UCLouvain

École polytechnique de Louvain

IoT based greenhouse monitoring

Author: **Pierre-Yves HURD**
Supervisors: **Yves DEVILLE, Ramin SADRE**
Readers: **Aurelien BUCHET, Xavier DRAYE**
Academic year 2022–2023
Master [120] in Computer Science

Acknowledgments

I would like to express my gratitude to my promotors, Prof. Yves Deville and Prof. Ramin Sadre, for their continuous support and guidance throughout my Master's research. Their expertise, insights, and patience have been indispensable in shaping this work.

Special appreciation is also extended to Xavier Draye, a Bioengineering teacher, and Marc Migon, the greenhouse manager. Their interest and collaboration in the proposed IoT greenhouse project were crucial to its successful realization.

I owe a great debt of gratitude to my parents for their unwavering support and encouragement throughout my studies. Their love and support have been instrumental in all my achievements.

Special thanks are due to the members of the WELCOME platform. Their willingness to provide equipment for the experiments, and their assistance in helping me utilize them effectively, significantly contributed to the successful completion of this project.

Furthermore, I appreciate the assistance of OpenAI's language model, ChatGPT, which was instrumental in the correction and revision of the text of this thesis. It has served as a valuable tool in refining the written content.

Lastly, I wish to express my gratitude to all those who have indirectly contributed to this work and supported me during my Master's study.

Contents

Chapter 1

Introduction

1.1 Context

In recent years, the advent of the Internet of Things (IoT) technology has led to a transformative shift in numerous domains, including agriculture. By integrating IoT technology with greenhouse cultivation, real-time monitoring of the growing conditions can be achieved, providing useful data for the management and optimization of plant growth.

Motivated by the opportunity to apply the knowledge gained in computer science studies to nature and environmental sustainability, this research explores the implementation of an IoT-based greenhouse monitoring system at the university's high-tech greenhouse, predominantly used by Bioengineering students.

In the existing setup within the university's greenhouse, environmental data is collected using various methods that do not utilize connected devices. This makes data retrieval and monitoring less efficient, as it requires manual intervention to access the information from each separate data collection point. Recognizing this deficiency, the research aims to develop a unified, IoT-based monitoring system capable of transmitting data to the cloud. This would allow easy, real-time access from anywhere, eliminating the need for direct interaction with the physical location of data collection. The key objectives of this project are to create a connected, scalable, and self-sustainable device that is cost-effective and easy to deploy.

1.2 Structure

The structure of this thesis is as follows:

Chapter 2: Background concepts introduces background concepts including

the Internet of Things (IoT), wireless communication protocols for IoT-based systems, The Things Network, backend technologies, and hardware communication protocols. This chapter also presents a brief overview of the current state of the art and related work in this domain, necessary to comprehend the proposed solution and the rationale behind the choices made.

Chapter 3: System architecture and design provides a detailed look into the architecture and design of the developed prototype. This includes a deep dive into the hardware components used, the electrical wiring necessary for their connection, the software implementation and data transmission aspects, and the backend infrastructure.

Chapter 4: Experiments is focused on three main experiments conducted to evaluate the performance of the proposed system: a coverage experiment, a power consumption experiment, and a real-world deployment experiment. These investigations scrutinize the power consumption of the device, the LoRaWAN network coverage within the greenhouse, and the system's operational stability and power management over an extended period, ensuring the system's energy efficiency, reliable data transmission, and real-world viability.

Chapter 5: Extending the prototype deliberates the potential improvements to the existing system. It explores the challenges and proposes solutions for integrating additional sensors into the system. The limitations of the current prototype are discussed, and possible enhancements are put forward.

Chapter 2

Background concept

In this chapter, I will introduce the general background concepts to better understand the system I will propose further. It focuses on the Internet of Things (IoT), wireless communication protocols for IoT-based systems, The Things Network, backend technologies, and hardware communication protocols. Additionally, I will present a brief overview of the current state of the art and related work in this domain.

2.1 Internet of Things (IoT)

The Internet of Things can be defined as a network of physical devices (called "things") that are embedded with sensors, software and other technologies that allow them to connect and exchange data with other devices and systems over the internet or other communication networks [1].

Over the past few years, IoT has become a worldwide interest and one of the most important technologies of the 21st century.It has gained significant attention due to its potential to revolutionize various sectors, including agriculture, healthcare, manufacturing, and cities, transforming them into 'smart' environments. For instance, in agriculture, IoT devices can monitor soil moisture levels to optimize watering schedules, while in healthcare, wearables can track a patient's vital signs in real-time [1].

The architecture of an IoT system typically comprises four interconnected components [2]:

- **Device:** This tier includes the "things" or physical objects embedded with sensors and actuators that gather and interact with information directly from their environment.
- **Connectivity:** This component refers to various technologies used by the devices to communicate and transfer data collected for further processing. These

includes different networks such as Wi-Fi, Bluetooth, LPWANs (Low Power Wide Area Networks), among others.

- **Data processing:** Once the data is collected (typically on the Cloud), it is processed using methods ranging from basic data aggregation to advanced analytics techniques, including machine learning.

- **User interface:** This element enables humans to interact with the IoT system. It includes capabilities such as visualizing the collected data, alerting users based on predefined parameters, and controlling the connected devices, among other functionalities.

These components, working in synergy, form the backbone of an IoT system. The following sections will dive deeper into each of these aspects, discussing their applications and significance in the context of a greenhouse monitoring system.

2.2 Wireless communication protocols

Wireless communication is a defining characteristic of the Internet of Things, enabling the seamless interaction between devices and systems. A crucial aspect of IoT systems is the selection of the appropriate communication protocol, which is highly dependent on the specific objectives and constraints of the system. There are three fundamental factors to consider when choosing a communication protocol: network range, data rate, and power consumption.

- **Network range:** Defines the maximum distance over which data can be reliably transmitted. This is typically classified as short-range or long-range communication.

- **Data rate:** Determines the speed at which data can be transmitted.

- **Power consumption:** Represents the energy required to transmit and receive data—an essential consideration for battery-powered devices.

The required network range substantially narrows down the options for the communication protocol. Short-range operations typically utilize protocols like WiFi, Bluetooth, or Zigbee. However, for long-range operations with low power requirements, as is the case for this project, different considerations apply. This section provides a detailed examination of the Long Range Wide Area Network (LoRaWAN) protocol, which I have chosen for the proposed solution, before comparing it to other popular Low Power Wide Area Networks (LP-WANs).

2.2.1 LoRaWAN

LoRaWAN is a wireless communication protocol that plays a fundamental role in the Internet of Things. Standing for 'Long Range Wide Area Network', LoRaWAN is an open global standard for wireless communications that was developed by the LoRa Alliance, a non-profit association. This alliance manages the LoRaWAN protocol and has overseen its rise to becoming the most widely deployed public and private network amongst all LP-WANs globally. This wide coverage makes it an attractive choice for various IoT applications [3].

LoRaWAN is a Media Access Control (MAC) layer protocol that operates on top of LoRa modulation. LoRa, which stands for 'Long Range', is a wireless communication technique rooted in Chirp Spread Spectrum (CSS) technology. It employs chirp pulses to encode data on radio waves, making it highly resistant to interference and capable of being received over vast distances. This robust modulation technique allows for the transmission of small data packets at low bit rates, over long distances. Hence, it's particularly suited for applications like low-power sensors and actuators [4].

LoRaWAN Architecture

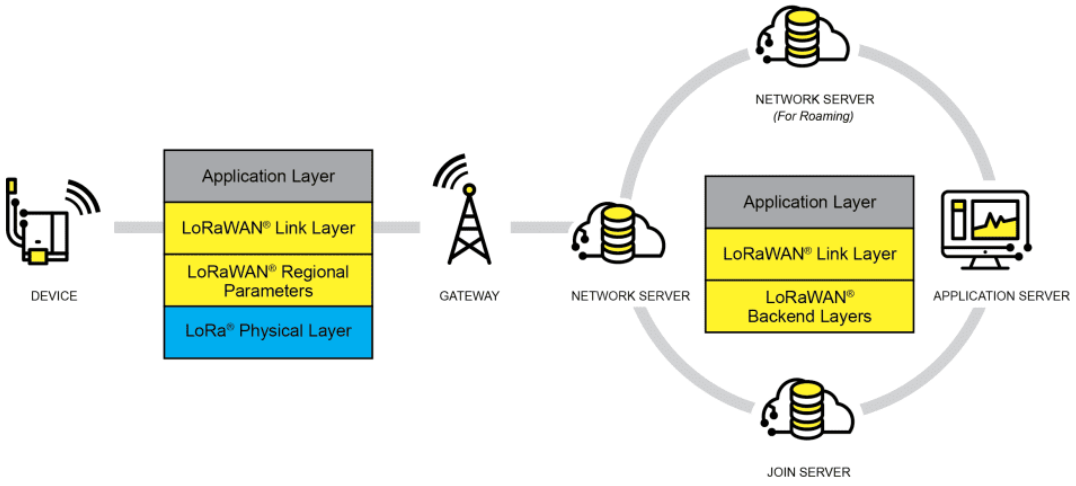


Figure 2.1: LoRaWAN network architecture [3]

As shown on the Figure 2.1, a typical LoRaWAN network architecture consists of the following key components [3, 5]:

- **Devices:** These are sensors or actuators that send LoRa modulated wireless messages to gateways or receive messages from them. These devices are the

sources or destinations of information within the network.

- **Gateways:** These devices receive messages from end devices and forward them to the Network Server.
- **Network Server:** This software runs on a server and is responsible for managing gateways, end devices, applications, and users in the entire LoRaWAN network.
- **Application Servers:** This software, also running on a server, is responsible for securely processing application data. It provides the means for the end devices to interact with user applications and services
- **Join Server:** This is another server-based software that handles join-request messages from end devices. The Join Server plays a critical role in secure device activation, root key storage, and session key generation.

End devices communicate with nearby gateways, each of which connects to the network server. LoRaWAN networks employ an ALOHA-based protocol, so end devices do not need to establish a connection with specific gateways. Instead, messages sent by end devices are broadcasted to all gateways within range [3]. The Network Server receives these messages and, if multiple copies of the same message are present, retains only one copy while discarding the rest through a process called message deduplication [5].

Regional parameters

LoRaWAN operates within the unlicensed radio spectrum, which is freely accessible to anyone, avoiding the need for costly licenses for transmission rights. Official regional specifications, referred to as Regional Parameters, are provided by the LoRa Alliance, governing the use of different frequency bands in different parts of the world [6]. The protocol operates within the unlicensed Industrial, Scientific, and Medical (ISM) bands.

In the proposed solution, the EU863-870 band, which complies with the European Telecommunications Standards Institute (ETSI) [EN300.220] standard [7], is used. This band is applicable in all European countries and some non-European countries.

EU863-870 default channels

The EU863-870 band implies a frequency range from 863MHz to 870MHz. Three default channel (868.1, 868.3, and 868.5MHz) are used by the end device to broadcast the join-request message by choosing randomly one of those. The EU863-870

band supports a maximum of 16 channels, including these three default and up to 13 additional channels.

For example, The Things Network (see section 2.3) uses the following 5 additional frequencies for uplink: 867.1, 867.3, 867.5, 867.7, and 867.9MHz.

EU863-870 duty cycle

Free use of these bands result in the necessity to regulate the access in order to avoid congestion. The ETSI sets the maximum duty cycle for the EU863-870 frequency at 1%, which is the maximum amount of time a device may spend communicating. So for example, if a device takes 1second to send a message, it has to wait at least 100seconds before being allowed to send another message.

EU863-870 data rates and maximum payload size

The data rate is the number of bits that are transmitted per unit of time. With LoRa modulation, the data rate depends on different factors such as the speading factor, the bandwidth, and the coding rate.

Data Rate	Spreading factor / Bandwidth	Bit rate [bit/s]	Maximum payload size [bytes]
0	SF12 / 125kHz	250	51
1	SF11 / 125kHz	440	51
2	SF10 / 125kHz	980	51
3	SF9 / 125kHz	1760	115
4	SF8 / 125kHz	3125	242
5	SF7 / 125kHz	5470	242
6	SF7 / 250kHz	11000	242

Table 2.1: EU863-870 data rates

As shown in Table 2.1, higher spreading factors result in lower bit rates and lower spreading factors result in higher bit rates. Furthermore, larger bandwidths allow higher data rates. Also, higher spreading factors are associated with smaller maximum payload sizes, while lower spreading factors enable larger payload sizes. These factors should be considered as they can impact the device's battery life, transmission range, and overall performance.

In summary, choosing the most appropriate regional parameters is crucial as it not only affects the operational characteristics of the LoRa device but also ensures its compliance with regional regulations and standards. The chosen regional parameters for the proposed solution are presented in section 3.3.2.

Message Types

LoRaWAN communication involves two types of messages based on the direction of transmission: uplink and downlink messages. Uplink messages are transmitted by end devices to the Network Server through gateways. If the uplink message is intended for the Application Server or the Join Server, the Network Server routes it accordingly. Downlink messages are initiated by the Network Server and sent to a single end device through a specific gateway [8].

Device Classes

The LoRaWAN specification outlines three device classes: Class A, B, and C. All LoRaWAN devices must support Class A, while Class B and Class C are optional and serve as extensions to the Class A device specification. All device classes support bi-directional communication (uplink and downlink).

Class A is mandatory for all LoRaWAN end devices. Communication in Class A is always initiated by the end device. Devices can send uplink messages at any time, and following an uplink transmission, the device opens two brief receive (downlink) windows, known as RX1 and RX2. There are specific delays between the end of the uplink transmission and the start of these receive windows. If the Network Server does not respond during these two receive windows, the subsequent downlink communication will occur after the next uplink transmission initiated by the device.

Typically, Class A devices are battery-powered, stay in sleep mode most of the time, and maintain long intervals between uplinks [9]. Given these characteristics align with the objectives of this project, the proposed solution will implement a Class A device.

Class B devices, in addition to Class A functionality, have scheduled receive slots, providing more predictable downlink opportunities. Class C devices, apart from Class A features, offer reduced latency on downlinks as they keep the receive windows open unless transmitting. These classes further extend the functionality and adaptability of LoRaWAN devices to meet diverse IoT requirements [3].

End Device Activation and security

In order to send and receive messages, end-devices must first establish a connection with the server. There are two types of end-device activation methods: Over-The-Air Activation (OTAA) and Activation By Personalization (ABP). The choice of the activation method impacts the device's security and some operational characteristics [10].

The OTAA method, considered as the most secure, requires a device identifier (DevEUI), an application identifier (AppEUI), and an application key (AppKey).

The AppKey, known only by the device and the application, is crucial for the OTAA method as it is used in the derivation of the session keys. Once a Join-Request, containing DevEUI and AppEUI, is sent to the network, the network responds with a Join-Accept message, which includes the necessary network session key (NwkSKey) and application session key (AppSKey), encrypted with the AppKey. The device then uses the AppKey to decrypt this message and obtain the session keys. These session keys are unique for each device and change with every activation, enhancing security [10, 11].

In contrast, the ABP method requires hardcoding the device address (DevAddr) as well as the network session key (NwkSKey) and the application session key (AppSKey) in the device. Unlike OTAA, the session keys remain static until manually updated, which could be a potential security risk if a key is compromised [11].

Security in LoRaWAN is underpinned by these 128-bit long session keys. The NwkSKey is shared with the network and is used for validating the integrity of each message via a Message Integrity Code (MIC). The AppSKey is used for encrypting and decrypting the payload, ensuring that only authorized entities can access the content of messages. The algorithm used for this is AES-128, similar to the algorithm used in the 802.15.4 standard.

To mitigate replay attacks, LoRaWAN uses frame counters. When a device is activated, frame counters (FCntUp for uplink messages, FCntDown for downlink messages) are set to 0 and incremented with each message. If the device or network receives a message with a frame counter lower than the last one, the message is ignored, thereby blocking repeated transmissions.

LoRaWAN also leverages Spread Spectrum techniques, including the 'CHIRP': Compressed High Intensity Radar Pulse technique, for robustness, as well as for interference immunity, channel sharing, and resistance to radio reflections [11].

Spreading Factor

As already mentioned, LoRaWAN operates on top of LoRa modulation. LoRa is based on Chirp Spread Spectrum (CSS) technology, in which chirps (or symbols) carry data on radio waves. For a visual explanation, see this video on LoRa chirps [12]. The spreading factor (SF) determines the chirp rate, thereby controlling the data transmission speed. The SF values range from 7 to 12 and influence the data rate, the transmission range, the time-on-air, and the battery life of the end devices.

The data rate in LoRaWAN is indirectly proportional to the SF. A higher SF results in a lower data rate and vice versa as shown in Table 2.1. This is because a higher SF means that the signal's chirp is longer, and consequently, fewer bits can be transmitted per unit of time.

Similarly, the transmission range is directly proportional to the SF. A higher SF

extends the communication range as the longer chirp duration improves the receiver's ability to distinguish the signal from the noise.

The time-on-air, which refers to the amount of time a message takes to be transmitted, is directly proportional to the SF. Higher SFs result in longer transmission times because each bit takes longer to transmit.

Finally, the energy consumption of the end devices is also directly affected by the SF. Since a higher SF results in a longer transmission time, the device's radio transmitter is active for a longer period, resulting in higher energy consumption and shorter battery life.

In conclusion, choosing the right SF is a balance between data rate, range, time-on-air, and energy consumption. This makes it a critical factor in the design and deployment of energy-efficient, long-range IoT applications using LoRaWAN.

2.2.2 Other Low Power Wide Area Networks (LPWANs)

Low Power Wide Area Networks (LPWANs) serve as the backbone of IoT communication, enabling the long-range connectivity of IoT devices while maintaining low power consumption. Apart from LoRaWAN, two of the most prominent LPWAN technologies are Sigfox and Narrowband IoT (NB-IoT). Both offer unique features and capabilities, rendering them suitable for different applications. This section compares these technologies based on a set of key features and analyzes their strengths and weaknesses.

Sigfox

Founded in 2010, Sigfox is a global LPWAN provider that employs binary phase-shift keying (BPSK) modulation. It uses a wide-reaching signal called Ultra Narrow Band (UNB), which provides excellent penetration and a wide coverage area. However, due to the narrow bandwidth, it offers a lower data rate, with a maximum of 100 bps. Sigfox supports small, infrequent transmissions and its simple and efficient protocol allows for long battery life on devices [13, 14]. Notably, unlike LoRaWAN, the use of Sigfox technology involves annual subscriptions, the cost of which depends on the number of messages sent and the number of devices connected [15].

NB-IoT

Narrowband IoT (NB-IoT) is a cellular-based LPWAN technology developed by the 3rd Generation Partnership Project (3GPP) to support IoT applications over licensed LTE cellular networks. NB-IoT uses a narrow band of the licensed spectrum, which provides high interference immunity and data security. NB-IoT offers higher data

rates (up to 250 kbps) and supports larger payload sizes compared to other LPWAN technologies. However, the use of the licensed spectrum means that there are network usage costs associated with NB-IoT. Furthermore, due to its higher data rates and complex protocol, NB-IoT devices typically consume more power, which may not be suitable for applications that require long battery life.

Comparaison and choice

	Sigfox	LoRaWAN	NB-IoT
Modulation	BPSK	CSS	QPSK
Frequency	Unlicensed ISM bands	Unlicensed ISM bands	Licensed LTE frequency band
bandwidth	100 Hz	250kHz and 125kHz	200 kHz
Maximum data rate	100 bps	50 kbps	200kbps
Maximum message per day	140 (UL), 4 (DL)	Unlimited	Unlimited
Maximum payload length	12bytes (UL), 8bytes (DL)	243 bytes	1600 bytes
Bidirectional	Limited / half-duplex	Unlimited	Unlimited
Range	10km (urban), 40km (rural)	5km (urban), 20km (rural)	1km (urban), 10km (rural)
Interference immunity	Very high	Very high	Low
Authentication & encryption	Yes (Data encryption is optional)	Yes (AES 128b)	Yes (LTE encryption)

Figure 2.2: Overview of three major LPWANs [14]

Given the characteristics summarized in Figure 2.2, it's clear that each LPWAN technology has its own strengths and weaknesses.

In terms of modulation, Sigfox's Ultra-Narrow Band modulation provides excellent penetration and a wide coverage area but at the expense of limited data rates. On the other hand, NB-IoT's narrowband modulation offers high data rates and payload sizes, yet consumes more power. LoRaWAN, with its Chirp Spread Spectrum modulation, finds a balance between coverage, data rate, and power consumption.

Considering the frequency and bandwidth, Sigfox and LoRaWAN operate in the sub-GHz ISM band, which is available worldwide and free to use. In contrast, NB-IoT operates within the licensed spectrum, providing interference-free communication but at a cost.

When it comes to the maximum data rate and payload size, NB-IoT stands out with superior capabilities compared to Sigfox and LoRaWAN. However, LoRaWAN still offers a decent data rate and payload size, adequate for this project.

Discussing range and interference immunity, each technology performs differently based on its modulation technique and frequency. While Sigfox and NB-IoT show strong performance, LoRaWAN's adaptive data rate and the option to increase the Spreading Factor make it highly efficient in increasing range and handling interference.

In terms of authentication and encryption, all three technologies provide mechanisms to ensure secure communication. However, LoRaWAN's dual-layer security for network and application levels offer an extra layer of protection, ensuring the integrity and confidentiality of the data.

While Sigfox and NB-IoT offer excellent capabilities for specific scenarios, LoRaWAN stands out for its versatility, striking a balance between key features like data rate, range, power consumption, and cost. This, coupled with its strong community support, open standard, end-to-end security, and deployment flexibility (private or public network), makes it a perfect fit for this project. Therefore, I chose LoRaWAN as the wireless communication protocol for this project.

2.3 The Things Network (TTN)

The Things Network (TTN) is a global collaborative IoT ecosystem that creates networks, devices, and solutions using LoRaWAN. TTN runs The Things Stack Community Edition, which is a crowdsourced, open and decentralized LoRaWAN network. The Things Stack Community Edition provide an infrastructure for managing the LoRaWAN network and handling data communication between devices, gateways, and applications.

The management application of The Things Network, called The Console, offers a user interface to register devices, manage gateways, and monitor network traffic [16]. Initiated by The Things Industries, The Things Stack Community Edition benefits from the contributions of over 100k community members from more than 100 countries worldwide [17]. TTN also offers paid plans for commercial-level projects that provide additional features and come with Service Level Agreement (SLA) guarantees [18].

2.3.1 Application

TTN allows users to create "applications" within which they can add, manage, and monitor end devices. These applications facilitate end device registration and management, as well as network data management. Once the end devices are set up, users can leverage features such as live packet transmission monitoring, packet pay-

load formatting, and adding integrations to facilitate data transfer to external services [19].

End devices

Adding a new end device on The Things Network involves either selecting predefined device information from the existing LoRaWAN device repository or manually entering the device information. Following this, users must select the appropriate frequency plan and generate the necessary keys, choose the device class, among other setup requirements [20].

Upon activation, the device's transmitted packets are visible in the live data section, where all the packet information can be monitored.

Payload formatter

To interpret data sent by end devices, TTN provides a payload formatter feature that converts binary packet payloads into readable fields. Users can choose between several types of payload formatters, including custom formatters written in JavaScript, automatic decoders for CayenneLPP formatted payloads, and Repository payload formatters provided by device manufacturers [21].

In the context of this project, a custom payload formatter was employed. The JavaScript function "decodeUplink()" was implemented to decode the received binary payloads into a JSON object (see example in section 3.3.2).

Integrations

The Things Network provides a range of integration options for further data processing, including MQTT and Webhooks [22]. MQTT is a publish/subscribe messaging protocol designed for IoT. Every application on TTN automatically exposes an MQTT endpoint. Webhooks, on the other hand, allow TTN to send application related messages to specific HTTP(S) endpoints.

2.3.2 Fair access policy

TTN's free usage of the network is regulated, as discussed in section 2.2.1. Notably, The Things Network enforces a 'Fair Access Policy' that further restricts the duty cycle beyond ESTI recommendations. This policy limits the uplink airtime to 30 seconds per day per end device and the downlink messages to 10 messages per day per end device [23]. These limitations emphasize the importance of choosing the appropriate transmission parameters to maximize the number of uplinks. Online

tools such as the airtime calculator [24] can assist in evaluating and optimizing the configuration based on the application requirements.

2.4 Backend technologies

The backend is the last part of an IoT system architecture having the role of processing, storing, and analyzing the data collected from the IoT device. The backend in IoT is typically a collection of servers, databases, and applications that work together to process and store the data from the IoT devices. These backend services can be deployed in various environments, including on-premises servers, cloud platforms, or a hybrid combination of both.

For the implementation of the backend in this project, a combination of several powerful technologies has been chosen: Docker for containerized deployment, Azure Cloud for hosting services, Telegraf for collecting and reporting metrics, InfluxDB as a time-series database for storing data, and Grafana for data visualization. Each of these technologies plays a unique role in the successful functioning of the IoT backend.

2.4.1 Docker

Docker is an open-source platform that enables the development, deployment, and management of applications in lightweight virtualized environments known as containers. A Docker container packages an application along with its code, runtime, system tools, system libraries, and settings into a standardized unit of software. This ensures that the application runs quickly, reliably, and consistently across different computing environments [25].

The strength of Docker lies in its containerization technology. Unlike traditional virtualization, where each virtual machine runs a complete operating system, Docker containers share the host system's kernel, making them lighter and faster to start. This encapsulation of the application and its dependencies ensures that it runs uniformly, regardless of discrepancies between development and staging environments.

A Docker container is defined by a Dockerfile, which includes instructions on how to build the Docker image. Once built, these Docker images can be stored and retrieved from Docker registries like Docker Hub, and then be run on any host machine that has Docker installed. This ensures consistency across various deployment environments.

In this project, Docker is utilized to containerize the Telegraf, InfluxDB, and Grafana services. This choice enhances the scalability, reliability, and interoperability of the system, as containers can be easily added, removed, stopped, or started without affecting the other services. Moreover, Docker's popularity and ease of use make it

an ideal environment for deploying each component of the backend, aligning with the project's requirements for a flexible, scalable, and reliable backend solution.

2.4.2 Azure Cloud

Azure Cloud is a cloud computing platform and online portal provided by Microsoft, offering more than 200 products and cloud services to help users build, run, and manage applications across multiple clouds, on-premises, and at the edge [26].

For this project, Azure Cloud's capabilities were specifically harnessed to host Docker containers, using services such as Resource Groups and Azure Container Instances. Resource Groups in Azure are a way to organize and manage resources that share common lifecycle, policies, and features. It is a logical container for resources deployed on Azure [27]. Azure Container Instances, on the other hand, offer the fastest and simplest way to run a container in Azure without having to provision any virtual machines or adopting any higher-level services. It offers an integrated Docker experience and can seamlessly work with the Docker CLI and Docker Compose [28].

The integration of Azure with Docker simplifies the transition from a local Docker environment to a cloud-based one, ensuring that applications can be developed locally in Docker and then easily deployed to Azure without major modifications [29]. This integration, coupled with Azure's scalability, robust security features, and flexible management tools, makes it an ideal choice for deploying and managing the backend components of this project.

2.4.3 Telegraf

Telegraf is a server-based agent for collecting and sending metrics and data. Written in Go, it compiles into a single binary with no external dependencies and requires a minimal memory footprint [30]. Developed by InfluxData, Telegraf supports a wide range of data inputs and outputs, with a flexible and versatile plug-and-play architecture.

In the context of this project, Telegraf plays a pivotal role in the data pipeline. It serves as a bridge between TTN and the InfluxDB database. By subscribing to the MQTT broker provided by TTN, Telegraf retrieves the data pushed by TTN, and forwards it to InfluxDB for storage [31]. This seamless integration between MQTT, Telegraf, and InfluxDB constitutes a key component of the project's backend architecture.

2.4.4 Influxdb

InfluxDB is a high-performance, distributed, and scalable time-series database developed by InfluxData [32]. It is specifically designed to handle large volumes of time-stamped data, which makes it an ideal fit for storing and retrieving data generated by IoT devices, system metrics, and other time-sensitive data sources.

Unlike traditional relational databases, InfluxDB is designed with unique indexing and querying capabilities well-suited for time-series data. Time-series data is characterized by high volumes of high velocity, continuously varying data points indexed by timestamp. This is a common scenario in IoT projects, where sensors continuously generate and transmit data over time [33, 34].

InfluxDB support two different query langage, InfluxQL and Flux, each of them designed for working with time-series data. Flux's syntax is designed to support chaining and pipelining of operations, allowing users to build complex queries that can handle transformations and analytics on data across disparate sources and time-ranges. Unlike SQL, which is highly optimized for joining and relating normalized data, Flux is specifically designed to handle denormalized data and operations specific to time-series data like sliding window, aggregations over time, joins on the time axis, and more.

In the context of this project, InfluxDB is used as the main data storage component in the backend. It stores data received from the IoT devices via the Telegraf agent, providing efficient storage and quick retrieval of sensor data for further processing and analysis.

2.4.5 Grafana

Grafana is an open-source platform used for analytics and interactive visualization. It allows users to create, explore, and share dashboards from a multitude of data sources. Grafana supports a wide range of data sources, including InfluxDB [35]. This compatibility with InfluxDB is further strengthened by Grafana's support for the Flux query language, enabling robust, flexible data retrieval and analysis.

Grafana provides a vast array of visualization options, including charts, graphs, tables, and maps, thus offering users the flexibility to represent their data in a way that best fits their needs and aids understanding. These visualization options, combined with its capability for real-time data monitoring, make Grafana an ideal tool for displaying the data collected from IoT devices.

In this project, Grafana plays a pivotal role in the visualization of sensor data collected from the greenhouse. The data is collected by Telegraf, stored in InfluxDB, and then visualized on a Grafana dashboard. This interactive dashboard enables real-time monitoring and exploration of the greenhouse data, thus providing critical

insights and facilitating data-driven decision-making.

2.5 Hardware communication protocols

In this section I will present the basics of hardware and the different protocols used for communication. The hardware requirement to create an IoT device comprise three elements: a microcontroller, sensors, and power supply. Each of the component are connected with wires for powering and communications between the microcontroller and the external modules (sensors). Here is general presentation of the different communication protocols that a microcontroller supports. In communication protocols, the main component is called the "master" and the external devices (sensors) are called "slaves".

2.5.1 Universal Asynchronous Receiver/Transmitter (UART)

The UART protocol is one of the simplest and oldest form of device-to-device digital communication. UART is a two-wire communication protocol that uses a single wire for transmitting (Tx pin from master connected to Rx pin of slave) and a second one for receiving data (Rx pin from master connected to Tx pin of slave). The data is transmitted asynchronously meaning that there is no clock signal to synchronize the output of bits from the transmitting device to the sampling of bits by the receiving device. Instead of a clock signal, start and stop bits are added to the data packet being transferred defining the beginning and the end of the data packet. The main advantages of the type of communication are that it requires only two wires, no clock signal is needed and has a parity bit to allow error checking. But the big problem of this communication protocol is that it only supports one slave per master meaning that each sensor takes two wires.

2.5.2 Serial Peripheral Interface (SPI)

The SPI protocol is a synchronous serial communication interface between the microcontroller and one or multiple sensors. The synchronous interface means It requires a clock signal to transfer and receive data, only the master controls the clock and provide clock signal to all slaves devices. Four pins are required such as: the SCLK pin, which the master employs to relay clock signals; the CS pin, which is utilized by the master for sensor selection during data transfer; the MOSI pin, enabling data flow from master to slave; and the MISO pin, used for data transmission from slave to master. This protocol requires minimum four wires as first but can communicate with multiple external devices using only one additional CS pin per sensor added.

2.5.3 Inter-Integrated Circuit (I2C)

This protocol combines the best features of UART and SPI as it only needs 2 wires to transmit data and can communicate with multiple slaves using the same wires. The two pins needed are the SDA where the data is transferred (blue link on Figure 3.5) and the SCL where the clock signal is carried (green link on Figure 3.5). The reason why multiple slaves can be connected using the same wires is that each slave is identified by a 7bits address. With I2C, the data is transferred in messages which are broken into frames of data. Each message has an address frame that contains the slave's address, and one or multiple data frames that contains the data being transmitted. The messages also contains start and stop bits, read/write bits, and ACL/NACK bits between the frames.

2.6 State of the art

The Internet of Things (IoT) has been increasingly applied in the field of greenhouse monitoring systems, leading to significant advancements in the sector. Several studies and implementations have been conducted in this area, demonstrating the potential of IoT in enhancing the efficiency and effectiveness of greenhouse management.

For instance, the paper "Internet of Things Approaches for Monitoring and Control of Smart Greenhouses in Industry 4.0" presents the current state of the art in the IoT-based applications to smart greenhouses, underlining benefits and opportunities of this technology in the agriculture environment [36]. The paper discusses the role of IoT in real-time monitoring, data collection and processing, and efficient control of indoor parameters such as light exposure, ventilation, humidity, temperature, and carbon dioxide level.

The study "Design and Implementation of an IoT Based Greenhouse Monitoring and Controlling System" provides an example of an IoT-based system for greenhouse monitoring and control, emphasizing the flexibility and suitability of such a system for maintaining the proper environment inside the greenhouse [37]. The system integrates heating, cooling, and water supply facilities, allowing for continuous online monitoring and control of temperature, humidity, and soil moisture.

In the paper "A layered IoT architecture for greenhouse monitoring and remote control", a Networked Control Systems-based architecture is proposed for two greenhouses, built on top of switched Ethernet and Wi-Fi [38]. The architecture introduces fault tolerance at the controller level, ensuring continuous operation even in the event of a controller failure. The paper also presents a channel allocation scheme that prevents interference in the greenhouse system.

Finally, the paper "Greenhouse Automation Using Wireless Sensors and IoT In-

struments Integrated with Artificial Intelligence" discusses the integration of wireless instruments with artificial intelligence (AI) algorithms and knowledge-based decision support systems for greenhouse automation [39]. The chapter highlights the advantages of such an approach, including implementation flexibility, energy reduction, and yield predictability.

These studies are representative examples of the current research in IoT-based greenhouse monitoring systems. They illustrate the transformative impact of IoT technology in the field, providing valuable insights for the development of the proposed system in this project.

2.7 Related work

Several previous works centered on the theme of "Louvain-la-Neuve Smart City" have significantly influenced this project, offering both technical insights and practical perspectives for the deployment of an IoT-based solution.

Robin Schoenmaker's research presented a comprehensive evaluation of LoRaWAN's network coverage and performance within the city [40]. This study served as a valuable reference point for understanding the reliability and effectiveness of LoRa and LoRaWAN in Louvain-la-Neuve.

François De Keersmaeker and Nicolas Van De Walle illustrated the practical feasibility of implementing an end-to-end system for monitoring environmental data using LoRaWAN and Wi-Fi technologies [41]. Their exploration of these technologies for gathering and transmitting data demonstrated the potential efficiency and effectiveness of such a system, which has directly influenced the development and design of the monitoring system in this project.

Olivier Latteur and Hadrien Libioulle developed a LoRa and LTE-based bicycle tracker, showcasing innovative ways to implement IoT solutions using LoRaWAN technology [42].

While each project was distinct in its scope and focus, all contributed to a comprehensive understanding of the challenges, possibilities, and practical aspects of IoT deployment in a smart city context. These contributions have served as guiding examples for the deployment and implementation of the greenhouse monitoring system.

Drawing on the insights gained from these studies and previous projects, this research aims to further advance the field of IoT-based greenhouse monitoring systems, addressing existing challenges and exploring new possibilities for innovation and improvement. The following chapters will detail the design, implementation, and evaluation of this system.

Chapter 3

System architecture and design

As explained in section 1.1, the objective of this thesis is to design and implement an IoT-based greenhouse monitoring system that is cost-effective, energy-efficient, and easy to deploy.

In this chapter, I provide an in-depth look into the architecture and design of the developed prototype. I will delve into the details of the hardware components used, such as the main board, sensors, power supply, and battery monitoring, as well as the electrical wiring necessary for connecting these components. Furthermore, I will explore the software implementation and data transmission aspects, including the lifecycle of the end device, packet format, and the use of Arduino IDE.

Lastly, I will present the backend infrastructure, examining the deployment of Azure Cloud, integration with The Things Network (TTN), and the utilization of Telegraf, InfluxDB, and Grafana for efficient data processing and visualization.

The code related to the prototype is available on Github at: `https://github.com/phurd02/uclouvain-iot-greenhouse`.

3.1 Overview of the proposed solution

The prototype for the IoT-based greenhouse monitoring system is designed to collect data from the greenhouse environment, transmit it using a wireless communication protocol, store it in a database and visualize the data on a dashboard. In this section, I will provide a high-level overview of the entire process, from the end device to the visual representation of the data in Grafana.

Figure 3.1 illustrates the four core components of the system:

- End device: The end device is responsible for collecting data from the attached sensors and transmit them. It utilizes the WiFi LoRa 32 (V2) board as its central component and is equipped with BME280 and AS7265X Spectral Triad

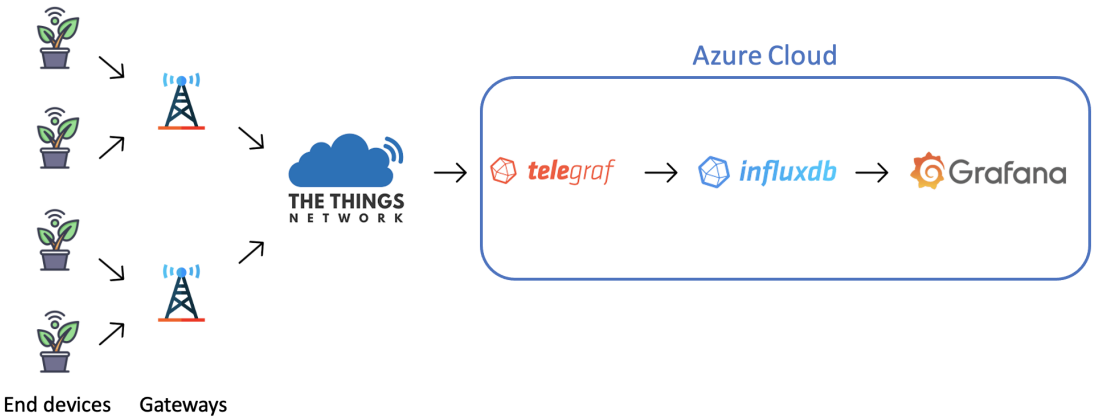


Figure 3.1: General architecture

sensors. The device is powered by a lithium rechargeable battery and a solar panel, managed by a TP4056 charger module.

- LoRaWAN gateway: The data gathered by the end device is transmitted using LoRaWAN. The LoRaWAN gateway receives the data and forwards it to The Things Network.
- The Things Network: I employed The Things Network to manage the devices and handle the transmitted packets. This platform functions as an intermediary between the LoRaWAN gateway and the cloud-based backend system.
- Cloud-based backend system: The backend system is hosted on the Azure Cloud and comprises three Docker images: Telegraf, InfluxDB, and Grafana. Telegraf collects data from The Things Network, InfluxDB stores the time-series data, and Grafana provides visualization of the data on user-friendly dashboards.

These components will be presented in detail in the following subsections.

3.2 Hardware components

3.2.1 Main board

The WiFi LoRa 32 (v2) board [43], designed and produced by Heltec Automation [44], was selected as the central component of the system. Powered by the ESP32 chip, a dual-core 32-bit MCU + ULP core, this board integrates the SX1276 [45] LoRa

node chip, offering LoRa communication support. Despite the board's capability to support WiFi and BLE communication, these features were not employed in the system since only long-range communication was required. An onboard OLED display initially served for debugging during development, but was later disabled due to its substantial energy consumption.

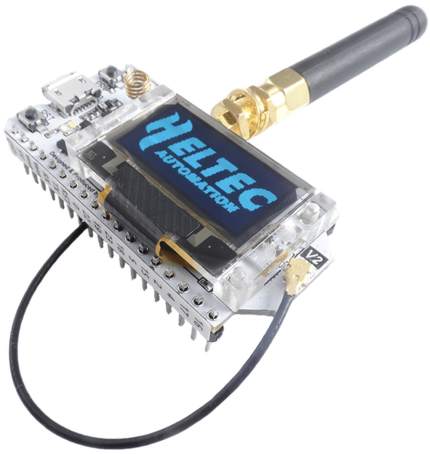


Figure 3.2: WiFi LoRa 32 (V2) board

I chose this board for several reasons. First, it uses the ESP32 chip, which is a low-power and highly integrated solution designed for IoT applications [46]. The ESP32 is widely used, meaning that numerous resources are available online, making it easier to work with. The integrated LoRa chip makes the board a suitable choice for development, and its compatibility with the Arduino IDE and various libraries proved helpful when starting the project and understanding how the board functions. Moreover, the integrated battery management system is beneficial for creating an energy-efficient device.

HTCC-AB01

Initially, I developed the prototype using the WiFi LoRa 32 (v2) board as the central component. However, upon discovering the energy-efficient HTCC-AB01 [47] board, I was intrigued by its potential benefits for the project and decided to evaluate its feasibility.

The HTCC-AB01, compared to the WiFi LoRa 32 (v2), provides lower energy consumption and an integrated solar panel power management system, thereby enhancing overall device efficiency. Despite these appealing features, the board's limited hardware resources could potentially restrict the addition of future sensors, thereby constraining the system's adaptability to evolving monitoring requirements.

Due to damage during the soldering process, the HTCC-AB01 was unfortunately rendered unusable, precluding a direct comparison of energy consumption with the WiFi LoRa 32 (v2) board in the experimental chapter. Yet, the advantages of the HTCC-AB01 provide valuable insights for future iterations of the project. In contrast, the WiFi LoRa 32 (v2) was ultimately chosen for the prototype due to its broader hardware resources, which would enable more flexibility for potential future sensor additions, even if it lacks the advanced solar power management features. I will detail in a further section how I successfully integrated solar power supply to the WiFi LoRa 32 using an external component.

3.2.2 Sensors

To collect comprehensive environmental data from a greenhouse, I selected two sensors: the BME280 and the AS7265X Spectral Triad Sensor.

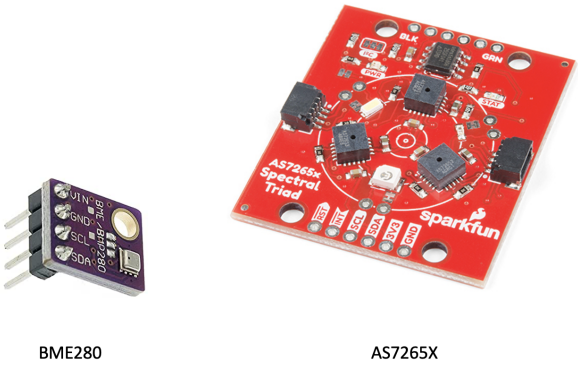


Figure 3.3: Sensors used in the prototype

The BME280 is a high precision digital sensor that measures humidity, barometric pressure and ambient temperature [48]. The small size and the low-power consumption are the key design parameters that makes it perfect for this project.

On the other hand, the AS7265X Spectral Triad Sensor is module that combines three spectral sensors, detecting light wavelengths ranging from 410nm (UV) to 940nm (IR). It can measure 18 individual light frequencies with precision down to $28.6nW/cm^2$ and accuracy of $\pm 12\%$ [49].

The combination of these two sensors enables the system to collect crucial greenhouse data, including temperature, humidity, pressure, and light. Additional sensors, such as soil moisture, CO2 level, plant growth monitors, and others, could be integrated for more extensive data collection. This potential for future expansion will be discussed further in a further chapter.

3.2.3 Power supply

To power the board with a solar panel, an external battery charging and management system is required, as the board does not include any integrated solar panel management features. The TP4056 [50] battery charger module is chosen as a suitable solution for this purpose.

The solar panel outputs between 5V and 6V with direct sun. Since the board does not have an integrated solar panel management system, an external module is necessary to regulate the charging process and maintain the health of the lithium battery. The TP4056 battery charger module is a widely used and cost-effective solution for charging single-cell lithium-ion or lithium-polymer batteries. It provides constant current and constant voltage charging, with programmable charging current (up to 1A) through an external resistor. The module also features an integrated battery protection circuit that prevents overcharging, over-discharging, and short-circuiting.

The solar panel charges the lithium battery through the TP4056 battery charger module. The module is responsible for regulating the charging process to ensure that the battery is charged safely and efficiently, while preventing overcharging and other potential issues. The lithium battery outputs 4.2V when fully charged, providing a stable power source for the board.

The board is equipped with a JST connector for easy connection to the lithium battery, as well as a voltage regulator to maintain a constant supply voltage for the board's components, ensuring reliable operation. The overall power system functions by harvesting energy from the solar panel, converting it to the appropriate voltage and current levels required to charge the lithium battery, and then supplying the board with the necessary power. This setup allows for continuous operation without the need for external power sources, making it ideal for remote greenhouse monitoring applications.

3.2.4 Battery monitoring

Monitoring the battery level of the device is crucial to ensure optimal performance and prevent unexpected shutdowns. This allows for timely replacement or recharging of the battery, depending on whether a solar panel or another external power source is used. The battery level can be deduced from its output voltage, with a fully charged battery at 4.2V and a nearly depleted battery at 3.2V. However, since the board's maximum readable voltage is 3.3V, the battery output voltage must be reduced for it to be read. To achieve this, I employed a voltage divider, a straightforward, cost-effective solution requiring just two resistors.

Voltage dividers

A voltage divider is a basic electronic circuit designed to reduce an input voltage (Vin) to a smaller output voltage (Vout). It consists of two resistors connected in series (as shown in Figure 3.4), with the primary objective of generating a specific ratio between the input and output voltages, determined by the resistors' values. The output voltage can be calculated using the following formula:

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_2}{R_1 + R_2}$$

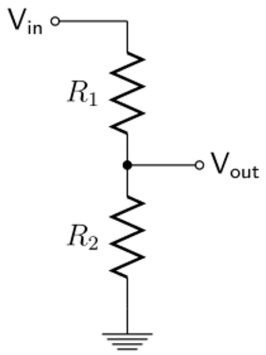


Figure 3.4: Voltage divider [51]

Voltage divider implementation

For this project, I utilized two resistors, both with a resistance value of 100k ohm. This equal resistance results in the output voltage (Vout) being half of the input voltage (Vin). Consequently, the battery output voltage range from 3.2V-4.2V is reduced to a board-readable range of 1.6V-2.1V.

The device's power supply is linked to the JST connector and the voltage divider, while the voltage divider output connects to pin 36 on the board. As an ADC (Analog to Digital Converter) pin, pin 36 transposes the analog voltage into a 12-bit digital number, denoting a value between 0 and 4095, where 0 corresponds to 0V, and 3.3V aligns with 4095. Theoretically, the analog voltage value will reside within the range of 1985 to 2606, considering the voltage interval of 1.6V-2.1V. A more detailed calibration and mapping explanation is presented in section 4.2.4.

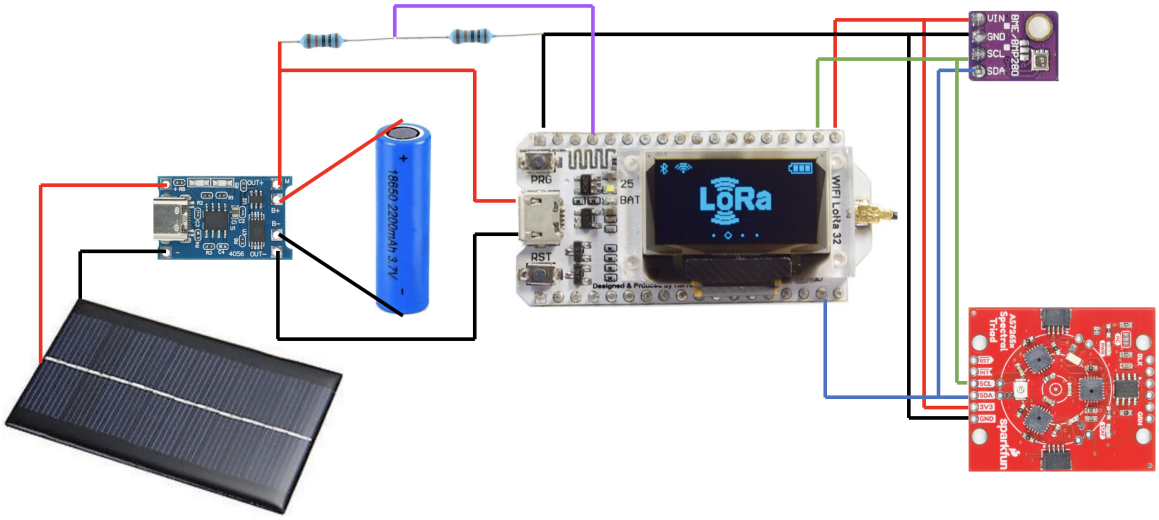


Figure 3.5: Wiring schematic

3.2.5 Electrical wiring

Figure 3.5 details the wiring schematic for the proposed system. The board has several voltage output pins available to power the sensors, including multiple pins providing 3.3V and one supplying 5V. However, these pins are not programmable and persistently supply power, which is inefficient as the device only necessitates sensor usage during extended periods.

To accommodate this, I've opted to use pin 21 (see pinout diagram [52]), a controllable voltage pin that can output 3.3V on demand. Hence, pin 21 powers the sensors only when data collection is warranted, significantly improving energy management. Additionally, each sensor must be connected to a ground pin.

Hardware components communicate and exchange control signals via various protocols (explained in section 2.5). The current prototype employs the I2C protocol for interactions between the board and sensors. The WiFi LoRa 32 pinout diagram [52] suggests the potential for other communication protocols, with numerous pins still unutilized. Nonetheless, several pins are already committed to onboard components like the OLED display and the LoRa chip.

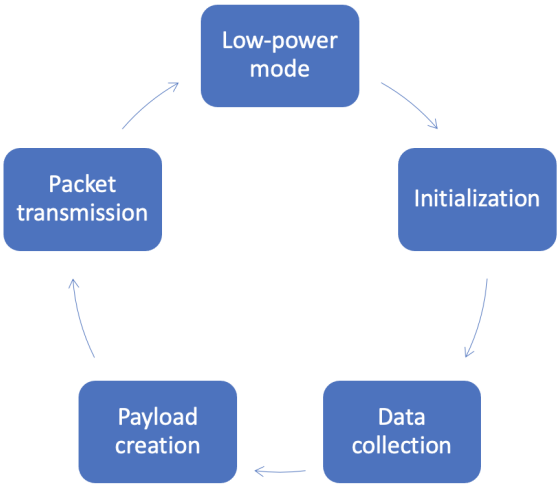


Figure 3.6: Phases of end device

3.3 Software implementation and data transmission

3.3.1 Lifecycle of the end device

Figure 3.6 illustrates the different states that the device will go through. As the data only needs to be collected at intervals of a few minutes, the device will spend most of its time in low-power mode. Let's detail each state:

Initialization

Upon startup, the end device initializes the hardware components, such as powering the sensors, LoRa transceiver, and onboard components. It also establishes communication with the sensors using the appropriate protocols (e.g., I2C) and configures the LoRa transceiver to use the correct frequency, spreading factor, and other parameters for data transmission.

Data collection

Once the end device is initialized, it collects data from the sensors. The data collection process involves reading the sensor values, processing the raw data, and formatting it according to the packet format described in the next section.

Payload creation

After collecting data from each sensor, the data is formatted and added to the payload byte array.

Packet transmission

After collecting and formatting the sensor data, the end device transmits the data to the gateway using the LoRaWAN protocol. The transmission involves encrypting the data, adding metadata (such as device identifiers and timestamps), and transmitting the packet.

Low-power mode

To conserve energy, the end device enters a low-power mode after transmitting the data. In this mode, most of the device's components, are powered down to minimize energy consumption. The device is configured to wake up periodically to perform the data collection and transmission tasks again.

3.3.2 Packet format

The LoRa packet format is composed of three elements: the preamble, the header (optional), and the payload. The payload contains all the data collected from the sensors and has a maximum size based on different LoRaWAN regional parameters, as explained in section 2.2.1.

This is the list of LoRaWAN regional parameter used in the solution:

- Frequency plan: EU863-870
- Bandwidth: 125 kHz
- Spreading factor: 7
- Coding rate: 4/5

The size of my payload is 44 bytes, here is the content:

Payload content	
Data	Size [bytes]
Battery	2
Temperature	2
Humidity	2
Pressure	2
18 Spectral channels	36

According to the online airtime calculator [24], based on the regional parameters and the packet size, each packet takes approximately 107.8ms of airtime. Based on the data shown on the figure 3.7, the airtime calculator estimation is very close to the real world data. As explained in section 2.3.2, TTN has a fair access policy that allows at most 30 seconds of uplink airtime per 24 hours. Theoretically, my device could send 11 packets per hour. However, the actual airtime depends on many different factors, such as weather and gateway location.

Figure 3.7: Packet data from TTN console

To optimize the payload size, I employed fixed point representation. This technique involves converting float values to integers by multiplying them by 100, effectively reducing the size of each data element from 4 bytes (for floats) to 2 bytes (for integers), yet still preserving the decimal precision.

When TTN receives binary payloads, it translates them into JSON objects (as shown in the Figure 3.7). The payload formatter is customizable, allowing me to divide each data value by 100 after transmission to obtain the original value with the correct decimal places. This approach ensures efficient data transmission and minimizes potential data loss while reducing the overall payload size, resulting in shorter transmission times and better compliance with the fair usage policies of the LoRaWAN network.

3.3.3 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a software application that I used to write, compile, and upload code to the microcontroller. Arduino IDE is a popular and open-source development platform that supports a wide range of microcontrollers, including the Wifi Lora 32 (v2) board. It provides a user-friendly interface and supports the programming languages C and C++. I chose to use Arduino IDE for many reasons such as its ease to use, the extensive libreary support, and the compatibility with the board I worked with.

3.4 Backend implementation

3.4.1 Azure Cloud deployment

For deploying the backend system, I chose Azure Cloud for different reasons. Firstly, Azure offers an integrated Docker environment, allowing for easy deployment, management, and scaling of containerized applications such as Telegraf, InfluxDB, and Grafana. Secondly, as a student, I received €100 of Azure credit, which enabled me to access and utilize the platform's resources without incurring additional expenses. Lastly, Azure Cloud's extensive range of services, global availability, and robust security features make it a reliable and secure choice for hosting IoT applications.

As explain in section 2.4.2, Azure Cloud organizes its resources using a hierarchical structure called Resource Groups [27]. Within the Resource Group I created, I deployed Docker container instances [28] for Telegraf, InfluxDB, and Grafana.

3.4.2 TTN integration

As explain before, The Things Network offers a set of integration such as: MQTT server, Webhooks, storage integrations, and others. I decided to use TTN's MQTT integration to transmit data to the backend system. MQTT is a lightweight messaging protocol that operates on a publish-subscribe model, making it well-suited for IoT applications [31]. The IoT devices act as publishers, sending sensor data to a specific topic on the TTN MQTT broker. the Telegraf agent acts as a subscriber, and receives the sensor data by subscribing to the relevant topic.

3.4.3 Telegraf and InfluxDB

I configured the Telegraf agent to consume the data from TTN as input and forward it to an InfluxDB instance using the TCP protocol. I selected InfluxDB, a high-performance, distributed, and scalable time-series database, to handle the large

amounts of time-stamped data generated by IoT devices. Its unique indexing and querying capabilities make it well-suited for efficiently storing and retrieving sensor data.

3.4.4 Grafana

For data visualization, I chose Grafana, an open-source platform for analytics and visualization. Grafana's support for a wide range of data sources, including InfluxDB, and its various visualization options made it an ideal tool for displaying the data collected from the IoT device. Figure 4.11 shows a dashboard example with the data collected by a deployed device.

Chapter 4

Experiments

In this chapter, I will discuss the three main experiments conducted to evaluate the performance of the proposed system: a coverage experiment, a power consumption experiment, and a real-world deployment experiment. These experiments focus not only on the power consumption of the device and the LoRaWAN network coverage within the greenhouse, ensuring the system's energy efficiency and reliable data transmission, but also test the system's operational stability and power management over an extended period. By examining these aspects, I aimed to confirm the system's practical viability and its alignment with the project objectives.

4.1 Network coverage experiment

The first experiment focused on the LoRaWAN coverage, ensuring that the device could use the desired network. Initially, I assumed that the network would be available, as previous theses [40] had already tested it, and typically, greenhouses are made of glass and other materials that do not obstruct radio waves. However, my assumption was put into question during my visit to the school's greenhouse, as UCL's greenhouse is a high-tech structure, with numerous metal structures everywhere. To guarantee reliable data transmission, I decided to evaluate the actual coverage inside the greenhouse, instead of basing my assumption on outdoor coverage data.

4.1.1 Experiment setup

Initially, I tried to use available coverage testing tools like the TTN Mapper, a community project that enables anyone to contribute data to create a map of actual coverage by the TTN gateways. The TTN Mapper is a free integration that only requires a device capable of sending GPS coordinates. The device sends packets with

the coordinates, and the signal data is stored and visually available on a map (see URL). Unfortunately, this method was not conclusive as the GPS (GTU7) module connected to my board was unable to determine coordinates indoors, resulting in no data being sent.

Alternatively, I defined a set of locations within the greenhouse. At each location, the device sent ten packets containing a counter from 1 to 10. For each packet received, I checked the counter value to ensure no packets were missing. Additionally, I collected the Received Signal Strength Indicator (RSSI) and the Signal-to-Noise Ratio (SNR), which are the two most commonly used signal strength indicators. I set the device's spreading factor parameter to 7 to evaluate the "worst" range case, as explained in section 2.2.1, the smaller the SF, the smaller the range.

RSSI and SNR

The RSSI is a relative measurement that determines whether the received signal is strong enough to ensure a good wireless connection from the IoT device. RSSI is measured in dBm, and its value is typically negative; the closer the value is to zero, the stronger the received signal [53].

The SNR is the ratio of received signal power to the noise floor. The noise floor represents the minimum level of background noise always present in the environment, caused by various factors such as atmospheric noise, thermal noise, and interference from other devices operating in the same frequency band. The SNR is measured in decibels (dB); the higher the value, the better the quality of the received signal [53].

4.1.2 Result and analysis

The data obtained from the experiment is detailed in Table 4.1, and its graphical representation can be seen in Figure 4.1.

During the experiment, I observed an anomaly at location P6, where two SNR values were missing from the data, even though the packets were received, and RSSI values were recorded. This anomaly only occurred at this specific location and could be attributed to various factors such as interference, obstacles, or gateway issues.

Despite its inaccuracy, I made an assumption that the SNR value was 0 for several reasons. Firstly, the extreme RSSI value concurred with this notion; for instance, at location P5, the RSSI value of -110 corresponded with an SNR of 0.5, indicating that the received signal power was very close to the noise floor power. Thus, it's plausible that with an RSSI of -114, the SNR is close to 0. Secondly, location P6 is in the middle of the greenhouse, theoretically the worst position compared to other positions on the sides.

Location	Avg SNR [dB]	Avg RSSI [dBm]
P1	8.88	-93.2
P2	8.41	-94.9
P3	8.87	-93.8
P4	8.19	-96.9
P5	7.29	-98.6
P6	6.20	-98.0
P7	6.98	-100.6
P8	7.88	-95.8
P9	7.85	-100.4
P10	9.38	-98.4
P11	8.61	-93.0
P12	8.12	-84.9

Table 4.1: Average RSSI and SNR values

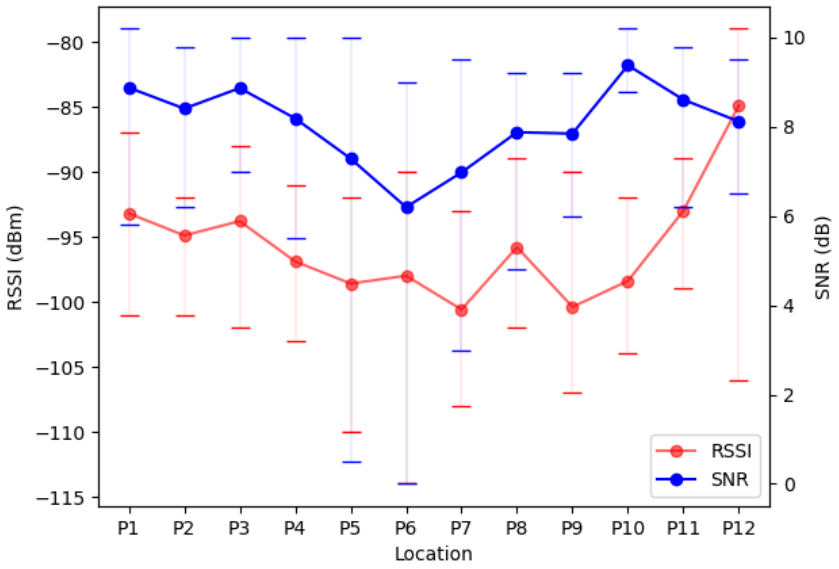
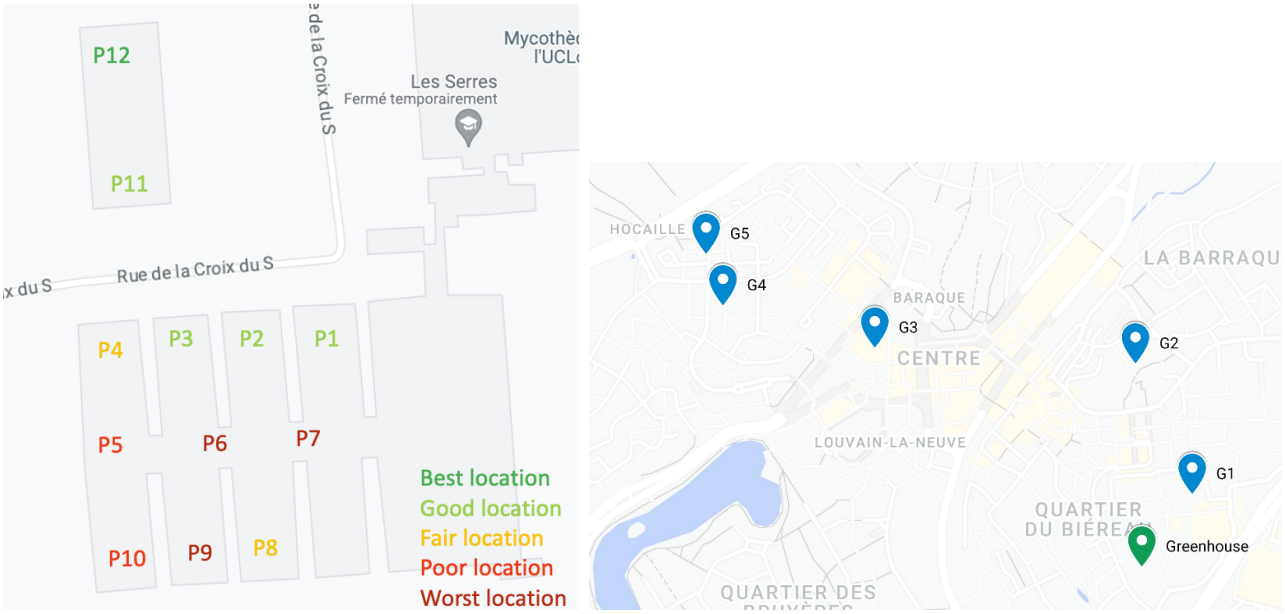


Figure 4.1: Graphical representation of the data

For clarity and visual representation, I set the SNR to 0, but this introduces uncertainty regarding the accuracy at that location. If these two data points were removed, it would significantly alter the interpretation as it would yield a better average SNR and average RSSI, suggesting that it is potentially a good location. However, this is

not the case due to data transmission instability.



(a) Measurement positions (b) Gateways locations

Figure 4.2: Coverage experiment maps

Despite the missing SNR values at location P6, the rest of the experiment results provide a comprehensive understanding of the actual network coverage. Notably, all ten packets were received at each location, confirming the network's availability within the greenhouse. Based on the results, the best network coverage is found at locations P1, P2, P3, P11, and P12, where the average SNR and average RSSI values are higher compared to other locations. For visual clarity, I color-coded the different positions in Figure 4.2a based on the results. This agrees with the location of the available gateways shown in Figure 4.2b as they are positioned to the north of the greenhouse. The packets were generally captured by three of the five gateways (G1, G2, and G4 on Figure 4.2b). Looking at the map, one might expect the gateway at location G3 to receive the packets as it's closer than G4. This can be explained by the "valley" topography of Louvain-la-Neuve, where G3 is situated in the lowest part of the city, while the others and the greenhouse are located on the higher side.

4.2 Power consumption experiments

In this experiment, I focused on evaluating the power consumption under different scenarios: without LoRaWAN communication, with LoRaWAN communication, and with different CPU frequencies. By analyzing the power consumption of the device in these different scenarios, I was able to assess the energy efficiency of the system, understand the impact of LoRaWAN communication and voltage divider on power consumption, and estimate the theoretical battery lifetime.

4.2.1 Experiment setup

To conduct the power consumption experiment, I used the electrical devices provided by the WELCOME (Wallonia Electronics and Communications Measurements) platform for students. These devices provided a consistent and precise means for measuring the power consumption.

equipment used:

- A DC power supply: I used the E3630A DC power supply from The Keysight Technologies. With 0.01% load and line regulation, the E3639A can maintain a steady output when power line or load changes occur which is important for maintaining a stable output and perforom accurate measurements. It offers three different independant output power range including a low range from 0 to 6volts at 5amps wich is whats needed to power up the mircocontroller.

- A digital multimeter: I used the DMM7510 $7\frac{1}{2}$-digit graphical sampling multimeter from Tektronix. This instrument offers a current measurement capability from 100pA to 10A with a 100kHz analog bandwidth that is suitable for both deep sleep as well as active current measurements. it can store the measurements in buffers that can be exported as CSV files to a USB key.

As shown on Figure 4.3, the power supply, and the digital multimeter were connected in a series configuration. The series connection allowed for a direct measurement of the current flowing through the IoT device, providing accurate data for analysis. I set the power supply to output 3.7V and set the multimeter to capture the current (mA) every ms making a total of 1000 data point every seconds.

To evaluate the power consumption of the device, I analyzed the CSV files exported from the multimeter. These files contain the current values (mA) measured at each timestamp (ms). By using the formula:

$$P(t) = V \times I(t) \tag{4.1}$$

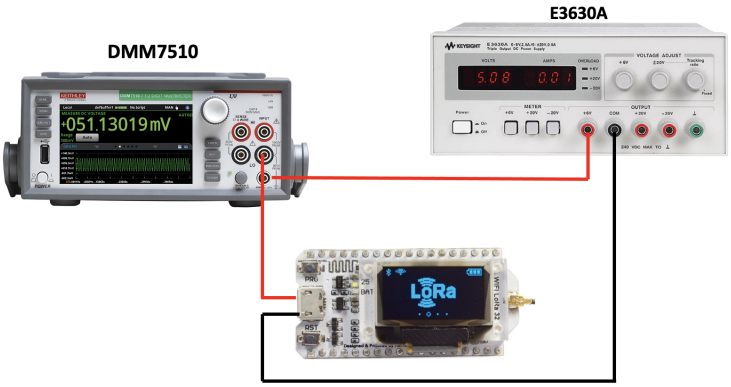


Figure 4.3: Experiment setup for power consumption measurements

Where $P(t)$ is the power at time t, V is the supply voltage (3.7V in this case), and $I(t)$ is the current at time t.

I calculated the power consumption at each timestamp, and then integrated over the desired time interval to find the total energy consumed. The energy consumed (in mWh) can be calculated using the formula:

$$E = V \times \int_{t_{min}}^{t_{max}} I(t)\, dt \tag{4.2}$$

Where E is the energy consumed, t_{min} and t_{max} represent the start and end of the desired time interval, and $I(t)$ is the current at time t.

The set of data collected made are useful to understand the device electrical behaviour, the power consumption during different phases, and calculate the potential battery lifetime.

4.2.2 Result and analysis

I processed the collected data using python scripts in order to generate graphs and to calculate power consumption. Each data point in the CSV files contains a current value in mA with an accurate precision of 4 decimal places and a sampling rate of 1 ms.

I collected data under different operating conditions, which allows me to analyze various power consumption aspects such as the power consumption of each sensor, the power consumption of the LoRaWAN transmission process, the power consumption when the device is in deep sleep (low-power mode), the power consumption of each phase of the device lifecycle (detailed in section 3.3.1), the impact of the device CPU frequency, and estimate a battery lifetime.

Sensors

In this section, I analyze the power consumption of individual sensors during their operation and data collection phases. To do this, I examine the graphs generated from the collected data, focusing on the periods when the sensors are active.

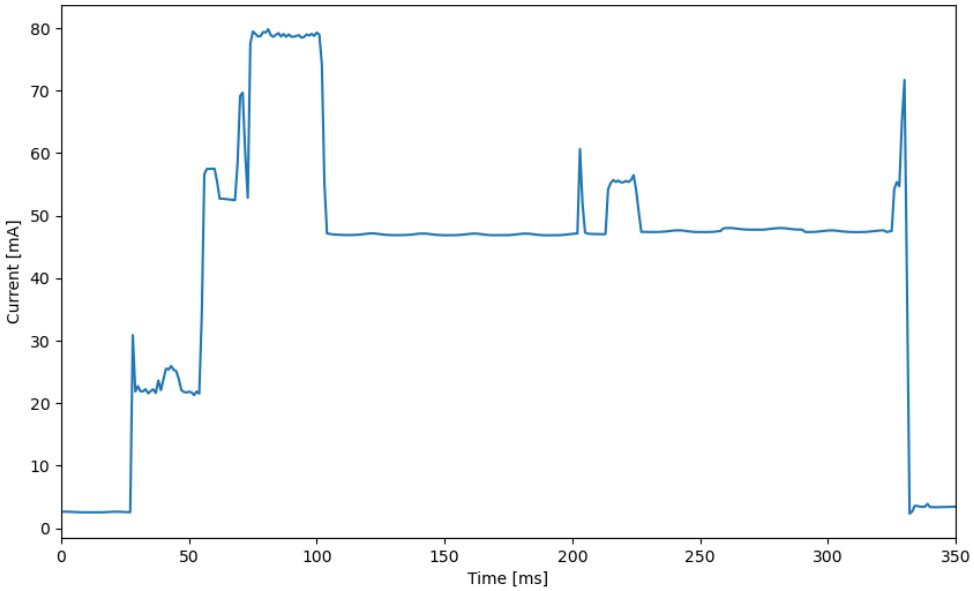


Figure 4.4: BME280

Figure 4.4 shows the power consumption of the device when only the BME280 sensor is connected. Based on the graph, several event can be identified such as the microcontroller initialization when waking up from low-power mode, the sensor being powered up, the communication being established between the microcontroller and the sensor, the data being collected, and finishing with the device going back to low-power mode. The total time in active mode is 305ms with an average current of 40.1mA which gives a total charge consumption of 0.0041mAh.

Figure 4.5 shows the power consumption of the device when connected to the BME280 sensor and to the AS7265X sensor. The impact of adding the AS725X sensor on the total time and the average current consumption is clearly displayed. The different events identified on the previous graph are present on this one too. Here we can see that the data collection takes much longer which concurs with the fact that the AS7265X collect data of 18 different spectral frequency channel. The total time in active mode is 1954ms which make it more than 6 times longer then without

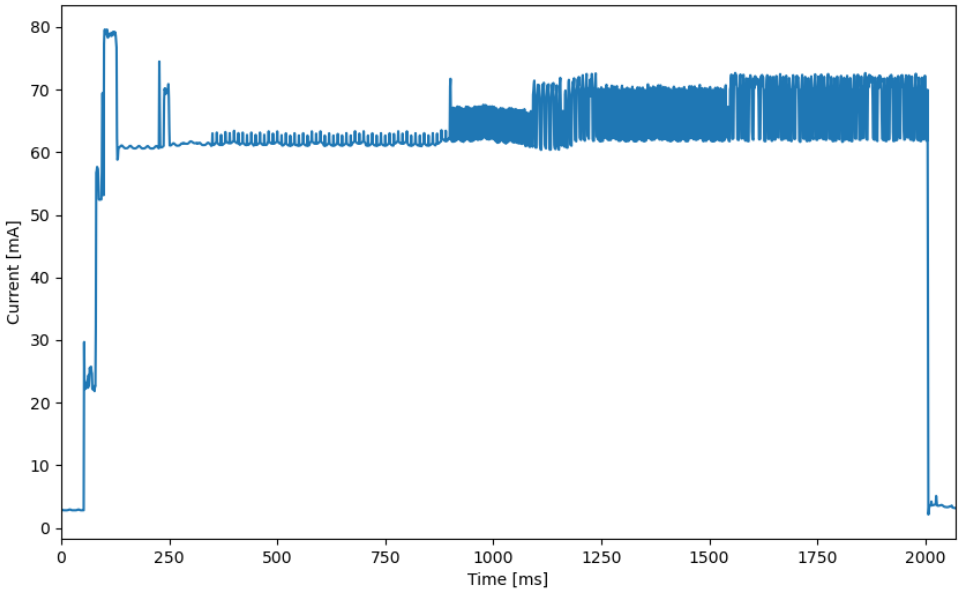


Figure 4.5: BME280 and AS7265X

the AS7265X sensor connected. The average current of 63.3mA gives a total charge consumption of 0.0343mAh.

Based on these results, we see that the AS7265X sensor is a power-hungry sensor compared to the BME280. It's a very precise sensor that collects many data, if the light data needed does not require such precision, maybe using a less power-hungry sensor could be considered. Some optimizations could also be made, such as disabling the light data collection during the night, to further reduce power consumption, this will be discussed in a further section.

LoRaWAN

In this section, I analyze the power consumption of the LoRaWAN communication process. The Figure 4.6 shows the power consumption of the device connected to the BME280 when LoRaWAN data transmission is enabled.

When comparing with the Figure 4.4 and the previously identified events, new events used for the data transmission are visible such as the packet being created with payload content, the packet being sent, the device waiting for acknowledgements, the device receiving the acknowledgments before returning to low-power mode. The device LoRaWAN activation process (OTAA) presented in section 2.2.1 is not visible

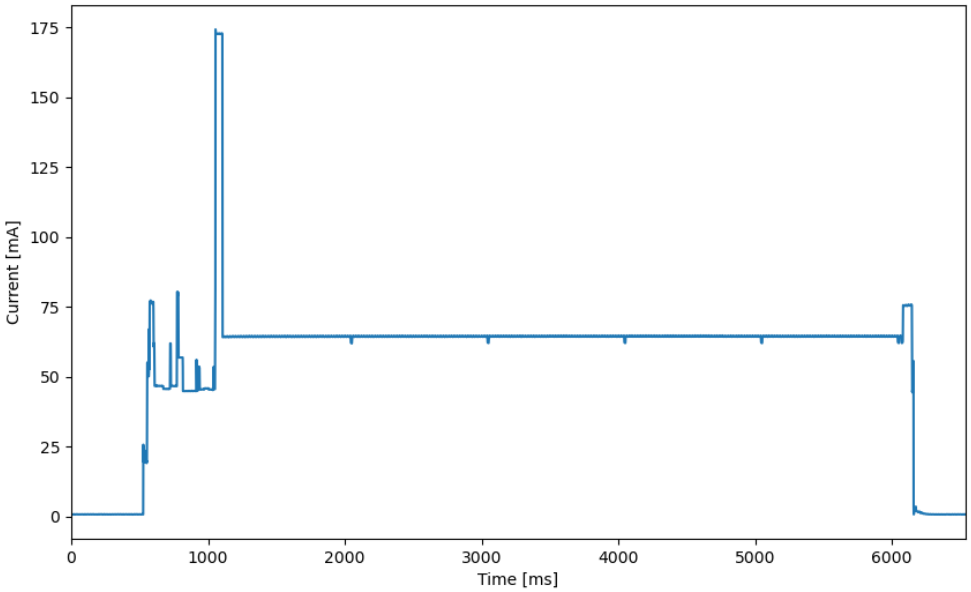


Figure 4.6: BME280 with LoRaWAN transmission

as its only made once during the first packet exchange with the server which make it irrelevant to the overall power consumption evaluation.

During the packet transmission, the current consumption increases to around 175mA for approximately 30ms. After this transmission, there is a 5-second period during which the current consumption remains constant at around 65mA, with short drops to 60mA every 1 second. This behavior is related to the LoRaWAN Class A protocol explained in section 2.2.1, once the uplink transmission is completed the device opens two short receive windows. There is a delay between the end of the uplink transmission and the start of the receive windows set to 5 seconds in this case. During this period the current level is maintained to keep the LoRaWAN module active, allowing it to receive any pending downlink messages or acknowledgements. The short drops in current consumption every 1 second might be related to the device checking for downlink transmissions.

The total time in active mode is 5636ms with an average current of 64.04mA resulting in a charge consumption of 0.1mAh.

Low-power mode

In this section, I analyze the power consumption of the device during low-power mode. The main objective of low-power mode is to save energy when the device is not performing active tasks, such as collecting data or transmitting packets. The device is in low-power mode more then 99% of the time which means that its power comsumption is crucial for the overall power consumption evaluation and battery lifetime estimation. In low-power mode, the CPU is paused by disabling its clock pulse, while only the RTC and ULP-coprocessor remain active[54].

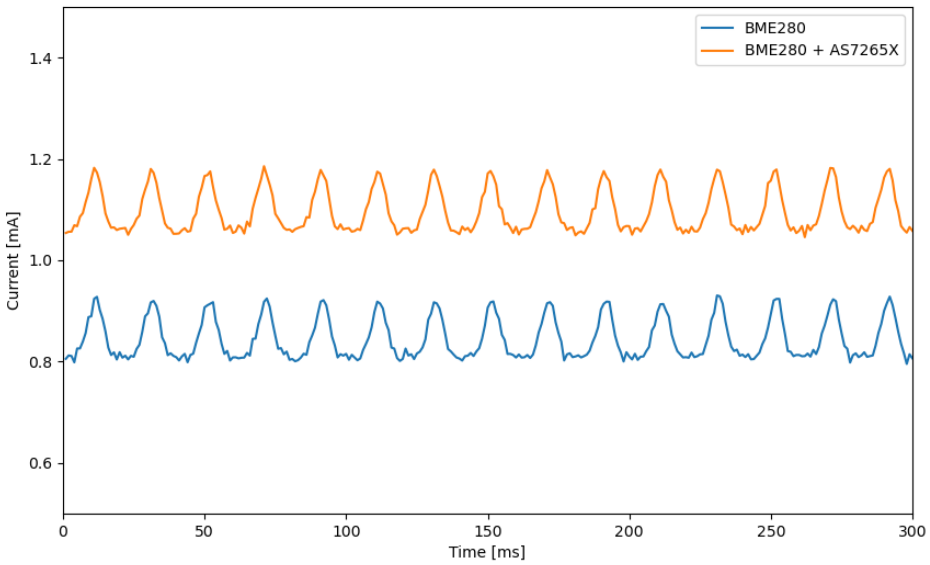


Figure 4.7: low-power mode consumption

When comparing the power consumption during low-power mode with different configurations, I observed that there was a difference in the average current consumption depending on the connected sensors. As shown on Figure 4.7, when only the BME280 sensor is connected, the average current consumption in low-power mode is 0.8mA, which is consistent with the values mentioned in the boards datasheet[55]. However, when the AS7265X spectral sensor is connected, the average current consumption in low-power mode increases to 1.1mA.

This difference in the average current consumption can be caused for different reasons. One possible explanation could be that the spectral sensor might have some internal leakage current when not powered up, which causes an overall increase in the current consumption. Another explanation could be linked to the sensor's circuitry,

which might draw a small amount of current even when it is not actively powered up, due to components like pull-up resistors or capacitors. These explanations are hypothetical, and further investigation should be made to identify the actual reason.

Impact of CPU frequency

In this section, I analyze the impact of different CPU frequencies on the power consumption of the device. The WiFi LoRa 32 supports three CPU frequencies: 80MHz, 160MHz, and 240MHz. To evaluate the impact of CPU frequency on power consumption, I conducted measurements with the device operating at each of these frequencies.

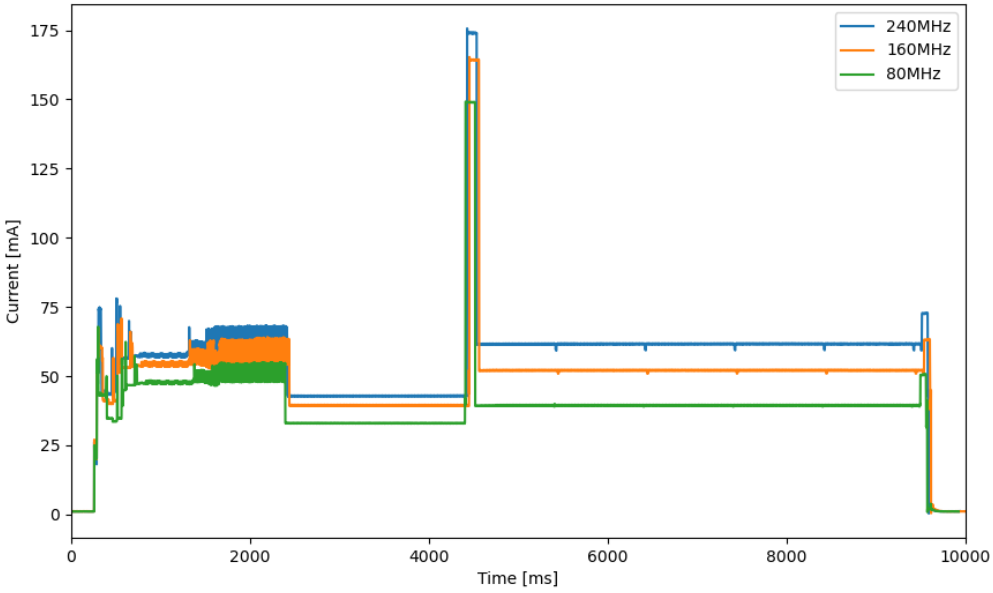


Figure 4.8: Power consumption with different CPU frequencies

Figure 4.8 show the power consumption of the device connected to the BME280 and AS7265X sensors, and LoRaWAN communication enabled, for each of the three CPU frequencies. It is important to note that the dataset contains an undesired 2-second delay, which will be addressed in the next subsubsection on overall power consumption. For now, let's focus on understanding the impact of CPU frequency on power consumption, keeping in mind that this delay will be removed in the final prototype.

Since the data where collected at different timings the unaligned plots makes it visually unclear for analysing, Table 4.2 summarizes the results:

CPU Frequency [MHz]	Active Period [ms]	Average Current [mA]
80	9315	41.2
160	9359	51.2
240	9332	58.04

Table 4.2: Power consumption with different CPU frequencies

As seen in Table 4.2, increasing the CPU frequency results in an increase in average current consumption. When the CPU frequency is increased from 80MHz to 160MHz, the average current consumption increases by 10mA. Further increasing the CPU frequency to 240MHz results in an additional increase of 6.84mA in average current consumption. This concurs with the fact that higher CPU frequencies require more power to operate, leading to higher overall power consumption.

Interestingly, the active period does not show a significant change with different CPU frequencies. This indicates that the increase in current consumption is not compensated by a proportional decrease in the active period, and thus, the overall power consumption is higher for higher CPU frequencies.

Figure 4.8 shows the impact of CPU frequency on the low-power mode consumption. The results indicate that the CPU frequency does not have a significant impact on the low-power mode consumption, with the average current remaining around 1.1mA in each case as discussed in the above section. Considering that the device spends 99% of its time in low-power mode, the impact of CPU frequency on power consumption during the active time becomes relatively less significant.

Overall power consumption

In this section, I provide an overview of the overall power consumption of the device by combining the results from the previous sections. This summary helps to better understand the relationship between different components and operating conditions, and their impact on the total power consumption of the device. The Figure 4.9 highlights the different events previously identified.

To summarize the results from the previous sections, the following points can be made:

- The AS7265X spectral sensor significantly impacts the overall power consumption which highlights the importance of selecting appropriate sensors and considering potential optimizations based on the specific application requirements.
- The LoRaWAN communication process results in a total charge consumption of 0.1mAh, with the majority of this consumption occurring during the 5-second

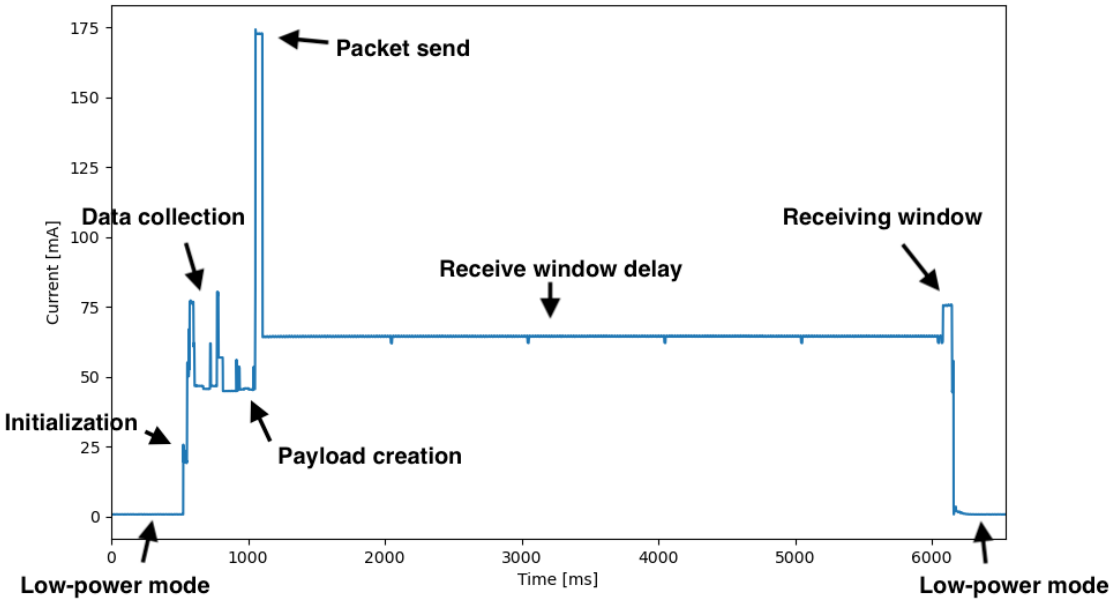


Figure 4.9: Power consumption with highlighted events

receive window period. This underlines the potential optimization of the LoRaWAN protocol configuration.

- The average current consumption in low-power mode is influenced by the connected sensors, with an increase in consumption from 0.8mA to 1.09mA when the AS7265X spectral sensor is connected. Understanding the cause of this increase is crucial for further optimizations and improving the overall power efficiency of the device.

- The CPU frequency has a noticeable impact on the average current consumption during active periods but does not significantly affect the low-power mode consumption. Given that the device spends 99% of its time in low-power mode, the impact of CPU frequency on the overall power consumption becomes relatively less significant. However, choosing a lower CPU frequency can still help minimize power consumption and prolong battery life.

As mentioned earlier, the dataset contains an undesired 2-second delay that impacts the overall power consumption. To provide an accurate estimation of the battery lifetime for the final prototype, this delay must be removed from the data. Once the delay is removed, the adjusted active periods and average current consumption values

can be used to calculate the overall power consumption and subsequently estimate the battery lifetime.

4.2.3 Battery lifetime estimation

In this section, I estimate the battery lifetime of the device based on the power consumption data presented in the previous section. To calculate the battery lifetime, I first need to determine the overall power consumption during the device's operation cycle, which includes both active and low-power periods. Then, I can use the battery capacity and the calculated power consumption to estimate the battery lifetime.

Assumptions

The assumptions made for the battery lifetime estimation are as follows:

- CPU frequency: 80MHz
- LoRaWAN: Enabled
- Sensors: BME280 and AS7265X
- Active mode duration: 7.315 seconds
- Low-power mode duration: 600 seconds (10 minutes)
- Active mode average current: 43.47mA
- Low-power mode average current: 1.1mA
- Battery capacity: 1100mAh

Results

Based on the aforementioned assumptions, the battery lifetime estimation can be calculated as follows:

Firstly, the percentages of time spent in active and low-power modes are calculated:

$$Active\ time\ percentage = \frac{7.315s}{7.315s + 600s} \times 100 \approx 1.2\%$$

$$Low - power\ time\ percentage = \frac{600s}{600s + 7.315s} \times 100 \approx 98.8\%$$

Subsequently, the average current consumption is calculated:

$$Average\ current\ consumption(mA) = \frac{(43.47mA \times 1.2\%) + (1.1mA \times 98.8\%)}{100} \approx 1.61mA$$

Finally, using a battery capacity of 1100mAh, the battery lifetime is estimated:

$$Estimated\ battery\ lifetime(h) = \frac{1100mAh}{1.61mA} \approx 683.23h$$

The estimated battery life is approximately 683.32 hours, corresponding to about 28 days. This estimation assumes constant operating conditions and does not take into account real-world factors such as battery self-discharge, temperature variations, and other environmental factors.

As previously mentioned, the spectral sensor has a significant impact on the power consumption, leading to an increase in both the low-power mode consumption and the active mode duration. After recalculating the power consumption data collected with the BME280 sensor and LoRaWAN enabled (CPU frequency of 240MHz), the overall average current consumption is approximately 0.67mA. This reduction in power consumption increases the estimated battery life to approximately 68 days, which is more than double the battery life with the spectral sensor.

All these battery lifetime estimations were made without considering the actual power supply architecture used in the final prototype, which includes a solar panel. The impact of the solar panel on the device's power consumption and battery life will be discussed in the following section.

4.2.4 Battery monitoring system implementation and considerations

In this section, I present the implementation and calibration of the battery monitoring system, as well as considerations related to its accuracy and power consumption. As explained in the previous chapter, the battery level is inferred from its output voltage, which is reduced using a voltage divider before being read by the board. The accuracy of this system is crucial for ensuring reliable operation of the device and making informed decisions about battery replacement or recharging.

To calibrate the battery monitoring system, a comparison was made between the analog value read by the device and the actual input voltage. I used the DC power supply used in previous experiments to select different input voltages and collect the corresponding analog values. Table 4.3 presents the collected data:

This data is used as calibration to calculate the battery percentage. First, the device uses this table to find the two voltage points between which the current analog value lies. It then calculates the voltage corresponding to the analog value by linear

Input voltage [V]	Analog value
4.2	2352
4.0	2250
3.7	2082
3.5	1959
3.35	1851
3.0	1631

Table 4.3: Measurements of analog value depending on the input voltage

interpolation between the two voltage points. This voltage is then converted to a percentage value based on the defined operating range of the battery.

This calibration procedure ensures that the device's battery monitoring system can accurately reflect the battery level, taking into account the specific characteristics of the voltage divider and the battery. However, it should be noted that this calibration is specific to the resistors used in the voltage divider and the battery's operating voltage range. If these parameters change, the calibration process will need to be repeated.

In addition to accuracy, it is essential to consider the power consumption of the voltage divider. A voltage divider continuously consumes power, as there is always current flowing through its resistors. This leads to a constant power drain on the battery. To minimize this drain, the resistors in the voltage divider should be chosen with relatively high resistance values, typically in the range of tens to hundreds of kilohms. This choice results in smaller current flow and lower power consumption. However, it is essential to balance the need for low power consumption with the requirements for accurate voltage measurements, as higher resistances can also lead to increased measurement noise and reduced accuracy.

The current flowing through the voltage divider (I) can be calculated using Ohm's Law, I = V / (R1 + R2). With a battery voltage of 4.2V (fully charged) and resistors of 100K Ohms each, the current I is approximately 21 µA (0.021mA).

4.3 Real-world deployment

To validate the functionality and resilience of the prototype in real-world conditions, an extended deployment experiment was conducted. The device was set up in the university's greenhouse and was left operational continuously for over a month. This prolonged test period served not only to verify the device's operational stability over time, but also to gain crucial insights into the performance of the power management system, especially the interplay between battery consumption and solar panel

efficiency.

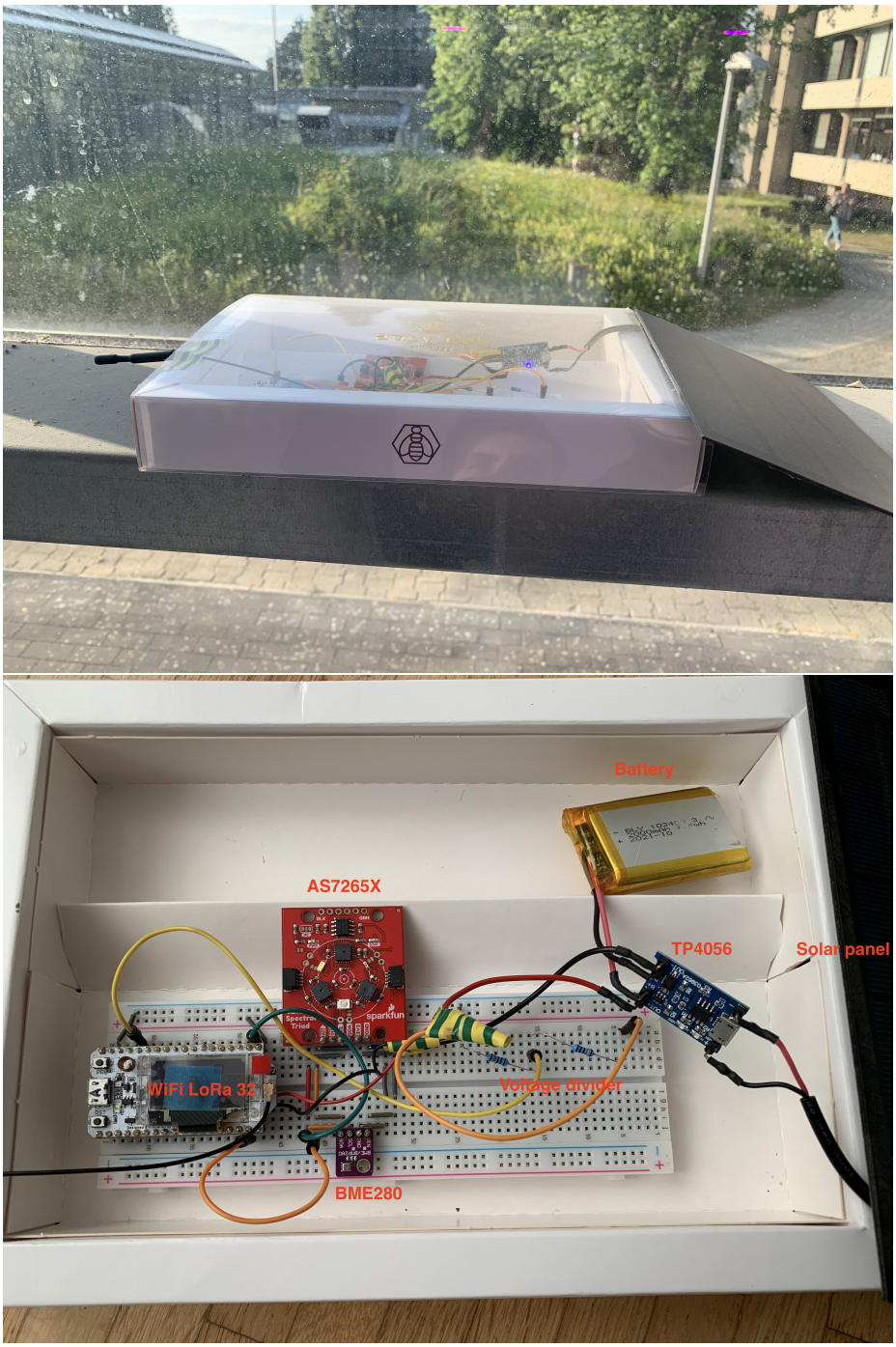


Figure 4.10: Prototype deployed in the greenhouse

Figure 4.10 depicts the prototype deployed in the greenhouse. As highlighted, the main components include the WiFi LoRa 32 (v2) board, the BME280 sensor, the AS7265X sensor, and the power system consisting of a rechargeable lithium battery and a solar panel connected to the TP4056 module. the voltage divider, using two 100K resistors, can also be observed. Despite its prototype-stage design using a breadboard for wiring, the device successfully collected and transmitted data over a month.

Data collection and visualization

The data collected during the extended deployment exhibited clear patterns that mirrored the diurnal cycle, thereby affirming the sensors' effectiveness in capturing and reflecting environmental conditions.

Figure 4.11 presents a basic example of a Grafana dashboard, which was utilized for data visualization. Grafana's versatile interface supports various visualization options, facilitating comprehensive trend analysis and correlation identification between different environmental parameters.

Detailed views of specific data, such as the battery level, can be achieved through individual dashboard panels.

Figure 4.12 illustrates the "Battery Evolution" panel, showing the battery level fluctuations over a week. The battery's regular discharge during the day and recharge cycle during periods of sunlight showcase the solar panel's effective performance. This pattern repeats on a daily basis, thereby ensuring a sustainable power supply for the prototype.

Similar diurnal patterns were observed in other data sets, including temperature, humidity, and spectral data, further validating the prototype's capability in real-world environmental monitoring.

Figure 4.11: Overview of the Grafana dashboard

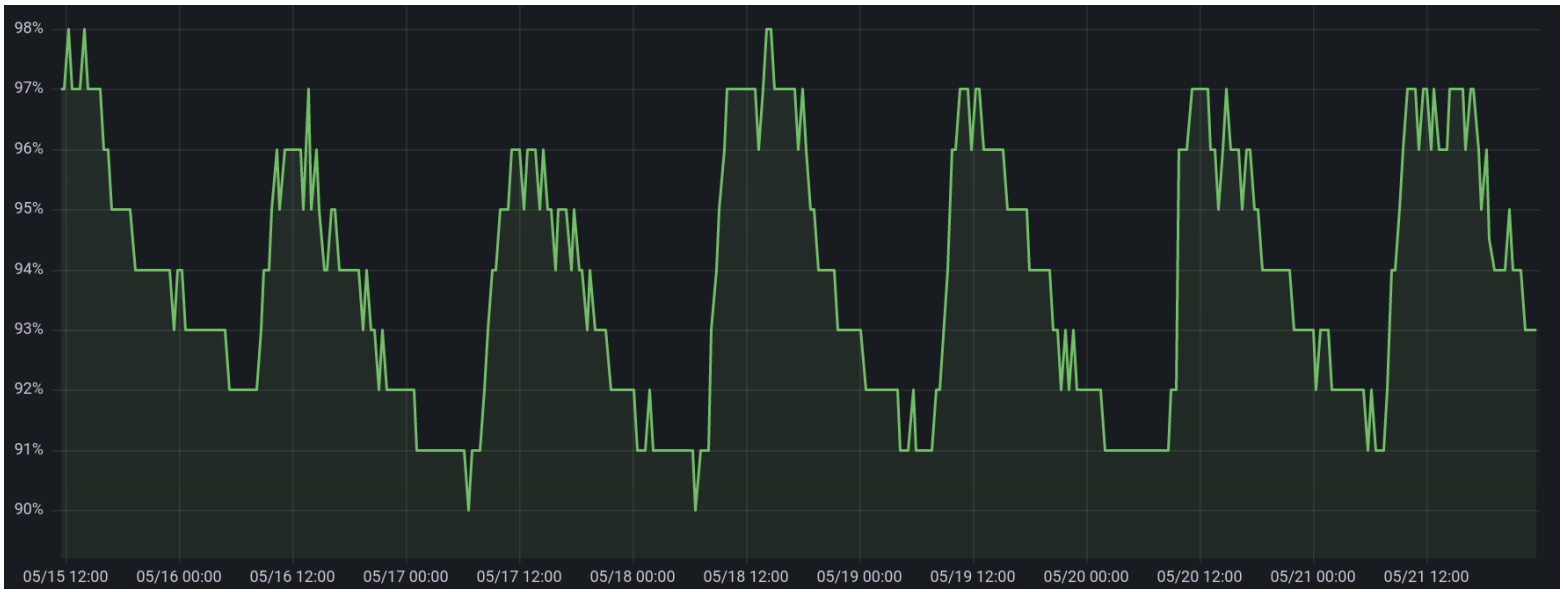


Figure 4.12: Zoomed in battery evolution panel

Chapter 5

Extending the prototype

In this chapter, I will discuss the challenges and considerations for the extension of the system. I will also discuss the limitations of the current prototype and propose possible improvements.

5.1 Challenges

The current prototype is equipped with two sensors, however, in order to capture a wider array of crucial greenhouse data such as CO2 levels, soil moisture, plant growth, among others, it may be necessary to integrate additional sensors. The incorporation of new sensors presents several challenges and requires thorough considerations.

Hardware compatibility and power requirements

One of the first steps when adding new sensors is to ensure hardware compatibility in terms of communication protocols and power requirements. Microcontrollers have limited communication pins. For sensors using the I2C protocol, they should have unique I2C addresses to share the same wires without interference. However, for those supporting only SPI or UART protocols, the number of additional sensors that can be integrated will be constrained due to specific wiring requirements per sensor, as discussed in section 2.5.

Power compatibility is another critical factor. The WiFi LoRa 32 board provides a maximum output of 3.3V, which must be considered when adding new sensors. For sensors that require a power supply exceeding 3.3V, one possible solution is to incorporate a step-up voltage regulator or a voltage booster. These components can take the 3.3V output from the board and step it up to a higher voltage according to the power requirements of the sensor. Furthermore, certain sensors require constant

power, significantly impacting the overall sustainability of the power supply system.

Software compatibility and data management

From a software perspective, the availability of compatible libraries for each sensor that supports the Arduino IDE is essential. While numerous libraries already exist, they might need adaptations for the specific dev-board used, as they are often contributed by different developers with a focus on specific boards.

After successfully integrating the sensor and being able to collect data, the packet format must be adapted. The collected data has to be converted into bytes in the Arduino sketch and added to the payload array. To ensure correct formatting, The Things Network (TTN) payload formatter must be updated. It should be noted that adding data to the payload influences the air-time of each uplink. This means that the number of uplinks might need to be adjusted to respect TTN's Fair Access Policy.

Backend adaptation

In the current system, the backend collects all data from the decoded payload. Therefore, it wouldn't necessitate changes when adding new sensors, simplifying the extension process to some extent.

5.2 Limitations and improvements

The following section provides an in-depth discussion on the limitations of the current prototype and proposes possible improvements to overcome these challenges.

Device management user interface

The current approach of modifying and re-uploading code to the microcontroller to change device settings is not very user-friendly. Developing a user interface, potentially web-based, would facilitate the adjustment of settings such as sleep time intervals, sensor activation, and devices information. This interface could further provide a live display of the sensor data, error logs, and the device's operational status, significantly improving user interaction with the system.

Integration of Additional Sensors

As explained above, the incorporation of new sensors into the current prototype can be challenging. Developing a generalized prototype that supports a predefined set of

sensors could be an ideal solution. This flexible system could automatically adjust its behavior based on the connected sensors and their configurations set through the user interface. Moreover, a plug-and-play system that automatically detects and configures newly added sensors would greatly simplify the extension of the system's capabilities.

Battery monitoring system

The current battery monitoring system, which uses a voltage divider circuit, induces a constant current draw, which, though relatively small, adds to the overall power consumption of the system. An improved solution could involve the use of a multiplexer (MUX) to control the connection between the voltage divider and the microcontroller.

Hardware

The prototype currently uses a breadboard for wiring and lacks a protective case, making it vulnerable to environmental conditions. Transitioning from a breadboard to a Printed Circuit Board (PCB) would provide a more robust and reliable wiring solution. Furthermore, designing a weatherproof case for the device would protect the electronics from external factors such as moisture, dust, and sunlight.

Main board

While the WiFi LoRa 32 (v2) board used in this project is effective, more recent boards may offer enhanced capabilities, such as better lower power consumption and more GPIO pins for connecting additional sensors.

Backend

The current backend system is highly functional but is optimized for local machine or Azure Cloud deployment. While this offers flexibility and scalability, it also entails ongoing operational costs. An improvement would be to consider deploying this system on the existing servers of the university, potentially reducing or eliminating these costs.

This would require an in-depth understanding of the university's server infrastructure, possibly necessitating adjustments to the Docker configurations to ensure compatibility. It's also important to consider factors such as data security, server load capacity, and network configurations. However, once implemented, this solution could lead to significant cost savings and enhanced data control, while still maintaining the advantages of a containerized solution.

Chapter 6

Conclusion

This master's thesis project was inspired by the transformative impact of the Internet of Things (IoT) technology across various domains, notably in agriculture. The project aimed to develop a prototype of an IoT-based greenhouse monitoring system.

The system, built around the WiFi LoRa 32 (v2) board and connected to BME280 and AS7265X sensors, collects environmental data, including temperature, humidity, pressure, light intensity, and battery level at regular intervals. Power is supplied by a combination of a lithium rechargeable battery and a solar panel, ensuring sustainable operation. The prototype efficiently sends data over LoRaWAN to The Things Network (TTN), which is then collected by the backend system composed of a Telegraf agent, an InfluxDB database, and a Grafana visualization application.

Through experiments and real-world deployment, the system proved its effectiveness. Data transmission was reliable, and the power supply system was self-sustaining, as the battery was capable of full recharging via the solar panel, thus demonstrating the system's potential for long-term deployment.

This project provided an invaluable opportunity to apply and integrate software, networking, and hardware skills that I gained during my academic study years, with the addition of significant learning in the hardware domain. The experimental component was particularly enlightening, revealing the necessity of practical testing to uncover unexpected behavior and optimize the system.

However, the current design of the system also has its limitations, providing several avenues for improvement. These range from developing a user-friendly interface for device management, enabling easy integration of additional sensors, revising the battery monitoring system to reduce constant current draw, upgrading the hardware and the main board, and considering deployment of the backend system on existing university servers. Each of these potential improvements represents a unique challenge, demanding further research and development.

In conclusion, this project has demonstrated the feasibility and benefits of an

IoT-based greenhouse monitoring system, providing a foundation for future research and development in this field. The experience gained and lessons learned during the execution of this project will undoubtedly prove valuable in future endeavours.

References

[1] Oracle. *What is IoT?* Accessed: 2023-05-24. 2023. URL: https://www.oracle.com/internet-of-things/what-is-iot/.

[2] Digital Directions. *Main Components of IoT*. Accessed: 2023-05-24. 2022. URL: https://digitaldirections.com/main-components-of-iot/.

[3] LoRa Alliance. *What is LoRaWAN® Specification*. Accessed: 2023-05-24. 2023. URL: https://lora-alliance.org/about-lorawan/.

[4] The Things Network. *What are LoRa and LoRaWAN?* Accessed: 2023-05-24. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/.

[5] The Things Network. *LoRaWAN Architecture*. Accessed: 2023-05-24. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/architecture/.

[6] LoRa Alliance. *RP002-1.0.3 LoRaWAN Regional Parameters*. Accessed: 2023-05-26. 2023. URL: https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/.

[7] European Telecommunications Standards Institute. *ETSI EN 300 220-1*. Accessed: 2023-05-26. 2017. URL: https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/03.01.01_60/en_30022001v030101p.pdf.

[8] The Things Network. *Message Types*. Accessed: 2023-04-11. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/message-types/.

[9] The Things Network. *Device Classes*. Accessed: 2023-04-11. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/classes/.

[10] The Things Network. *End Device Activation*. Accessed: 2023-05-26. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/.

[11] The Things Network. *Security*. Accessed: 2023-04-11. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/security/.

[12] Richard Wenner. *LoRa CHIRP*. Accessed: 2023-04-21. 2017. URL: https://www.youtube.com/watch?v=dxYY097QNs0&ab_channel=RichardWenner.

[13] Sigfox Build. *What is Sigfox 0G technology?* Accessed: 2023-05-28. 2023. URL: https://build.sigfox.com/sigfox.

[14] Daviteq. *Sigfox Technology – Make Things Come Alive*. Accessed: 2023-05-28. 2021. URL: https://www.daviteq.com/blog/en/sigfox-technology-make-things-come-alive/.

[15] Sigfox. *Get to know Sigfox*. Accessed: 2023-05-28. 2023. URL: https://build.sigfox.com/study#direct-cost.

[16] The Things Stack. *Console*. Accessed: 2023-05-15. 2023. URL: https://www.thethingsindustries.com/docs/the-things-stack/interact/console/.

[17] The Things Network. *Quick Start*. Accessed: 2023-05-15. 2023. URL: https://www.thethingsnetwork.org/docs/quick-start/.

[18] The Things Industries. *Pricing plans*. Accessed: 2023-05-26. 2023. URL: https://www.thethingsindustries.com/stack/plans/.

[19] The Things Network. *Adding Applications*. Accessed: 2023-05-24. 2023. URL: https://www.thethingsindustries.com/docs/integrations/adding-applications/.

[20] The Things Network. *Adding Devices*. Accessed: 2023-05-24. 2023. URL: https://www.thethingsindustries.com/docs/devices/adding-devices/.

[21] The Things Network. *Payload Formatters*. Accessed: 2023-05-26. 2023. URL: https://www.thethingsindustries.com/docs/integrations/payload-formatters/.

[22] The Things Network. *Adding Integrations*. Accessed: 2023-05-24. 2023. URL: https://www.thethingsindustries.com/docs/integrations/adding-integrations/.

[23] The Things Network. *Fair Use Policy*. Accessed: 2023-05-26. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/.

[24] Arjan. *Airtime calculator for LoRaWAN*. Accessed: 2023-05-26. 2023. URL: https://avbentem.github.io/airtime-calculator/ttn/eu868/.

[25] Docker. *Docker overview*. Accessed: 2023-05-24. 2023. URL: https://docs.docker.com/get-started/overview/.

[26] Microsoft. *What is Azure?* Accessed: 2023-04-26. 2023. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/.

[27] Microsoft. *Manage Azure resource groups by using the Azure portal*. Accessed: 2023-04-26. 2023. URL: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal.

[28] Microsoft. *What is Azure Container Instances?* Accessed: 2023-04-26. 2023. URL: https://learn.microsoft.com/en-us/azure/container-instances/container-instances-overview.

[29] Docker. *Deploying Docker containers on Azure*. Accessed: 2023-04-26. 2023. URL: https://docs.docker.com/cloud/aci-integration/.

[30] Influxdata. *Telegraf*. Accessed: 2023-05-24. 2023. URL: https://www.influxdata.com/time-series-platform/telegraf/.

[31] *MQTT*. Accessed: 2023-05-13. 2022. URL: https://mqtt.org/.

[32] InfluxData. *Introducing InfluxDB 3.0*. Accessed: 2023-05-24. 2023. URL: https://www.influxdata.com/products/influxdb-overview/.

[33] InfluxData. *Introduction to InfluxDB*. Accessed: 2023-05-24. 2023. URL: https://awesome.influxdata.com/docs/part-1/introduction-to-influxdb/.

[34] InfluxData. *Time series database (TSDB) explained*. Accessed: 2023-05-24. 2023. URL: https://www.influxdata.com/time-series-database/.

[35] J. Walker. *What is Grafana and When Should You Use It?* Accessed: 2023-05-24. 2022. URL: https://www.howtogeek.com/devops/what-is-grafana-and-when-should-you-use-it/.

[36] Chiara Bersani et al. "Internet of Things Approaches for Monitoring and Control of Smart Greenhouses in Industry 4.0". In: *Energies* 15.10 (2022), p. 3834. DOI: 10.3390/en15103834.

[37] Shablu Nath et al. "Design and Implementation of an IoT Based Greenhouse Monitoring and Controlling System". In: *Journal of Computer Science and Technology Studies* 3 (Jan. 2021). DOI: 10.32996/jcsts.2021.3.1.1.

[38] Hassan Ibrahim et al. "A layered IoT architecture for greenhouse monitoring and remote control". In: *SN Applied Sciences* 1.3 (2019), p. 223. ISSN: 2523-3971. DOI: 10.1007/s42452-019-0227-8.

[39] Redmond R. Shamshiri et al. "Greenhouse Automation Using Wireless Sensors and IoT Instruments Integrated with Artificial Intelligence". In: *Next-Generation Greenhouses for Food Security*. Ed. by Redmond R. Shamshiri. Rijeka: IntechOpen, 2021. Chap. 1. DOI: 10.5772/intechopen.97714.

[40] Robin Schoenmaeckers. "Evaluating LoRa and LoRaWAN performance in Louvain-la-Neuve". In: (2019).

[41] Nicolas Van De Walle François De Keersmaeker. "Using LoRaWAN and Wi-Fi for smart city monitoring in Louvain-la-Neuve". In: (2021).

[42] Hadrien Libioulle Olivier Latteur. “Louvain-la-Neuve, a Smart City: a LoRa based and LTE assisted bike tracker”. In: (2022).

[43] Heltec Automation. *WiFi LoRa 32 (V2)*. Accessed: 2023-05-24. 2018. URL: https://heltec.org/project/wifi-lora-32/.

[44] Heltec Automation. *Heltec Automation*. Accessed: 2023-05-24. 2023. URL: https://heltec.org/.

[45] Semtech. *SX1276 Datasheet*. Accessed: 2023-05-24. 2015. URL: https://www.alldatasheet.com/datasheet-pdf/pdf/800239/SEMTECH/SX1276.html.

[46] Espressif Systems. *ESP32 Series Datasheet*. Accessed: 2023-05-10. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[47] Heltec Automation. *CubeCell – Dev-Board*. Accessed: 2023-03-15. 2023. URL: https://heltec.org/project/htcc-ab01/.

[48] Bosch Sensortec. *BME280*. Accessed: 2023-05-24. 2023. URL: https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/.

[49] Sparkfun. *Triad Spectroscopy Sensor - AS7265x*. Accessed: 2023-02-10. 2023. URL: https://www.sparkfun.com/products/15050.

[50] NanJing Yop Power ASIC Corp. *TP4056 datasheet*. Accessed: 2023-04-25. URL: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf.

[51] Wikipedia. *Voltage divider*. Accessed: 2023-04-13. 2023. URL: https://en.wikipedia.org/wiki/Voltage_divider.

[52] Heltec Automation. *WiFi LoRa 32(V2) pinout diagram*. Accessed: 2022-11-10. 2023. URL: https://resource.heltec.cn/download/WiFi_LoRa_32/WIFI_LoRa_32_V2.pdf.

[53] The Things Network. *RSSI and SNR*. Accessed: 2023-05-24. 2023. URL: https://www.thethingsnetwork.org/docs/lorawan/rssi-and-snr/.

[54] LastMinuteEngineers. *Insight Into ESP32 Sleep Modes Their Power Consumption*. Accessed: 2023-05-09. 2020. URL: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/.

[55] Heltec Automation. *WiFi LoRa 32 (V2) Datasheet*. Accessed: 2023-04-05. 2020. URL: https://resource.heltec.cn/download/Manual%20Old/WiFi%20Lora32Manual.pdf.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**