efficiency.

The official development framework, ESP-IDF, supports two Bluetooth stacks. I have decided to use the Apache MyNewt NimBLE stack, since it is less resource intensive than the other option, BlueDroid. Also, this would allow me to port the wireless protocol from my application with relatively few changes to any SoC supported by Apache MyNewt, including the Nordic Semiconductor nRF52 Series.

As a development board, I have chosen the Seeed Studio XIAO ESP32-C3. This is a very compact development board, which can also be used as an SoC module. It was one of the few development board options with an external antenna connector available at the time of purchase (spring of 2023), while also being inexpensive and including additional features which could potentially be useful for my application.

Chapter 3

Hardware

In this section, I will describe the hardware architecture of the selected development board and of the WSN node prototype.

3.1 SoC

The ESP32-C3 has a single RISC-V core that can run at either 80 MHz or 160 MHz generated from an external crystal clock or internal fast RC oscillator. It includes 400 KiB of SRAM and 8 KiB of RTC memory, which can retain data in deep-sleep mode. Some variants of the ESP32-C3 SoC also include 4 MiB of on-package flash memory, up to 16 MiB of external QSPI flash is also supported.

The selected development board uses the ESP32-C3FN4 chip variant which includes the aforementioned 4 MiB of on-package flash and has an on-board 40 MHz crystal oscillator as well as a low-speed 32.768 kHz crystal oscillator for RTC [17][23].

The SoC supports multiple low-power modes. In figure 3.1, the block diagram shows which modules are powered in either the light or deep sleep modes. While this prototype does not use the sleep modes, the power consumption in these modes will be tested, since they will be crucial to achieve sufficiently low energy consumption in the final application.

The SoC can be programmed without the use of a dedicated programmer thanks to a bootloader contained in the ROM memory. The CPU can be programmed using UART or the internal USB Serial / JTAG controller [24]. My application also uses this integrated controller for serial communication between the server and a data-readout device.

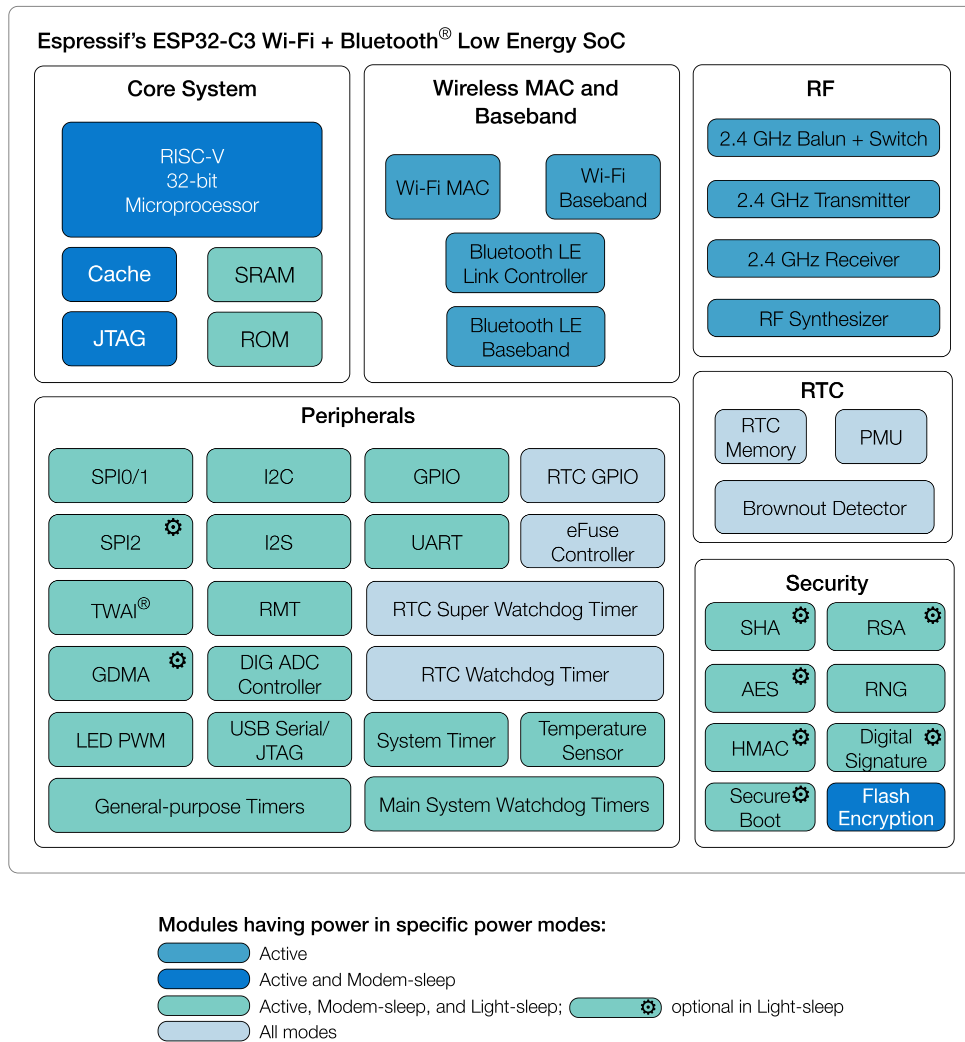


Figure 3.1: Espressif ESP32-C3 block diagram [17]

3.2 Development board

Seeed Studio XIAO is a series of miniature development boards. They come with USB-C connectors and a unified 20×17.5 mm footprint with 14 pins. In addition to 2.54 mm pin headers for use as a development board, these pins are extended into castellated holes, which allow for the development board to be soldered to a custom PCB to be used as a microcontroller module [25].

The XIAO ESP32-C3 board includes an LDO linear voltage regulator and a Li-Ion battery charger. The ESP32-C3FN4 SoC and all supporting components are placed under an RF shield. Li-Ion battery (in a 1S configuration) can be connected by soldering wires to pads on the underside of the board (or, when used as a wireless SoC module, by reflow soldering the pads on the module to pads on a custom PCB) [26]. The USB-C is connected to the integrated USB Serial / JTAG controller of the ESP32-C3 and can

Figure 3.2: Seeed Studio XIAO ESP32-C3 board photo [26]

be used for flashing the SoC [23][17]. An antenna can be connected to a U.FL coaxial connector (a flexible PCB antenna is included with the board). There are also two push switches (boot mode and reset) and a charging indicator LED.

3.3 Device design

Design of the prototype devices is very simple, as can be seen on the schematic in figure 3.3 and on the photo in figure 3.4. Since most of the required components is already present on the development board, the only required external component is a sensor. In the prototype, a DHT11 temperature and humidity sensor (U2) is connected to GPIO 3 of the ESP32-C3. It is powered with 3.3 V from the board's linear regulator.

Both server and node use the same firmware. To switch between server and node roles, GPIO 2 is pulled up internally and when shorted to ground using a jumper link (JP1), the board switches to server role.

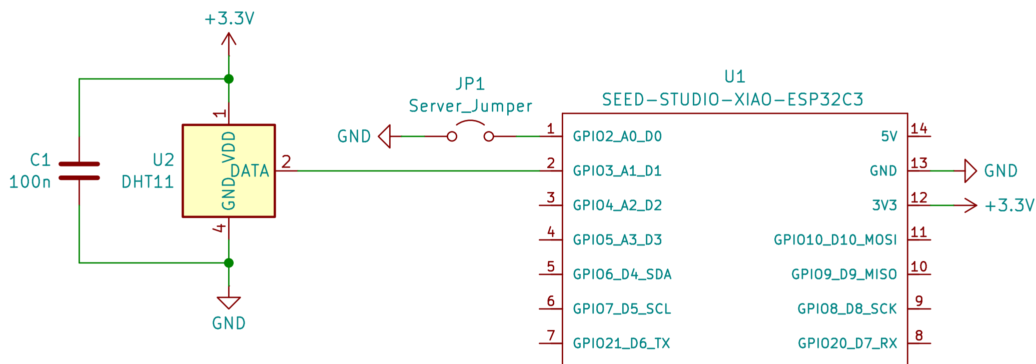


Figure 3.3: WSN node schematic

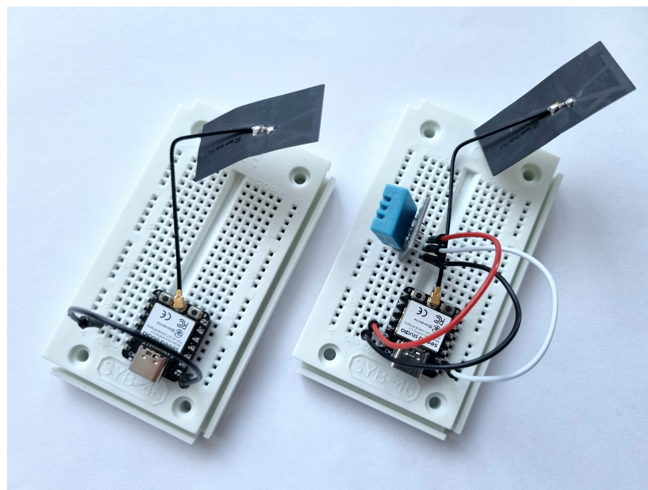


Figure 3.4: WSN server (left) and node (right)

Chapter 4

Application architecture

The main features of this application are the following:

- collecting measurements using a sensor connected to a node,
- transmitting data from node to server,
- storing measured data, and
- transferring the data to a readout device.

In this chapter, I will look in greater detail into the data transmission and storage features, since they influence the application architecture the most. I will describe the decisions I have made when designing these aspects of the applications, which I had to consider in the implementation of the device firmware and accompanying software.

Also, because of the application's requirements for long term reliable operation with minimal maintenance, I will briefly discuss the consideration I had to make when designing and implementing the firmware.

4.1 Network architecture

The network architecture design has been largely influenced by the properties and limitations of the selected WCP – Bluetooth LE. In this section I will only mention those that were important for the design process – see section 5.2 for a more detailed description of Bluetooth LE.

Due to the communication range and network structure flexibility requirements of my application, combined with the relatively short range of Bluetooth LE, simple star networks cannot be used. Other topologies, which allow for other nodes in the network

to relay information to the central from nodes, which are not in its range, should be used. The most common options are tree and mesh networks.

Since ESP32-C3 supports the optional Bluetooth Mesh extension of Bluetooth LE which allows for simple implementation of a mesh network, I considered using it for my application. However, after further research and testing of the ESP-BLE-MESH stack, several disadvantages of Bluetooth Mesh became apparent that made it unsuitable for my application.

- Small message size – due to only using the GAP profile to transmit data, a single Bluetooth Mesh message is limited to 11 bytes of payload. The protocol supports message segmentation, which however decreases the payload size to 10 or 9 bytes. The maximum message size is then increased to 384 bytes, but the data rate is very low due to segmentation, which creates a significant overhead of headers sent with each message [9].
- No Bluetooth LE Long Range support – Bluetooth Mesh only supports the uncoded 1 Mb/s PHY and cannot utilize the significantly increased range of the coded PHY variants.
- Limited low power support – Bluetooth Mesh supports low-power devices that disconnect and sleep to save power. However, to keep the network running, they always need to be in range of a permanently powered device which stores and relays messages for them. Otherwise, the network would need to be restarted every time devices need to connect and, due to the relatively high complexity of the Bluetooth Mesh protocol and security system, this operation would greatly increase connected time and thus energy consumption.

I have therefore decided to use a protocol built on top of “traditional” point-to-point Bluetooth LE connections using the connection-oriented GATT protocol. Due to the specific requirements of my application I have designed this protocol from scratch with focus on simplicity, reliability and extendibility.

My protocol creates a tree network with a single server device and many node devices, that other nodes, which are out of range of the server, can connect to. The network naturally forms layers based on their distance from the server. Two types of data are transmitted through the network:

- configuration changes (including server’s current time for RTC synchronization), which propagate from server to nodes, and
- measurement data, which propagates from nodes to server.

As was already mentioned, the wireless network is powered down most of the time to save energy and only starts up periodically, the devices using RTCs to stay synchronized.

After the network starts up, the nodes are in peripheral mode, advertising their presence, while the server is in central mode and scans for nearby nodes (see section 5.2.1 for explanation of Bluetooth-related terminology that I use in this thesis).

After the server finds an advertising node, it connects to it, sends configuration data to it and retrieves stored measurement data. This includes measurements collected by the node and also measurements from nodes on lower network layers. After the transmission is completed, the node switches from peripheral to central mode, effectively turning into a server. It then proceeds to scan for nearby nodes, pass on its configuration and collect measured data. In figure 4.1, there is an example of data transmission in a network with a server and 3 nodes in 2 layers.

Figure 4.1: Example network communication diagram

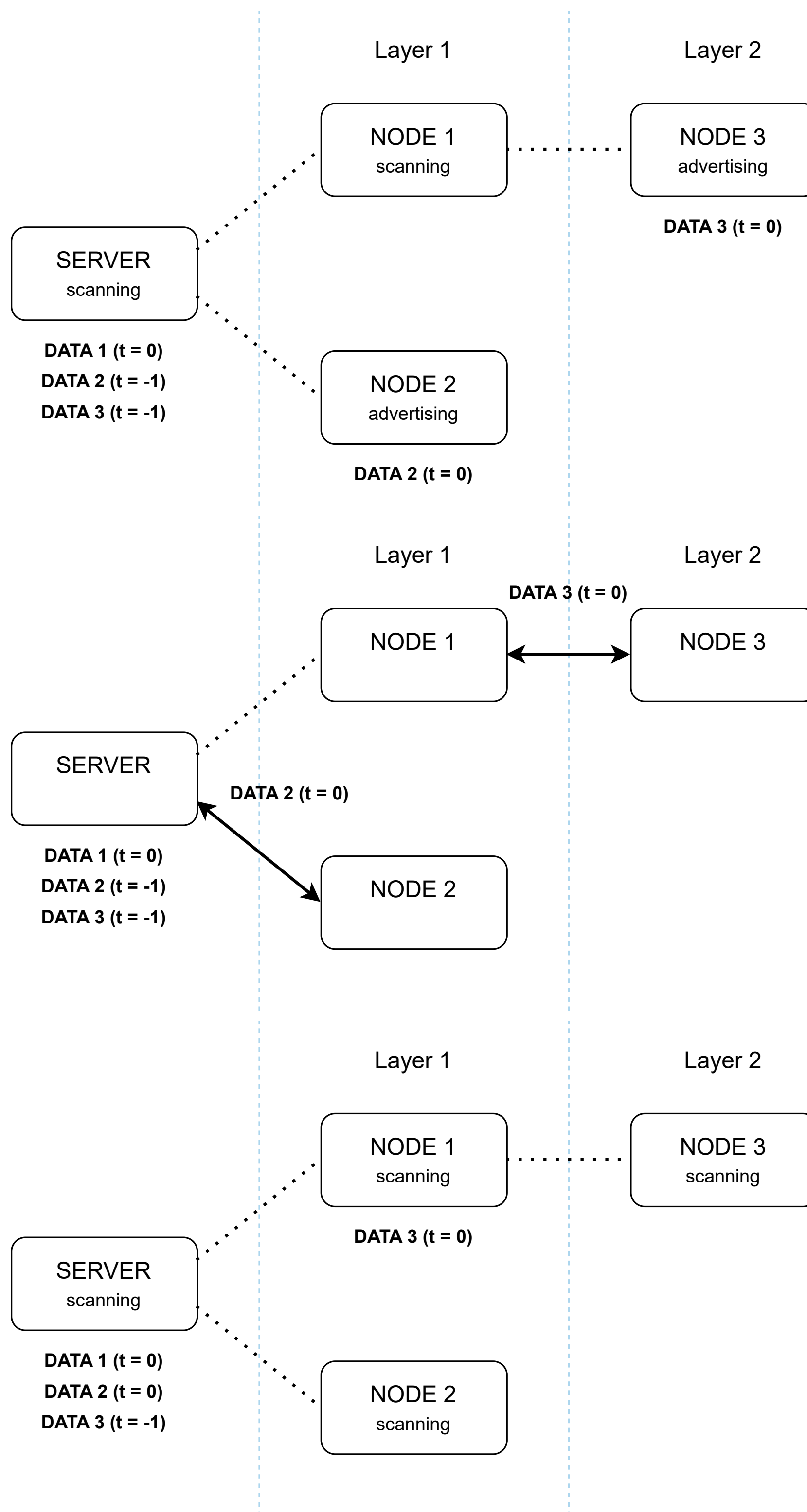


Figure 4.1: Example network communication diagram (continued)

One disadvantage of this approach is that measurement data only propagates towards the server by one layer per communication cycle – for example, if a node is 3 layers deep, the most up-to-date data the server has from the node is at least 2 communication cycles old. However, since the intended application is long term monitoring, it was assumed that this would not be an issue. Also, this only affects data transmission and not the configuration change messages, which propagate through the entire network in one cycle. This approach helps with decreasing communication time each cycle and greatly simplified the protocol.

4.2 Measurement data storage and transmission

Due to the architecture of the communication protocol and, generally, the mode of operation of the application, nodes need to be able to store measurement data before it can be sent to the server.

The data needs to contain information about its origin and when it was collected. Since the application could be adapted to any type of sensor and there could theoretically be different types of sensor nodes present in the same network, the type of measurement needs to be included with the data. A checksum should be calculated to guarantee that the data was not damaged in storage or during transmission.

I have decided to use the node's SoC's Bluetooth hardware MAC address as the node's origin ID, as it is reasonable to expect that there will not be two nodes with identical MAC addresses in any network. Since the data collection occurs periodically and there may be multiple measurement cycles in between communication cycles (and thus more than one measurement per file or message), timestamp of the batch measurement start as well as the measurement cycle period is stored and sent with the data.

In the prototype, data is stored in the same way in both the node and the server – on a portion of the main flash memory of the SoC. However, this would not be an optimal solution for a final product, since, in the selected board, an ESP32-C3 variant with on-package flash is used, which cannot be replaced when degraded from writes. In the final application, a separate flash memory should be used to store the measurements. Also, since the ESP32-C3 has 8 KiB of RTC SRAM, which can retain data even in deep sleep, it should be more than sufficient for most applications to use it for data storage in nodes in place of non-volatile flash memory.

To transfer the data from server to a readout device (a PC), the prototype server uses UART via the integrated USB-UART bridge. A Python script communicates with the server via specifically formatted plain text messages and plots the measured data.

4.3 Reliability

Microcontrollers have very limited resources when compared to modern desktop computers or servers. While these more powerful computer systems usually contain many different mechanisms for ensuring long-term stability and reliability, the options available to low-power embedded systems are very limited. Instead of extensive software checks or system redundancy, simpler and more energy efficient solutions are required.

While RAM capacity has traditionally been quite limited on microcontrollers, modern IoT SoCs usually have at least 256 KiB of RAM to facilitate communication stacks, which can be quite resource intensive. Still, with relatively lightweight protocols and stacks like Bluetooth LE and NimBLE, user applications are not nearly as memory limited as was the case in the past.

However, even though memory capacity may not be an issue, heap fragmentation, which occurs with frequent dynamic memory allocation and deallocation, can cause the system to run out of memory due to no continuous blocks of heap being left after a long enough run time. Thus, it is preferable to avoid dynamic allocation when possible.

One solution is to have a number of statically allocated memory blocks or buffers, which the application itself allocates to its components when needed. This is one of the reasons why higher level languages like Python, which is supported on the ESP32-C3 platform via the MicroPython or CircuitPython frameworks, are not optimal for long-term operating embedded applications like mine.

Another technique used to increase reliability are watchdog timers. These can be used to restore the function of the system in case of a freeze-up caused by a programming bug, unexpected inputs or other non-standard situations. A watchdog timer can reset the CPU if it itself is not reset in time, which would happen regularly in normal program operation.

Chapter 5

Firmware and software

5.1 Node and server firmware

As was already mentioned, the node and server share the same firmware, because they share a significant amount of functionality.

5.1.1 Server operation

After the server is initialized, it waits for time synchronization. This is done through the data readout application and is further described in section 5.3.

After synchronization is done, the application runs two tasks in parallel.

One is the server process, which scans for advertising nodes. When it finds one, it connects to it as described in section 4.1, sending it configuration and current time and reading measurement data off of it. This measurement data is then saved in a file stored on a SPIFFS partition on the system's main flash memory. The process runs indefinitely, with no timeouts and unlimited tries.

The second process periodically checks, whether there is any new data stored in the SPIFFS partition. If yes, it reads the data and sends it to the readout application over serial.

5.1.2 Node operation

When started, the node initializes the temperature and humidity sensor DHT11 connected to it. After that, it starts advertising for an unlimited time for the purpose of time synchronization. When a server or another node connects to it, it writes configuration and current time into the node.

After the time is synchronized and the node is configured, the application waits for the next cycle. There are two types of cycles – measurement cycles, when a measurement is taken by the node, and communication cycle, where after taking the measurement the network starts communicating to transmit the measured data. The period as well as the ratio of measurement and communication cycles can be configured. All nodes in the network are synchronized with a 1-second precision to ensure they all start communicating at the same time.

When a cycle starts, the application reads out measurement data from the connected sensor and writes it to a file on a SPIFFS partition on the system's main flash memory. The specifics of data storage are described in section 4.2.

If the current cycle is a communication cycle, the node then starts the node process, which advertises its presence, allowing server or other nodes to connect to the node, reading out its data. This is described in section 4.1. Then, either after the data was successfully read out or the connection times out, the node service stops and the server process is started. In this mode, the node acts similarly to the server, scanning for other nodes and trying to read out data from them. This way, the measurement data can propagate through the network from outermost layers to the server. The node only spends limited time in this mode, after which it waits for the next cycle.

5.1.3 ESP-IDF framework

ESP-IDF is the official framework for the Espressif ESP32 family of SoCs. It supports C and C++ programming languages and comes with several open-source libraries, some of which I use in my application. It has a flexible configuration system which can set most of the system parameters and include or exclude some optional libraries. Its build system is based on CMake [27].

ESP-IDF utilizes a modified version of FreeRTOS as its real-time operating system. It supports preemptive task scheduling and multiple ways of secure task-to-task communication and resource management, including queues, mutexes and semaphores.

The firmware for my application is written in C17 using ESP-IDF v5.2.1. It uses no third party library other than those included with ESP-IDF.

My code uses parts of example code provided by Apache and Espressif, which is licensed under Apache license [18] and is attributed in my application's source code. Also, examples by Espressif are used, which have been released into public domain under Creative Commons CC0.

5.2 Bluetooth LE stack and implementation

For my application, I have selected the Apache Mynewt NimBLE Bluetooth LE stack, which is included in ESP-IDF. As compared to the other included stack, Bluedroid, NimBLE is less resource intensive and, thanks to being a part of the multi-platform Apache Mynewt RTOS / framework, would allow the Bluetooth-related code to be relatively easily ported to another platform, if desired.

Bluetooth LE applications mainly utilize two Bluetooth profiles (sets of protocols) [7]:

- Generic Access Profile (GAP), which is used to broadcast information about the device for purposes of discovery (advertising) and establishing connection between devices, and
- Generic Attribute Profile (GATT), which are used to transmit information between two connected devices using a hierarchical structure of services, characteristics and descriptors.

5.2.1 Bluetooth-related terminology

There are two sets of roles defined by the two Bluetooth LE profiles.

GAP defines four roles: broadcaster, observer, central and peripheral. The broadcaster and observer roles will not be used in my application, since they do not support connections. Central is a device which actively scans for peripheral devices and can establish a connection with one or more of them. Peripheral devices advertise their presence when they are ready to be connected to.

GATT defines two roles: server and client. When a client connects to a server, it can access (read or write) its characteristics and descriptors.

Although generally independent of each other, the central GAP role is usually linked with the client GATT role and the peripheral GAP role with the server GATT role. However, in my application (as described in chapter 4.1), the device I call “server” uses the cental and client GAP and GATT roles respectively, while the “node” devices start with peripheral and server roles and later switch to the central and client roles.

To avoid confusion, from now on, I will be using the term “central” to describe both the GAP central role and the GATT client role, and the term “peripheral” for both the GAP peripheral role and the GATT server role.

5.2.2 Establishing connection (GAP)

In my application, peripherals broadcast advertising packets to signify they are available for connection – they have not sent data yet in this communication cycle. Centrals scan for the advertisement packets and connect to any advertising node they find.

The advertisement packet in my application contain the device name, TX power level and supported device features. The packets can also contain other data, including information required for pairing and custom data, which may for example communicate information about the device before establishing connection or may be used for device identification or verification.

In my application, to simplify the debugging process, nodes are discovered by matching the device name from the advertising packet. Further, no pairing is used, which would enable encryption. In the final application, out-of-bounds pairing should be used, which will enable any two devices in the network to pair and establish a secure connection thanks to a single security key which can be loaded into them when they are added to the network.

5.2.3 Data transfer (GATT)

GATT is based on services, which define a specific function of the device. There are several standardized services, for example the Blood Pressure Service or the Media Control Service. Each service has a 128-bit UUID, while the standardized services also have a shortened 16-bit UUID (which can also be purchased for use with a custom non-standardized service) [28].

While services by themselves do not contain any value, each service has at least one characteristic. Characteristics have a data value that can be read or written (depending on the application) and may also contain one or more descriptors. Characteristics are usually used to transfer data, while descriptors should contain additional information about the data – for example, a characteristic may contain a measured value with a descriptor containing the unit. Same as services, characteristics and descriptors also have a 128-bit UUID.

The communication protocol, whose basic architecture was described in section 4.1, uses GATT to send data between pairs of devices (node and server or two nodes), propagating it through the network. The GATT services, characteristics and descriptors implemented in my application are listed in table 5.1.

Attribute	UUID	Permission	Description
Node service	e4a4f6ce-5d66-4468-9e6e-58eac601aad2	—	Measurements
Data characteristic	4ab41637-0bdc-400d-b1f1-2f3f489e110b	R	Read measured data
		W	Request data at given index
Count descriptor	7de6febc-a0c8-404e-a4d2-6a53e0ec358b	R	Count of available measured data files
Confirm descriptor	dff2573f-bc93-4efe-a3bf-da770125529d	R	Perform read to confirm data was received correctly
Config service	bc3b6bd1-d1b0-4965-8500-06b50a5dde7e	—	Configuration and time synchronization
Time characteristic	ccaa1cec-8804-4c27-8cbd-bd6aa54e4010	R	Read current time
		W	Set current time
Values characteristic	5ea04c21-345f-4483-85c1-04883cb9e229	W	Update device configuration

Table 5.1: GATT services, characteristics and descriptors

There are two services representing the two types of data which are transmitted through the network: measurement data are transmitted through the Node service and the configuration changes and real time are transmitted using the Config service. The following description illustrated the way data is exchanged after a central (server or node which has already transmitted data) connects to a peripheral (node which has not yet transmitted data).

1. Central runs GATT service discovery on peripheral to find all supported services, characteristics and descriptors.
2. Central sets the maximum transmission unit (MTU) to 256 B (from 23 B default).
3. Central writes current Unix timestamp as a 64-bit signed integer to the Time characteristic of the Config service, peripheral confirms write.
4. Central writes its configuration, which contains settings for communication cycle synchronization and connection timeouts, to the Values characteristic of the Config service, peripheral confirms write.
5. Central reads peripheral's data count as a 16-bit unsigned integer from the Count descriptor of the Data characteristic of the Node service.

6. If there is no data left (everything was already read or peripheral has 0 data count), skip to step 10. Otherwise, central requests data by writing its index as a 16-bit unsigned integer to the Data characteristic of the Node service, peripheral confirms write.
7. Central then reads from the Data characteristic of the Node service to receive a set of measurements. As described in section 4.2, this contains information about measurement type and count, its origin (the MAC address of the node that measured the data), Unix timestamp of first measurement, measurement interval in seconds, and the measured data points. In this prototype, the data points are a pair of single-precision float values representing temperature (°C) and relative humidity (%).
8. To confirm that data was successfully received, central reads from the Confirm descriptor of the Data characteristic of the Node service. This causes the node to delete the data from its memory, marking is as sent.
9. Back to step 6.
10. If there is no data left to read, central disconnects from peripheral, sending the HCI disconnect code 0x13 – Remote user terminated connection.

If any error occurs during communication or an invalid operation is performed, including attempting to read data before setting the index, requesting non-existent or already deleted data or sending incorrect length data, the connection can be terminated by either the central or the peripheral. HCI disconnect code, which describes the nature of the error, is sent.

If transmission ends with an error, it is retried. Nodes only try to reconnect three times before skipping the current transmission cycle and the scanning operation has a set timeout. Server is configured to try for an unlimited period of time with unlimited attempts.

If the peripheral receives code 0x13, marking successful transmission, it stops advertising and switches to central mode.

5.3 Data readout application

For the purposes of reading data off of the server in the prototype application, I have created a Python 3 script which communicates with the server through serial interface over the ESP32-C3's integrated USB-UART bridge.

Because there are logs being sent over this serial line as well, the application uses specifically formatted text strings instead of sending raw binary data, which could potentially cause issues with serial monitor applications. Currently, there are two such data strings implemented:

- Current time string, starting with !@CTIME
- Measurement data string, starting with !@DATA

Upon startup, the script connects to the provided serial interface and sends a current time string. This synchronizes the server's RTC and allows it to propagate current time to the nodes.

A plot window with temperature and relative humidity subplots is created. When the script receives a measurement string, it adds the data to the plot, dynamically updating it. Example output plot can be seen in figure 5.1.

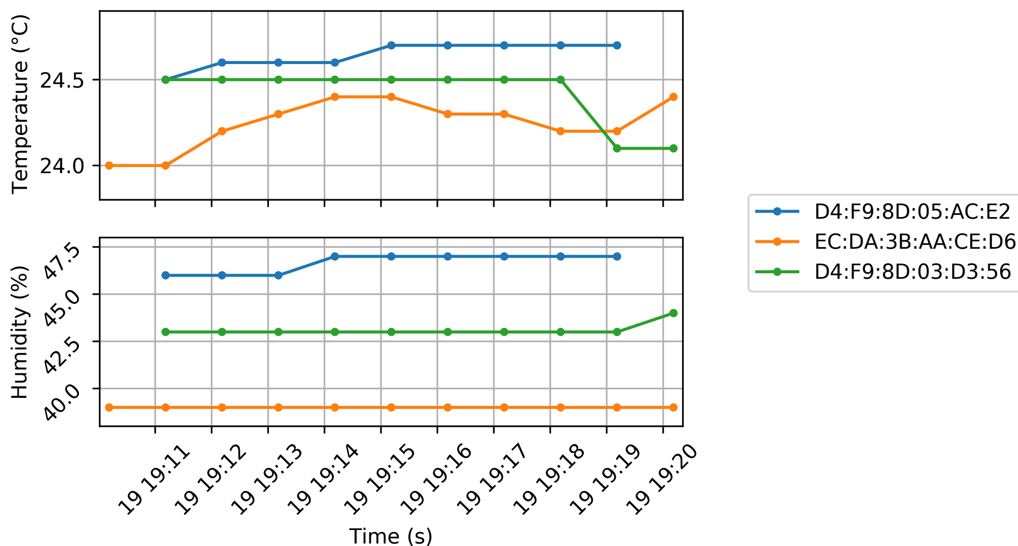


Figure 5.1: Example readout application output plot

Chapter 6

Experiments and results

After implementing the firmware for the server and nodes, I have performed a series of tests to evaluate the network's reliability, power consumption and communication range. Unless otherwise stated, the network was configured with the following parameters:

- 60 s measurement cycle period, 1 communication cycle every 4 cycles;
- 15 s advertising timeout of node in peripheral role, up to 3 connection retries;
- 10 s scanning timeout of node in central role;
- +9 dBm TX power level;
- 160 MHz CPU clock and 80 MHz flash clock.

The testing was done on a server and up to three nodes. The network was run in various conditions and configurations to find possible issues and ensure stable long-term operations.

After debugging the firmware, I have performed a series of stability tests, the longest of which has run for over 10 hours. In all of these tests, the network performed as expected with no dropped or lost data and with no unhandled errors affecting the operation of the devices.

6.1 Current consumption testing

In this section I will go through a series of tests, showing and explaining the measured current consumption plots. At the end of this section, I will give approximate current charge (energy) consumption figures for the common operations.

I measured the current consumption of the devices with a FNIRSI FNB-48 USB power meter. I collected the data using an unofficial data logger program [30], which logs the measurements with a period of 10 ms.

It is important to note that the measured current consumption is of the entire system – including the DHT11 sensor connected to nodes.

To measure the node's current consumption during communication I have changed the network's parameters so that communication occurs on every 60 s cycle. In figure 6.1, the current consumption of a node is plotted with labels marking different parts of the node's operation.

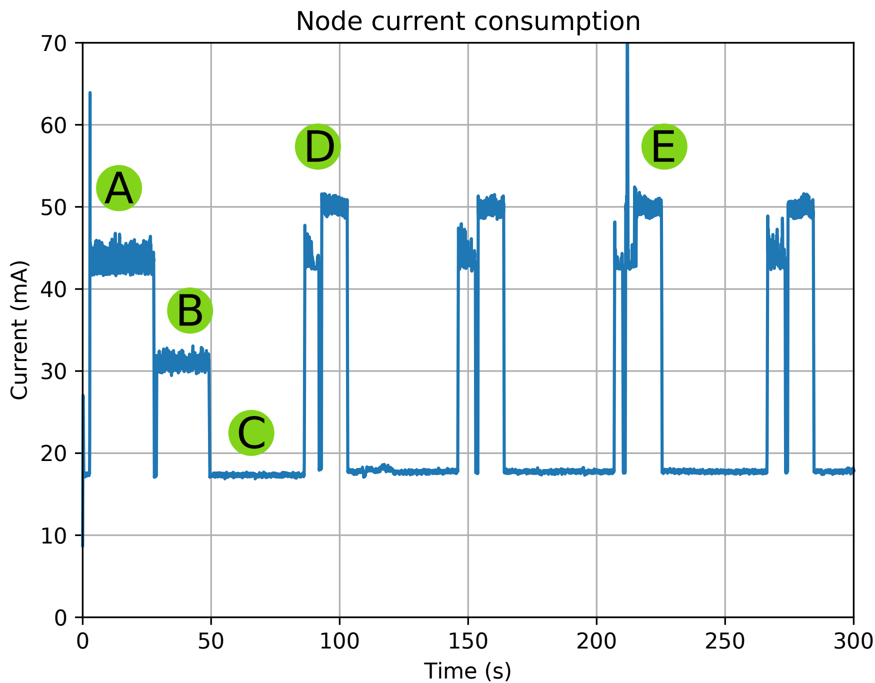


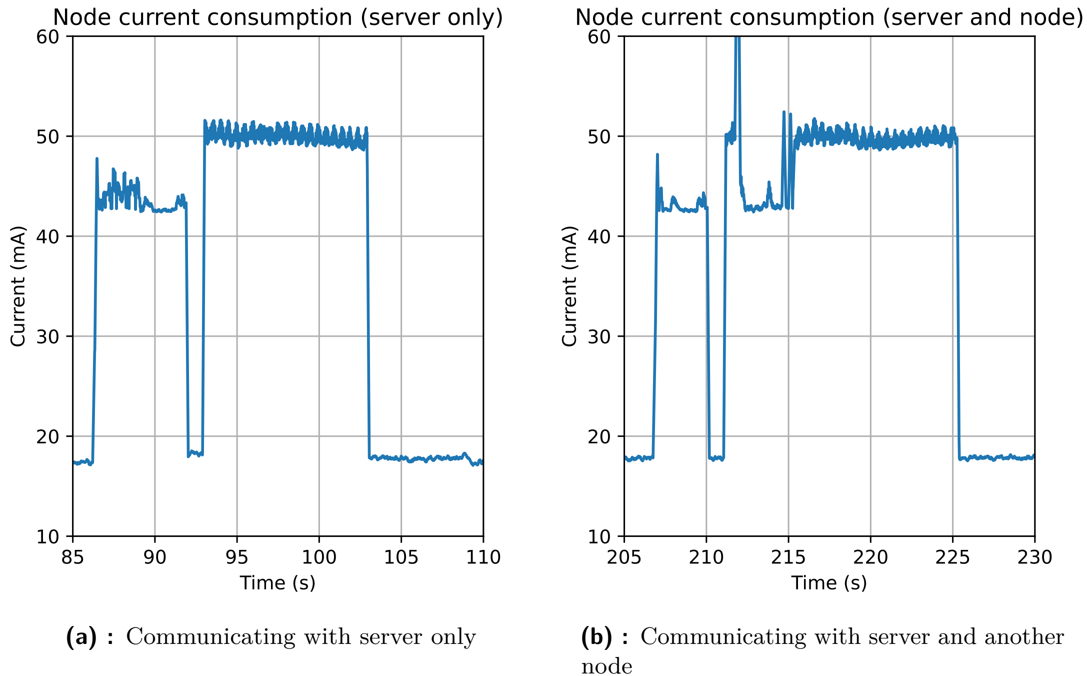
Figure 6.1: Current consumption of a node

After start-up, label Ⓐ marks the node advertising and waiting for time synchronization. This consumes on average 44.1 mA.

Because the entire flash memory of the node was erased before it was programmed for this test run, it was necessary to initialize the SPIFFS file system where measurement data is stored. This operation is marked Ⓑ, the node consumes on average 31.2 mA over 20.4 s for a total charge consumption of 18 μAh.

After the initialization is complete, the node waits for the upcoming cycle, the waiting period is labeled Ⓒ. Because I have not implemented sleep in the prototype, the CPU stays active while being idle, consuming on average 17.9 mA.

In this example, communication occurs every cycle. Labels (D) and (E) mark the wireless transmission – in the first case the node just communicates with the server, while in the second case the node also connects with another node in a lower network layer. These two examples are shown in more detail in figure 6.2.



(a) : Communicating with server only

(b) : Communicating with server and another node

Figure 6.2: Current consumption of a node – communication cycle detail

The communication is visibly divided into two parts. First, the node is in peripheral mode and advertises for up to 15 s. The actual length of advertising and subsequent communication depends on the order in which the nodes are being discovered by the server or another node – in my testing I saw the time measurements to be grouped around 3.7 s and 6.8 s, which would correspond with the node being first and second, respectively, to be connected to by the central. The average current during the read operation was 43.2 mA. This resulted in approxite charge consumption of 44 μAh or 81 μAh depending on the order of connection.

In the second part of the communication cycle, the node switches into central mode and proceeds to scan for 10 s. The average current was 49.6 mA, which results in average charge consumption of 138 μAh per communication cycle.

When an advertising node in peripheral mode is found, the node in central mode connects to it and reads out its measurement data. This can be identified on the current consumption plot by a peak greater than 60 mA followed by decreased current consumption for around 3 seconds. I measured average consumed charge of 43 μAh, which is very close to the consumption of a node in peripheral mode.

While the node is in central mode, it can sequentially connect to multiple peripheral nodes, because the scanning restarts after every successful communication. Since the connection can be established anytime during the 10 s of scanning and will not happen equally to every node in the system, the actual energy consumption depends heavily on the spatial configuration of the network.

I have done another test with the default configuration to measure current consumption during measurement cycles, the current plot is in figure 6.3.

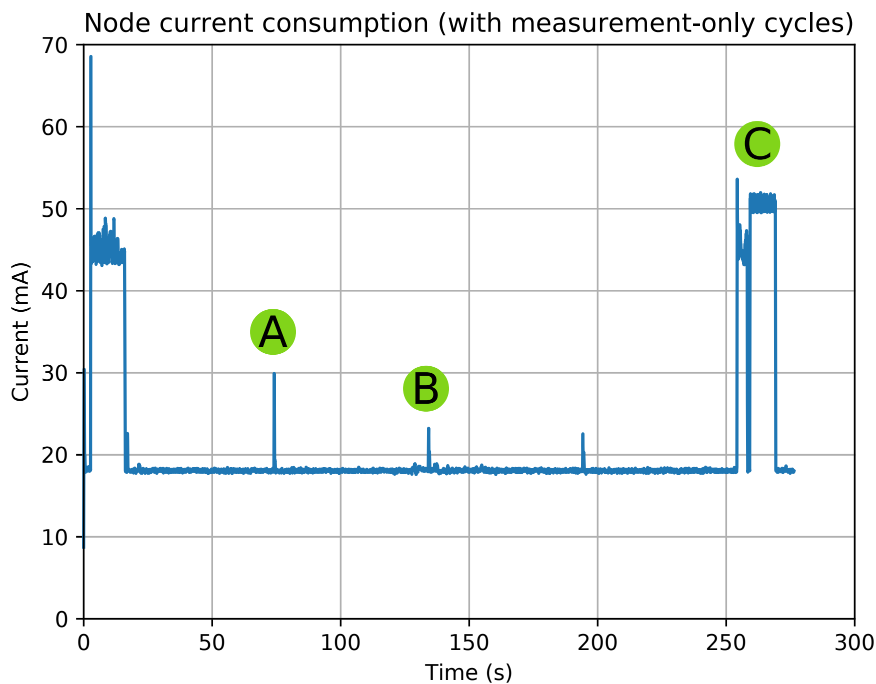


Figure 6.3: Current consumption of a node – measurement cycles enabled

As expected, the measurement cycles labeled (A) and (B) consume far less energy than the communication cycle labeled (C). The first measurement cycle requires slightly more power, because a new file is created in the SPIFFS file system on the flash memory. The measurement cycles last 0.6 s with an average electric charge consumption of 3.4 μAh, the new file creation (which happens after every communication cycle) consuming extra 0.2 μAh.

A notable difference between figures 6.1 and 6.3 is the missing initialization of SPIFFS (marked (B) in 6.1), because the SoC's flash memory was not cleared before the second test run.

I have also recorded power consumption of a server, plot of which is in figure 6.4. For this test, I have again changed the network's parameters so that communication occurs on every 60 s cycle and I used two nodes for the testing.

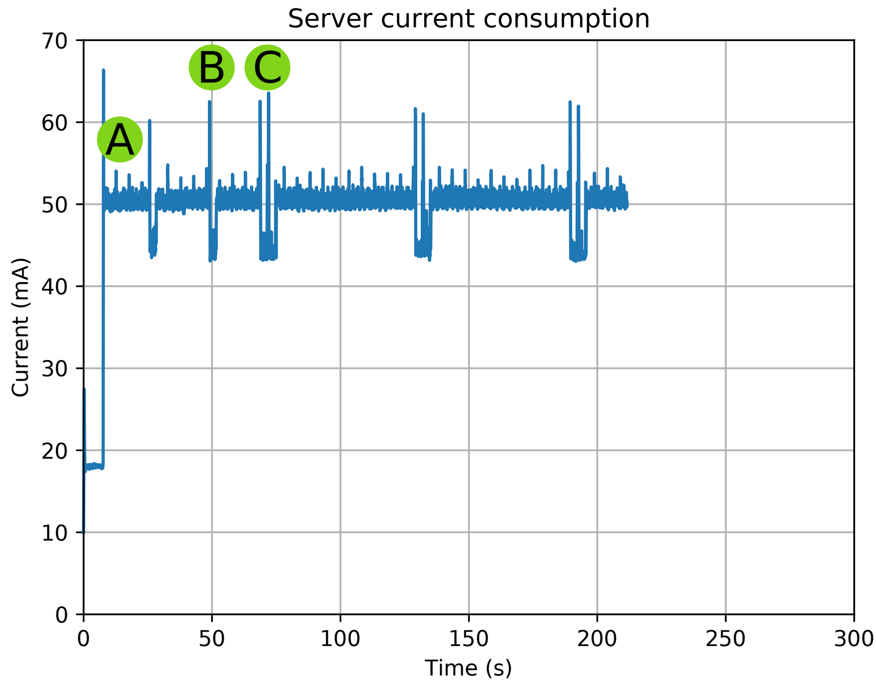


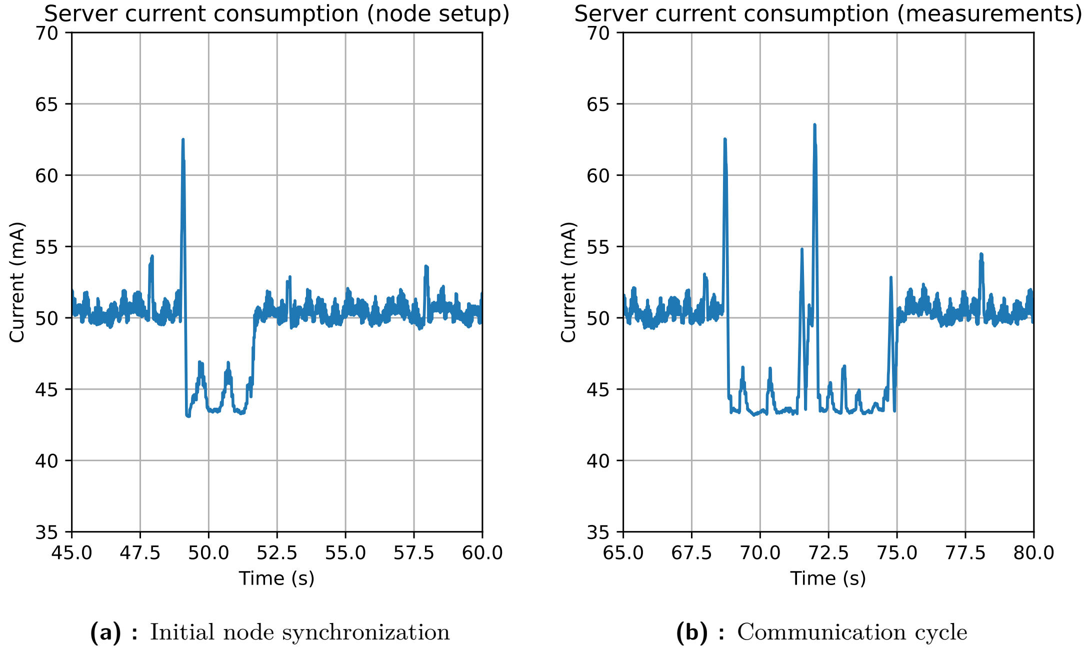
Figure 6.4: Current consumption of a server

After the server synchronizes its RTC over serial, it starts scanning for nearby nodes, which is labeled Ⓐ in the plot. The average current consumption of the server was 50.4 mA, very close to a scanning node in central mode.

There are two types of connection events visible in the current consumption plot. The short synchronization when node first connects to a network is labeled Ⓑ, while Ⓒ marks the synchronized communication cycles when measurement data is sent to the server. In figure 6.5, these two types of communication are shown in more detail.

Both of the plots contain the specific current consumption curve that could also be seen with node in central mode in figure 6.2b. A notable difference is that the communication during synchronization is slightly shorter due to no measurement data being sent – initial setup takes on average 2.6 s, while communication cycle takes 3.1 s per connected node. The average consumed electric charge is 33 μAh for initial setup and 39 μAh per node for communication cycle.

Compared to the node, the server's overall current consumption was not nearly as optimized in the prototype, with the server scanning for nodes the entire time it is running. Because scanning is by far the most energy intensive operation, in the final application, a better strategy would need to be implemented.



(a) : Initial node synchronization

(b) : Communication cycle

Figure 6.5: Current consumption of a server – communication detail

As a final test, I measured the development board's current consumption when the SoC is in deep sleep mode. I used a simple test program that periodically goes to sleep and wakes up after a few seconds using RTC.

When powering the board through USB, I have measured an average current consumption of 205 μA in deep sleep, which is much higher than the datasheet figure of 5 μA for the SoC by itself. I suspect most of the difference is consumed by the charging status LED on the development board, which is lit at a low intensity (visibly flickering) when the board is powered via USB with no battery attached.

To confirm this theory, I soldered a Li-Ion battery to the battery pads on one of the boards. To measure the current, I used a Pro'sKit MT-1710 multimeter in its μA range, which has a datasheet resolution of 0.1 μA and accuracy of $\pm$1.0 % of reading $\pm$1 μA [31]. With this setup, I measured 38.4 μA current consumption during sleep.

In table 6.1, I have summarized the current or electric charge consumption of various operations that I measured and calculated.

In repeated measurement, the tolerance of the results was within 10 %.

Operation	Current	Electric charge consumption	Note
CPU idle	17.9 mA	—	—
Deep sleep	38.4 μA	—	—
Node – measurement	—	3.4 μAh / cycle	—
Node – communication in peripheral mode	43.2 mA	44 μAh / cycle	when connected to first by the server
		81 μAh / cycle	when connected to second by the server
Node – scanning in central mode	49.6 mA	$\geq$ 138 μAh / cycle	at least 10 s of scanning per communication cycle
Node – communication in central mode	—	43 μAh / connected device	connection to nodes on a lower layer
Server – scanning	50.4 mA	—	—
Server – communication	—	33 μAh / connected device	initial node synchronization
		39 μAh / connected device	communication cycle

Table 6.1: Average current and electric charge consumption of nodes and server with default parameters

6.2 Communication range testing

To measure communication range, I used the signal strength reported by the server when it discovers a node. Because this network is intended mainly for indoor operation, I measured signal strength drop in a long corridor and across multiple floors to simulate signal strength loss from both distance and obstacles. The measurements were done in a residential apartment building constructed from reinforced concrete panels.

For these tests, I shortened the communication cycle period to 15 s with no measurement cycles in between, the advertising timeout was shortened to 10 s and the scanning timeout was set to 1 s. The TX power level was kept at +9 dBm. Every measurement was repeated 3 times in separate test runs.

For distance testing without obstacles, a node was placed on one end of a corridor and the server was moved in 5 m increments down the corridor. In figure 6.6, the signal strength drop is plotted.

To test signal loss due to obstacles, the node was placed in a residential floor access corridor and measurements were taken on the floors above in the same spot. The floor to ceiling height of the corridor is 2.5 m and the floor is 270 mm thick. In figure 6.7, the signal strength drop is plotted. The test run was ended if more than 3 in the first 5 measurements on the particular floor ended with a communication error.

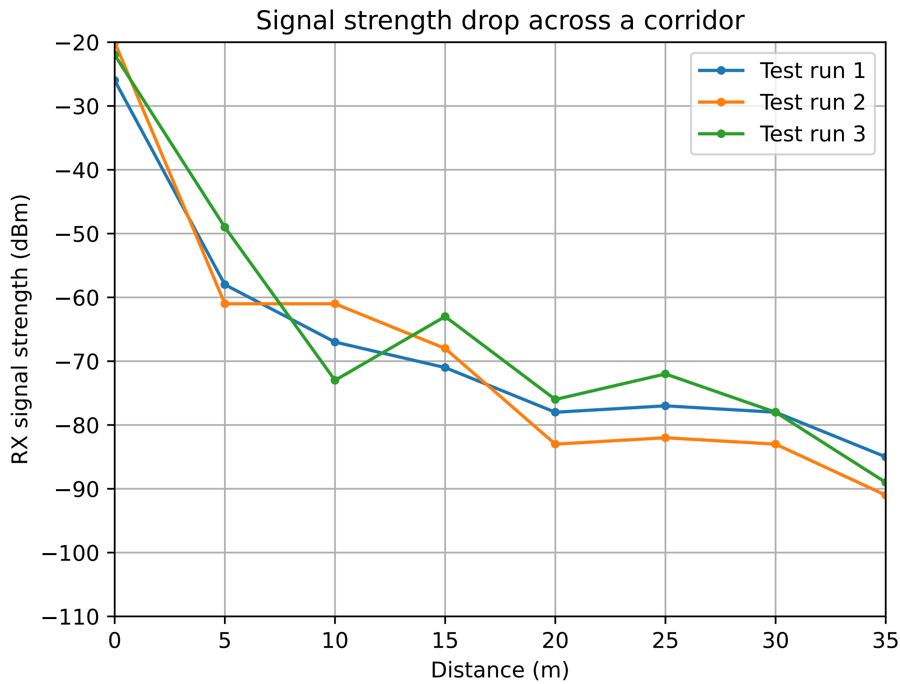


Figure 6.6: Signal strength drop across a corridor

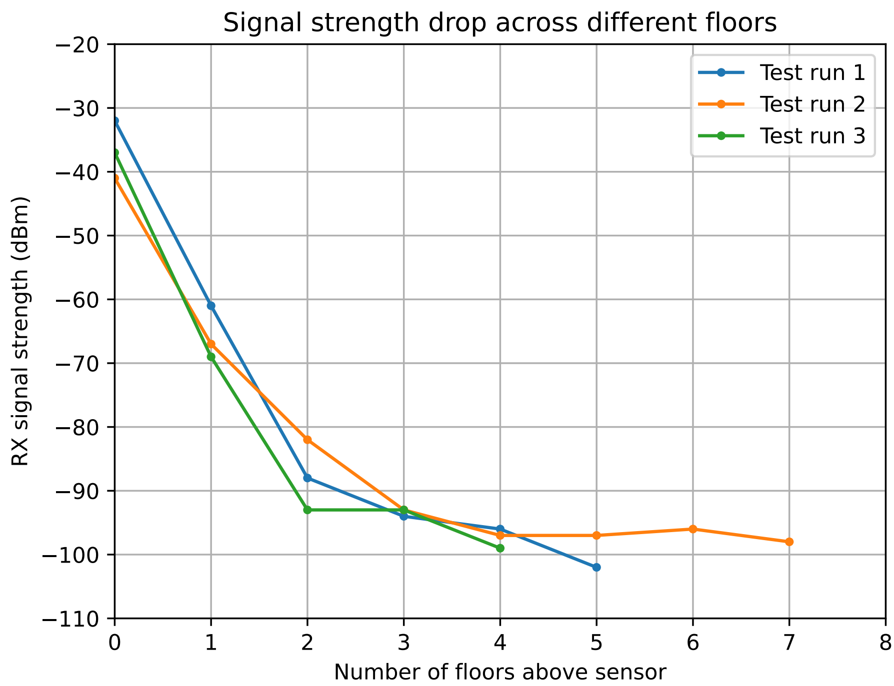


Figure 6.7: Signal strength drop across different floors

As can be seen from the graph, the signal can pass through multiple reinforced concrete walls or tens of meters of free space without any issues. This confirms that Bluetooth LE has suitable range for the application.

6.3 Testing at different TX power levels

I have repeated some of these tests with different TX power levels to see their impact on the communication range and energy efficiency of the nodes.

I have decided to test at 0 dBm because, from the range testing, it was clear that +9 dBm offered higher range than would be required by many applications. At 0 dBm, I would also be able to put the measured current consumption in context with the datasheet values in table 6.1.

The ESP32-C3 SoC supports up to +21 dBm TX power for Bluetooth LE [17]. However, in the Czech Republic, the maximum radiated power at the 2.4 GHz frequency band is legally limited to 100 mV e.i.r.p., or +20 dBm [32]. Because the ESP32-C3 has TX power configurable in steps of 3 dBm, I selected +18 dBm as the value to be tested.

The power testing for 0 dBm and +18 dBm was done with a server and a single node. Current consumption during scanning was identical with +9 dBm due to scanning not depending on transmission. In table 6.2, the average electric charge consumption of operations which transmit data is compared.

Operation	Average electric charge consumption at power level		
	0 dBm	**+9 dBm**	**+18 dBm**
Node – communication in peripheral mode	43 μAh / cycle	44 μAh / cycle	47 μAh / cycle
Server – initial node synchronization	32 μAh / device	33 μAh / device	34 μAh / device
Server – communication during cycle	39 μAh / device	39 μAh / device	40 μAh / device

Table 6.2: Comparison of average electric charge consumption at different TX power levels

As can be seen, the difference caused by transmission power level is negligible except for node communicating in peripheral mode due to the measurement data that is being transmitted by the node. If a larger amount of data was to collect in the node's memory, the difference would be even greater. It can also be seen that the power difference between 0 dBm and +9 dBm is smaller, than between +9 dBm and +18 dBm – this is expected, because the power level is measured on a logarithmic scale.

I tested the communication range of all three power levels in the corridor. Identical configuration was used and for all power levels 3 separate runs were performed. Average RX signal strength is compared in a plot in figure 6.8.

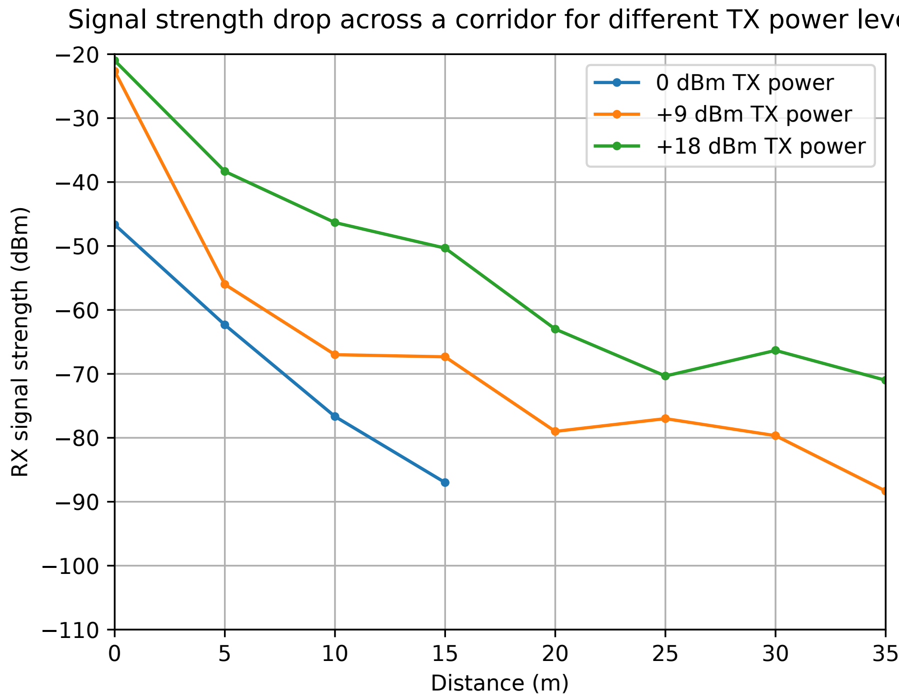


Figure 6.8: Signal strength drop across a corridor for different TX power levels

In this test, with 0 dBm TX power the communication started dropping out at 15 m node to server distance, at notably higher signal strength compared to the drop-outs in the test of signal loss across different floors.

The difference in received signal power between +9 dBm and +18 dBm TX power levels was notably higher than 9 dBm, namely 15 dBm on average. This could be due to imperfect measurement, imprecise TX power regulation in the SoC, as well as due to the unpredictable way the RF signal might have propagated through the corridor.

With the data I measured using the prototype, I could refine the daily energy consumption estimate I made in section 2.2.3. The following parameters were kept the same between the two estimations:

- measurement cycle period of 15 minutes,
- communication cycle period of 3 hours, and
- deep sleep to save power.

While deep sleep was not implemented in the prototype, it was tested separately and would certainly be utilized in the final application. I have added 3 s of idle time to each measurement operation to compensate for the wake-up time.

I decided to use +9 dBm TX power – its slight increase in energy consumption over 0 dBm is outweighed by its significantly longer range. I have decided to set node peripheral timeout to 30 s and node central timeout to 10 s.

I calculated the power consumption for the following scenario: a node in first layer, which contains 9 nodes in total – on average, it is connected to after 4 previous nodes (the energy consumption of this will be interpolated from the measured values). The node then, on average, connects to 2 other nodes in second layer (the number could change depending on whether another nearby node in the range of the lower-layer nodes finishes its communication sooner).

In table 6.3 the estimated daily electric charge consumption of the described network based on my design is calculated.

Operation	**Daily count**	**Daily time required (s)**	**Daily electric charge consumption (mAh)**
Measurement	96	384	1.8
Communication in peripheral mode after on avg. 4 nodes	8	152	1.8
Communication in central mode with on avg. 2 nodes	8	49	0.6
Scanning until timeout	8	80	1.1
Deep sleep	—	85735	0.9
Total	—	86400	**6.2**

Table 6.3: Estimated daily electric charge consumption of a WSN node

In section 2.2.3 and table 2.2, I have estimated the daily current consumption of the SoC by itself to be 12.38 mAh – almost exactly double the currently estimated value. This was caused in large part due to largely overestimating the current consumption of the radio. The datasheet values are specified for a 100 % duty cycle, but, in my application, the radio is utilized at a much lower percentage due to the small size of most exchanged messages.

Even though the value in table 6.3 might be underestimated, there are still areas which could be optimized, and so, in my opinion, a daily energy consumption on this order of magnitude could realistically be achieved using a solution based on my prototype.

Chapter 7

Discussion, conclusion and future work

In this thesis, I proposed a cost-efficient wireless sensor network for long-term data acquisition. I designed a prototype device, on the basis of which a fully featured WSN could be developed. I then tested the prototype's communication range and power consumption to evaluate the design's suitability for its intended application.

By researching currently available wireless technologies and weighing their strengths and weaknesses in the context of an ultra-low power WSN, I have selected two wireless communication protocols which seemed to be the best suited for the application's requirements – Bluetooth LE and Zigbee.

I have decided to base the nodes on the Espressif ESP32-C3 wireless SoC because of its exceptional RF performance, well-documented SDK and very low cost of both the SoC and its development boards. Compared to competing solutions from other manufacturers, including Texas Instruments SimpleLink series and Nordic Semiconductor nRF52 series, the considered parameters of the Espressif ESP32-C3 were on-par or better, except for its notably higher current consumption. Calculations of estimated daily energy consumption however showed that the difference, while significant, would not make the ESP32-C3 unusable for the application and, in my opinion, its other advantages outweighed this flaw.

Since the selected SoC does not support Zigbee, I based the network's communication protocol on Bluetooth LE. In the design of the protocol, I focused on simplicity, reliability and extensibility. The network consists of a server and a number of nodes which operate in synchronized cycles, collecting measurements and then connecting together to send the measurement data to the server. The network has a tree topology with the server as its root and the nodes separated into layers by their physical distance from the root. The nodes located closer to the server relay data from more distant nodes, which may be outside the server's communication range. To simplify the design and lower energy consumption, the measurement data only propagates through the network by one layer towards the server per communication cycle. This was considered to be an acceptable trade-off for the intended application of long-term monitoring, where the increased latency is not likely to pose an issue.

The firmware was implemented in the C programming language, using the ESP-IDF development framework, FreeRTOS real-time operating system and Apache MyNewt NimBLE Bluetooth LE stack. To read out data from the network, a Python 3 script was created, which communicates with the server via its integrated USB-UART bridge.

In testing, the network proved to be reliable and was able to achieve better energy efficiency than was at first estimated. Due to a relatively low data bandwidth utilization, the RF current consumption was significantly lower than the datasheet values, which are given for 100 % network utilization. After measuring the energy consumption of all common operations, I was able to update my initial energy consumption estimate with real-world values, proving that a final node based on this design could have a run-time of more than 200 days on a single charge of a common 3000 mAh 18650 Li-Ion battery.

Thanks to the simple and highly configurable design of the WSN, large networks consisting of more than 100 nodes can be constructed. The tree topology of the network helps to keep the communication cycles shorter by aggregating data in nodes closer to the server and allowing multiple parallel connections between nodes further from it.

Bluetooth LE proved in my testing to have sufficient range for more spread-out networks. I have tested at three different TX power levels, of which the medium value of +9 dBm proved to have a very good range of more than 35 m and to be able to pass through multiple reinforced concrete walls, while consuming only marginally more power compared to the lower TX power level of 0 dBm.

In the prototype, not all features of the final WSN were implemented. The omitted features, which would need to be implemented in the final product, include encryption, wireless communication with readout device and on the fly configuration of the network. The selected solution however proved itself to be a viable basis for development of a fully featured product. The network protocol and data storage format are ready to be extended to support multiple types of sensors in addition to the currently supported temperature and humidity sensor DHT11.

The ESP32-C3 platform is very well suited for sensor applications thanks to supporting many digital interfaces including SPI, I2C and CAN-compatible automotive interface [17]. The size of the flash image of the current firmware is 654 KiB, or 65.4 % of the currently assigned 1 MiB flash program partition, which allows for many more sensor drivers and other features to be implemented. If necessary, the program partition can be expanded to use a larger portion of the 4 MiB system flash memory, meaning that memory limitation is unlikely on this platform.

The SoC proved to be sufficiently power efficient, however, the prototype does not implement most of the power saving features which I suggested in the thesis. Some were tested on their own to prove they could be used to achieve the energy efficiency required for the intended application. Of these features, the most important to implement is deep sleep when the node is idle while waiting for a next cycle. This also requires special hardware design considerations to minimize any energy loss in the power delivery both

during and outside deep sleep periods, which could significantly decrease the battery runtime.

Throughout this thesis, I have suggested a number of features which could be implemented in the final product to make it a very capable and competitive solution. The extensible design of the application has a great potential to find many uses if it is implemented into a cost-efficient and user-friendly device.

Appendix A

Bibliography

1. MOSTEFA, Benfilali; ABDELKADER, Gafour. A survey of wireless sensor network security in the context of Internet of Things. In: *2017 4th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)* [online]. 2017, pp. 1–8 [visited on 2024-04-15]. Available from DOI: 10.1109/ICT-DM.2017.8275691.
2. KHALIFEH, Ala'; MAZUNGA, Felix; NECHIBVUTE, Action; NYAMBO, Benny Munyaradzi. Microcontroller Unit-Based Wireless Sensor Network Nodes: A Review. *Sensors* [online]. 2022, vol. 22, no. 22, article no. 8937 [visited on 2023-07-12]. ISSN 1424-8220. Available from DOI: 10.3390/s22228937.
3. NIKOUKAR, Ali; RAZA, Saleem; POOLE, Angelina; GÜNEŞ, Mesut; DEZFOULI, Behnam. Low-Power Wireless for the Internet of Things: Standards and Applications. *IEEE Access* [online]. 2018, vol. 6, pp. 67893–67926 [visited on 2023-07-11]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2018.2879189.
4. CHEN, Wu; LIU, Jiajia; GUO, Hongzhi; KATO, Nei. Toward Robust and Intelligent Drone Swarm: Challenges and Future Directions. *IEEE Network* [online]. 2020, vol. 34, no. 4, pp. 278–283 [visited on 2023-06-22]. ISSN 1558-156X. Available from DOI: 10.1109/MNET.001.1900521.
5. AKPAKWU, Godfrey Anuga; SILVA, Bruno J.; HANCKE, Gerhard P.; ABU-MAHFOUZ, Adnan M. A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges. *IEEE Access* [online]. 2018, vol. 6, pp. 3619–3647 [visited on 2024-04-15]. Available from DOI: 10.1109/ACCESS.2017.2779844.
6. MOUSAVI, Seyed Mehdi; KHADEMZADEH, Ahmad; RAHMANI, Amir Masoud. The role of low-power wide-area network technologies in Internet of Things: A systematic and comprehensive review. *International Journal of Communication Systems* [online]. 2022, vol. 35, no. 3, e5036 [visited on 2023-06-22]. Available from DOI: 10.1002/dac.5036.
7. *Bluetooth Core Specification* [online]. Bluetooth SIG, 2016. Version 5.0 [visited on 2023-07-15]. Available from: https://www.bluetooth.com/specifications/specs/core-specification-5-0/.

8. *Bluetooth Core Specification Addendum 5* [online]. Bluetooth SIG, 2015. [visited on 2023-07-20]. Available from: `https://www.bluetooth.com/specifications/specs/csa-core-specification-addendum-5/`.
9. *Mesh Profile* [online]. Bluetooth SIG, 2019. Version 1.0.1 [visited on 2023-07-17]. Available from: `https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/`.
10. *IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* [online]. 2020. [visited on 2023-07-15]. Standard. The Institute of Electrical and Electronics Engineers. ISBN 978-1-5044-7284-5. Available from DOI: `10.1109/IEEESTD.2021.9363693`.
11. *IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN* [online]. 2021. [visited on 2024-04-21]. Standard. The Institute of Electrical and Electronics Engineers. ISBN 978-1-5044-7389-7. Available from DOI: `10.1109/IEEESTD.2021.9442429`.
12. *ESP32-C6 Series Datasheet* [online]. Espressif Systems, 2023. Version 1.0 [visited on 2024-04-22]. Available from: `https://www.espressif.com/sites/default/files/documentation/esp32-c6_datasheet_en.pdf`.
13. *IEEE Standard for Low-Rate Wireless Networks* [online]. 2020. [visited on 2023-07-15]. Standard. The Institute of Electrical and Electronics Engineers. ISBN 978-1-5044-6689-9. Available from DOI: `10.1109/IEEESTD.2020.9144691`.
14. *CC1312R SimpleLink High-Performance Sub-1 GHz Wireless MCU datasheet* [online]. Texas Instruments Incorporated, 2020. [visited on 2024-04-28]. Available from: `https://www.ti.com/lit/ds/symlink/cc1312r.pdf`.
15. *TexasInstruments/simplelink-lowpower-f2-sdk* [online]. Texas Instruments Incorporated, 2024. Version lpf2-7.40.00.77 [visited on 2024-04-27]. GitHub repository. Available from: `https://github.com/TexasInstruments/simplelink-lowpower-f2-sdk/tree/lpf2-7.40.00.77`.
16. *ESP8684 Series Datasheet* [online]. Espressif Systems, 2024. Version 1.5 [visited on 2024-04-28]. Available from: `https://www.espressif.com/sites/default/files/documentation/esp8684_datasheet_en.pdf`.
17. *ESP32-C3 Series Datasheet* [online]. Espressif Systems, 2023. Version 1.4 [visited on 2023-07-15]. Available from: `https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf`.
18. *espressif/esp-idf* [online]. Espressif Systems, 2024. Version 5.2 [visited on 2024-04-28]. GitHub repository. Available from: `https://github.com/espressif/esp-idf/tree/v5.2`.

19. *nRF52840 ProductSpecification* [online]. Nordic Semiconductor ASA, 2023. Version 1.8 [visited on 2020-04-28]. Available from: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.8.pdf.
20. *nrfconnect/sdk-nrf* [online]. Nordic Semiconductor ASA, 2024. Version 2.6.1 [visited on 2024-04-28]. GitHub repository. Available from: https://github.com/nrfconnect/sdk-nrf/tree/v2.6.1.
21. *NRF Connect SDK v2.6.1 documentation* [online]. Nordic Semiconductor ASA. [visited on 2024-04-28]. Available from: https://docs.nordicsemi.com/bundle/ncs-2.6.1/page/nrf/index.html.
22. *CC1310 SimpleLink Multiprotocol 2.4 GHz Wireless MCU datasheet* [online]. Texas Instruments Incorporated, 2023. [visited on 2024-04-28]. Available from: https://www.ti.com/lit/ds/symlink/cc2652r.pdf.
23. *XIAO ESP32-C3 schematic* [online]. Seeed Studio, Inc, 2022. [visited on 2024-04-29]. Available from: https://files.seeedstudio.com/wiki/XIAO_WiFi/Resources/Seeeduino-XIAO-ESP32C3-SCH.pdf.
24. *ESP32-C3 Technical Reference Manual* [online]. Espressif Systems, 2024. Version 1.1 [visited on 2024-05-14]. Available from: https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf.
25. *Seeed Studio XIAO INTRODUCION* [online]. Seeed Studio, Inc. [visited on 2024-04-29]. Available from: https://wiki.seeedstudio.com/SeeedStudio_XIAO_Series_Introduction/.
26. *Getting Started with Seeed Studio XIAO ESP32C3* [online]. Seeed Studio, Inc. [visited on 2024-04-29]. Available from: https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/.
27. *ESP-IDF Programming Guide* [online]. Espressif Systems. [visited on 2024-04-29]. Available from: https://docs.espressif.com/projects/esp-idf/en/v5.2.1/esp32c3/index.html.
28. *Assigned Numbers* [online]. Bluetooth SIG, [n.d.]. 2024-05-03 [visited on 2024-05-05]. Available from: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf?v=1714945359311.
29. *FNB48 User Manual* [online]. Shen Zhen Shi Fei Ni Rui Si Technology Co., Ltd. (FNIRSI). Version 0.6 [visited on 2024-05-20]. Available from: https://img.wqdres.com/res/0/20231229/f87ff89d49524b1180cc0c52a99fc2a2.pdf.
30. BARYLUK, Witold. *baryluk/fnirsi-usb-power-data-logger* [online]. 2023. [visited on 2024-05-21]. GitHub repository. Available from: https://github.com/baryluk/fnirsi-usb-power-data-logger.
31. *Pro'sKit(R) MT-1710 3-3/4 True-RMS Auto Range Multimeter User's Manual* [online]. Prokit's Industries Co., Ltd., 2013. 1st Edition [visited on 2024-05-22]. Available from: https://www.manualslib.com/manual/738267/Proskit-Mt-1710.html.

32. *General Authorisation No. VO-R/12/11.2021-11 for the use of radio frequencies and for the operation of equipment for broadband data transmission in the bands 2.4 GHz–71 GHz.* [online]. Czech Telecommunication Office, 2021. [visited on 2024-05-23]. Available from: `https://ctu.gov.cz/en/sites/default/files/obsah/stranky/74784/soubory/vo-r-12-11.2021-11enfin.pdf`.