Universidade do Minho
Escola de Engenharia

Micael Bruno da Silva Coutinho

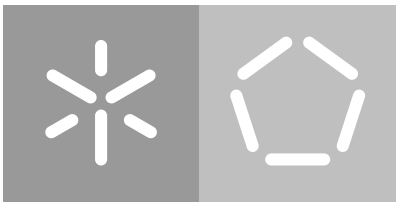
Irrigation Planning System for Agricultural Soils

Fevereiro de 2022

Irrigation Planning System for Agricultural Soils

Micael Bruno da Silva Coutinho

Uminho | 2022



Universidade do Minho
Escola de Engenharia
Departamento de Eletrónica Industrial

Micael Bruno da Silva Coutinho

Irrigation Planning System for Agricultural Soils

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação dos professores
Sérgio Adriano Fernandes Lopes, Luis Miguel Valente Gonçalves

Fevereiro de 2022

Acknowledgements

Firstly, I would like to show my gratitude to my advisors for their guidance and support during this project, and in particular to Professor Doctor Sérgio Lopes for the insightful discussions and dedication over the course of this work. I also would like to thank everyone involved in the research project 02/SAICT/2017-28247-FCT-TO-CHAIR, that supported the work developed in this dissertation.

Then, I want to thank my close relatives for the moral support provided over the course of this work, specially to my parents Paulo and Maria for their sacrifices and putting me first, my sister Magda for being always present and for showing me the value of education and wisdom, my grandparents, Alice and Ezequiel, who unfortunately is not with us anymore, but played a crucial role in my life.

I also intend to express my appreciation for my colleagues and closest friends, in particular to Pedro and Mafalda, who always were there to celebrate during great times and support during the bad ones.

Lastly, I want to express my gratitude towards everyone who, directly or indirectly, played a part in this dissertation, in what has been a very difficult year for everyone.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

O objetivo principal desta dissertação é o desenvolvimento de um sistema de planeamento de regas para a agricultura. Este projeto dá seguimento ao algoritmo desenvolvido em [1], capaz de criar um plano de rega de acordo com informações relativas ao solo, colheita, tipo de irrigação, humidade do solo e a previsão meteorológica de uma determinada localização.

O sistema desenvolvido é composto por uma aplicação web e uma rede de dispositivos eletrónicos no campo. O sistema efetua todo o trabalho necessário, desde a aquisição de dados relativos à humidade do solo até à exibição dos planos de rega ao agricultor.

A aplicação web utiliza a stack tecnológica MERN para fornecer uma interface ao utilizador, onde é possível gerir os pontos de rega e campos agrícolas, observar previsões meteorológicas e visualizar e obter atualizações relativas aos planos de rega, assim como alertar o agricultor através de notificações push quando condições alarmantes se verificam. Para além da interface com o utilizador, esta também obtém informações meteorológicas, executa o algoritmo de planeamento e agrega os dados de humidade do solo recolhidos pela rede de pontos de rega, através de um servidor CoAP.

A rede de dispositivos eletrónicos no campo está encarregue de recolher informação relativa à humidade do solo e enviá-la para o servidor de hora a hora, recorrendo a diferentes tecnologias de forma a proporcionar uma solução flexível de baixo custo, com duas possibilidades de configuração, standalone e WSN, adequadas para diferentes cenários. A comunicação entre os dispositivos no campo e o servidor é baseada no protocolo CoAP.

A configuração standalone é constituída por uma PCB, que combina um microntrolador low power com um circuito de energy harvesting. A esta, são conectados um painel solar, um conversor step-up, uma bateria Li-Po e um módulo de comunicações móveis (capaz de utilizar as tecnologias móveis GPRS/UMTS e NB-IoT), assim como até seis sensores de humidade do solo.

A configuração WSN recorre à mesma PCB que a configuração standalone, utilizando um transceiver LoRa em vez do módulo de comunicações móveis. Esta comunica através da camada física LoRa com um edge device baseado na plataforma Raspberry Pi, que encaminha os pacotes recebidos pela rede LoRa através do protocolo CoAP para o servidor. A rede LoRa desenvolvida é capaz de enviar mensagens downlink diárias e um data-rate adaptativo, que controla o link budget através do spreading factor e da potência de transmissão, recebendo pacotes recorrendo a um esquema adaptativo de seleção do spreading factor (ASFS) [2].

Palavras chave - Irrigação, WSN, Web

Abstract

The main objective of this dissertation is the development of an irrigation planning system for agriculture. This work builds upon the algorithm developed in [1], capable of creating an irrigation plan according to soil, crop, irrigation, soil moisture and weather forecast of a given location.

The developed system is composed by a web application and a network of field electronic devices. The system does all the necessary work, from the retrieval of the soil moisture data to the display of irrigation prescription plans to the farmer.

The web application resorts to the MERN technological stack to provide an interface to the farmer, where irrigation points and crop fields can be managed, forecasts observed and the irrigation plans can be retrieved and updated, while also alerting the farmer through push notifications when dangerous conditions are verified. Besides the interface with the farmer, it also gathers weather information, performs the irrigation planning and retrieves soil moisture data from the irrigation points through a CoAP server.

The network of electronic devices is in charge of retrieving soil moisture information and sending it to the server on an hourly basis, using different technologies to provide a flexible low-cost solution with two different configurations, standalone and WSN, suitable to many different scenarios. The communication between field devices and the server is based on CoAP protocol.

The standalone configuration consists of a PCB, where a low power microcontroller is paired with an energy harvesting circuit. To it, a solar panel, a step-up converter, Li-Po battery and a cellular communication module (capable of connectivity with both GPRS/UMTS and NB-IoT technologies) are connected, along with up to six soil moisture sensors.

The WSN configuration makes use of the same PCB as the standalone configuration, using a LoRa transceiver instead. It communicates through the LoRa physical layer to an edge device based on the Raspberry Pi platform, which forwards the packets received from the LoRa network through CoAP to the web server. The LoRa network developed is capable of daily downlink messages and adaptive data-rate, where the link budget is controlled through the spreading factor and the transmission power, receiving packets through an adaptive spreading factor selection (ASFS) scheme [2].

Keywords - Irrigation, WSN, Web

Contents

List of Acronyms

C

CAD Channel Activity Detection.

COAP Constrained Application Protocol.

G

GIS Geographical Information System.

I

IOT Internet of Things.

J

JSON JavaScript Object Notation.

M

MAC Medium Access Protocol.

MERN MongoDB Express React Node.

O

OSI Open System Interconnection.

R

REST Representational State Transfer.

RTOS Real Time Operating System.

T

TOA Time On Air.

U

UART Universal Asynchronous Receiver Transmitter.

List of Figures

List of Tables

1. Introduction

Water scarcity is a big environmental concern, with efforts being made worldwide to minimize the impacts and guarantee a more sustainable future of it. Agriculture is one of the most important activities in the world, but it accounts for the use of a lot of water resources. Minimizing water consumption is a priority. Proper irrigation planning can improve crop yield, while using the water resources efficiently.

This work addresses the problem by providing the farmer with an irrigation planning system, prescribing the right amount of water so that crop yield and water usage are not compromised. It takes into account the conditions of the environment, such as weather forecasts (temperature, humidity, wind and precipitation) and the location of the crops (distance to the coast, altitude and soil type). Besides, it also considers some conditions specific to the farmer, including the crop type, irrigation method and soil moisture. Lastly, the system is practical to use and easily accessible.

1.1 Motivation

The main motivation behind this work is the innovative irrigation algorithm, which applies optimal control theory to a mathematical model in order to maximize the efficiency in the use of water, providing the minimum amount of water necessary, while keeping the crops safe. In order to make the algorithm useful in the daily life, it needs a system to support it, capable of retrieving soil moisture information in the crop fields, meteorological data and other important variables. It is also necessary to provide a friendly platform, where the results from the model can be delivered to the user.

Aside from the mathematical model, there is the need to provide a flexible solution, suitable from the small farmer to the massive crop producer. The developed system should be usable in both scenarios. Due to the variability in the irrigation facilities present in crop fields, it is not viable to develop an automatic actuation solution that could easily be adapted to all of them. If the actuation task is left on the hands of the farmer, the irrigation plans calculated by the model can be used by both small farmers, that may not have an automatic irrigation system and also by farmers that do not wish, or cannot afford, to install a new one.

Usually, the farmer has to deal with crop and soil variability within the crop fields. The system should take into account all the possible scenarios and be adaptable enough to suit the farmers' needs. Provided enough flexibility in the deployment of nodes in the crop fields, the user is able to decide on how many and where the nodes are placed and what crops are cultured, making the system easier to understand and use.

1.2 Approach

In order to bring the solution stated before to life, this work requires some building blocks, namely, a user interface, electronic devices to measure soil moisture, and the necessary software inherent to the UI, data acquisition and communication.

Regarding the user interface, its main goal is being easy to use. It should not limit the user to a single platform. Hence, its design has to take into consideration both desktop and mobile devices and adapt its behaviour to best-suit the platform in use by the farmer. The best way to achieve this, balancing development effort with practicality, is to develop a web-based user interface.

When it comes to the electronic field devices, they should be as close to plug-and-play as possible. This will allow them to be easily configured by anyone with little knowledge of electronics or the need for extra tools.

The data acquisition process has to take place with no user intervention. Hence, the devices should withstand reasonable periods of data acquisition without the need for a user, to, for instance, recharge the battery.

Lastly, these devices are the main source of information for the system. So, the communication technology used should not pose as a limitation. The main parameters to take into consideration are the range and energy spent while communicating.

1.3 Document Structure

This dissertation is separated into 7 chapters. The following chapter presents the state of the art on irrigation monitoring and planning. Chapters 3 to 5 describe the system development following the waterfall methodology, respectively the stages of analysis, design and implementation. In the last chapter, conclusions and future improvements over the developed system are reported.

2. State of the Art

This chapter is structured in three sections: firstly, the overall picture of the existing solutions related to the problem are addressed. It is followed by a description of the technologies used in the developed work, in order to provide insight into the upcoming chapters. Lastly, articles and products related to the developed system are described.

2.1 Overview

There are multiple studies related to the monitoring/planning of the irrigation in agricultural soils, as well as interesting solutions on the market. They can be analysed in multiple fronts, such as the sensor characteristics, the building blocks of the WSN and technologies applied in it, how the planning or actuation of the irrigation takes place, the system autonomy, the cost, how hard it is to use and what kind of interaction is available to the farmer. Even though some of these metrics are closely related, they can also be addressed individually.

Regarding the soil moisture sensing (Fig.1), it is mainly determined in one of two ways: the Volumetric Water Content (VWC) or the water tension in the soil, according to the type of sensor: volumetric or solid-state.

The volumetric sensors directly measure the water content in the soil, mostly through resistance or dielectric constant of the soil, where the resistive sensors are more prone to corrosion, since the probes are in direct contact with the soil, causing electrolysis of the sensors overtime because there is a DC current flowing. The resistive sensors measure the ions that are dissolved in the water, being affected by external factors such as the use of fertilizer. Since the capacitive sensors measure the dielectric formed by the water and the soil, they provide greater immunity to these extreme factors.

The solid state sensors use two electrodes to measure the electrical resistance in the soil. They fall behind to the volumetric sensors because they are slower to measure field conditions and do not work accurately in highly saline soils.

The measurements from the sensors are mainly provided in one of two forms: a variable voltage or the SDI-12 protocol, widely used alongside data loggers in agriculture applications. Lastly, recent studies have been carried out in order to utilize communication indicators such as RSSI (Received Signal Strength Indication) to measure soil moisture [3].

With respect to the sensor network (Fig.2), its main building blocks are the end nodes, which, besides measuring soil moisture and other useful variables, have communication capabilities and low power concerns, possessing sometimes energy harvesting techniques, in order to maximize system

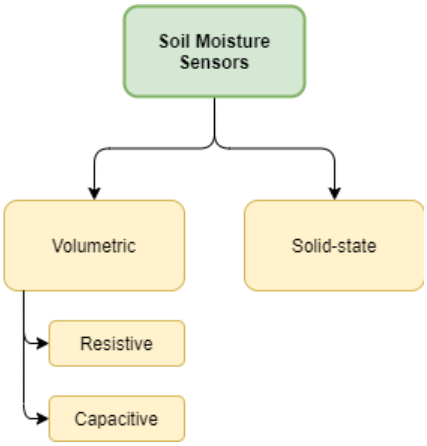


Figure 1: Most popular types of soil moisture sensors

autonomy [4]–[16]. Some end nodes also integrate actuators [8], [9], [13], [17]–[25], performing the automatic irrigation of the crop field, discarding the need of user intervention. However, they usually imply the replacement of the irrigation systems that may be deployed in the crop field. The end nodes can forward the sensed information directly to a server on the internet, through GPRS/UMTS technology or to an edge device/gateway, which acts as a data link between the nodes deployed in the crop field and the internet, such as the ZigBee and LoRa network technologies, referred later in this section. Other frequently used are relay nodes [4], [26], whose main task is to route the information from the end nodes to other relay nodes or the edge device, in order to achieve larger area coverage. In parallel, the solutions based on data loggers can forward the information retrieved from soil moisture sensors directly to the internet or provide a local user interface to gather data and visualize it.

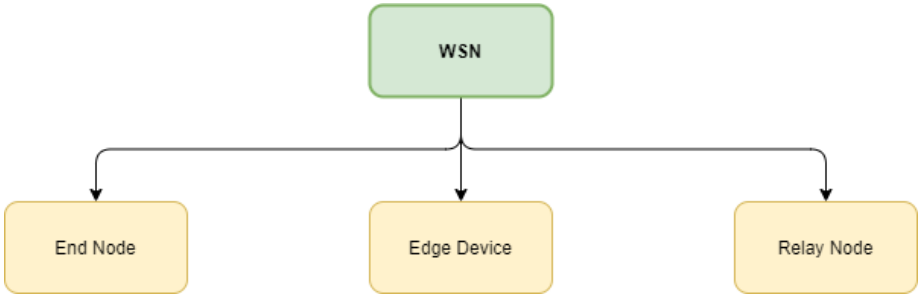


Figure 2: Main building blocks of a soil moisture WSN

Another important factor is the network technology (Fig.3). Several systems make use of ZigBee [4], [6], [7], [9], [10], [12]–[14], [19], [21]–[25], [27]–[34]. The main obstacle of this technology is the communication range (100m up to 1km, forcing in many cases the utilization of complex network topologies and relay nodes, to make the network coverage acceptable. These techniques make the network more interdependent and add points of failure. Recently, due to the rising popularity of

LPWAN networks, many systems started integrating LoRa [18], [26], [35]–[37]. This technology is tailored for low power applications, with low data transmission rates and relatively long range (approximately 10km in rural areas), while also making use of ISM bands, as ZigBee does. It makes use of a gateway to connect the network to the internet. Even though there are many of them on the market that obey to the LoRa specification, they are expensive. For private deployments, where the node density is not too big, alternative solutions can be explored, reducing the necessary hardware and consequently the cost [38]. Other frequently adopted network technology is Wi-Fi [8], [11], [39], which also suffers from the same communication range issues as ZigBee. At the edge device level, or in the case of end nodes that communicate directly with a web server, the technology mainly used is GPRS/UMTS [5], [10], [12], [13], [15], [37], [40], [41], where the superior QoS (Quality of Service) requirements contrast with additional fees, due to the use of licensed bands.

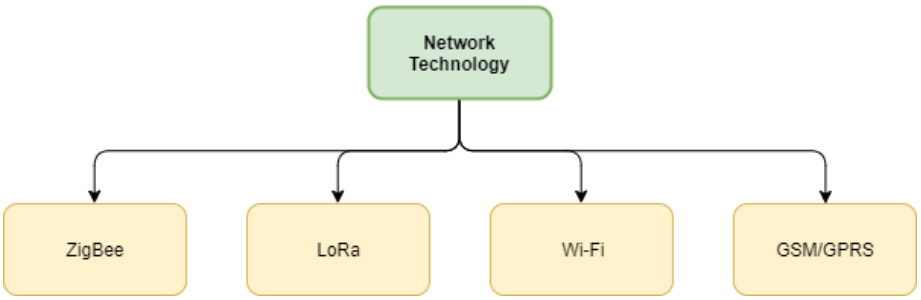


Figure 3: Popular network technologies for PA applications

Regarding the interaction with the farmer (Fig.4), it is mainly carried out through mobile or web applications, where the latter can be supported by already existing platforms to provide the retrieved information from crop fields [17], [30], [39]. Generally, these applications supply plots regarding the soil moisture evolution in each end node, usually accompanied by geographical information (GIS systems), mapping the soil moisture in a given area [5], [6], [29], [30], [36], [40], [41]. They can also provide irrigation prescription plans and a means for the user to control irrigation valves, when the end nodes have actuation capabilities. Lastly, some applications alert the user through SMS messages when alarming conditions are verified [16], [21], [22], [41], [42].

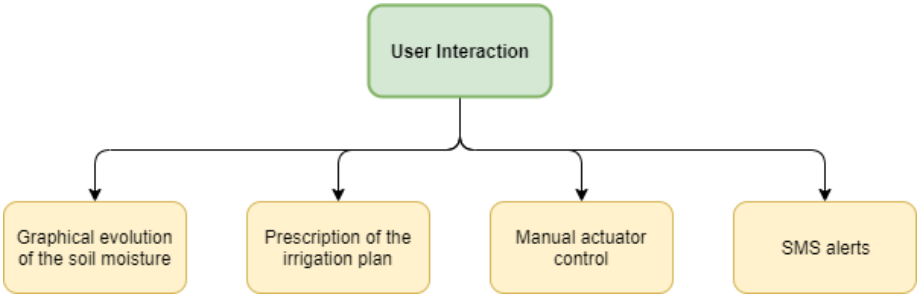


Figure 4: Common forms of interaction between the farmer and PA systems

Relatively to the monitoring of the soil irrigation (Fig.5), there are many options. The simplest is based on soil moisture thresholds [24], [27], [29], [41], which in many scenarios is not ideal. Following, there are fuzzy logic based systems [9], [20], which can yield positive results, considering their simplicity. One of the most relevant options is the use of mathematical models [5], [8], [21], [35]. Being a physical simplification of the reality, they are more precise than the aforementioned. Lastly, recent studies started making use of machine learning models. Even though they are highly dependent on the information used for model training in order to yield acceptable results, their applications in precision agriculture scenarios are endless, such as in the detection of crop diseases and in the optimization of fertilizer usage.

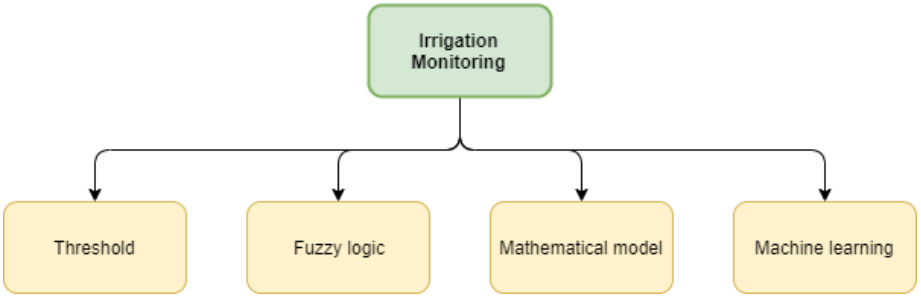


Figure 5: Irrigation monitoring in PA applications

2.2 Main Technologies Used

2.2.1 LoRa

Covering the physical layer of the OSI model, LoRa makes use of the free ISM sub-gigahertz bands to enable long range and low power communication of up to 10 km in rural areas.

LoRa is a spread spectrum modulation technique based on CSS technology, developed by Semtech. LoRa modulation (see Fig.6) is carried out by representing each bit in multiple chirps of information. The rate at which each symbol of information is sent is referred to as the symbol rate, and it relates to the bandwidth or chip rate through the spreading factor, the number of symbols that represent each bit of information, as given by $2^{SF} = BW/Rs$, where BW is the bandwidth and Rs is the symbol rate. A higher SF, or sweeping factor results in a longer time on air but in a better SNR, achieving longer communication distances. Each SF step (which can vary from six to twelve) doubles the duration of each symbol, making it easier to decode, but reduces the bit rate approximately by half and doubles the ToA (Time on Air).

A LoRa packet (Fig.7) is comprised by a preamble, used to synchronize the receiver with the incoming data flow. Depending on the mode of operation, a header may or may not be sent. It contains the payload size, a forward error correction code rate and specifies the presence of an

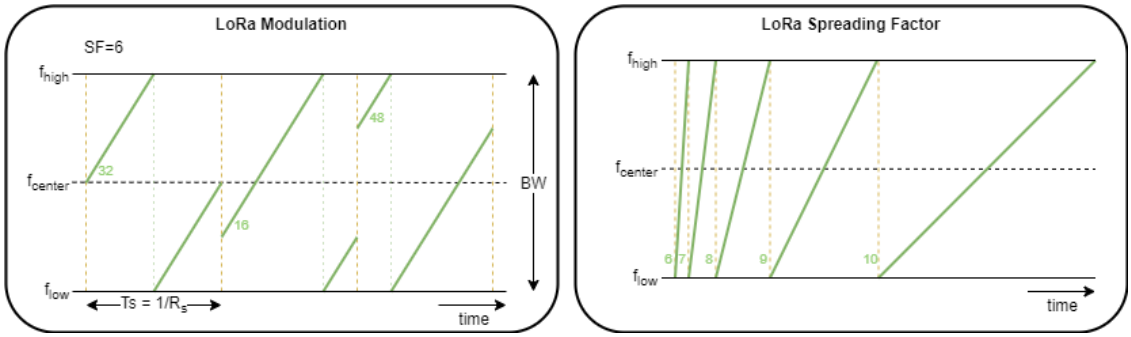


Figure 6: LoRa modulation example (left) and spreading factor (right)

optional CRC for the payload. In the implicit header mode, the receiver knows in advance these values. Then, the payload and the optional CRC are what follows.

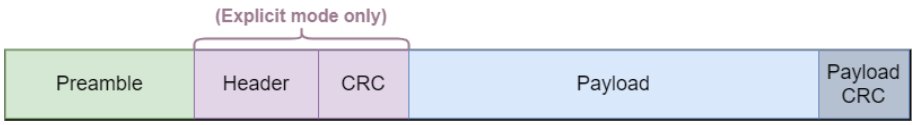


Figure 7: LoRa message format

The LoRa physical layer can be extended by the LoRaWAN network (Fig.8), covering the MAC and network layers. Its network structure is a star of stars, comprised by four components:

- **End Node** - Low power devices capable of LoRa communication. LoRaWAN supports three different classes of end nodes, to address the needs of a wide range of applications. The data sent by an end node is received by every gateway in range;
- **Gateway** - Responsible for capturing the end node packets and forwarding them to the network server. Its hardware capabilities enable it to communicate over multiple channels and spreading factors at the same time, accommodating up to 10000 end nodes. Its high performance is accompanied by a hefty price tag;
- **Network Server** - Performs the filtering of duplicate packets (that may arrive from multiple gateways), security checks and decryption, manages the network (including the data rate of the end nodes) and sends the data to an application server;
- **Application Server** - Provides a means for the user to access the data retrieved by the end nodes.

The LoRaWAN packet (Fig.9 builds upon the physical layer payload (Fig.7 with a MAC header, indicating the message type and LoRaWAN version. It is followed by the frame header, containing a 32-bit short device address, frame control flags (which include ADR control mechanisms by the network

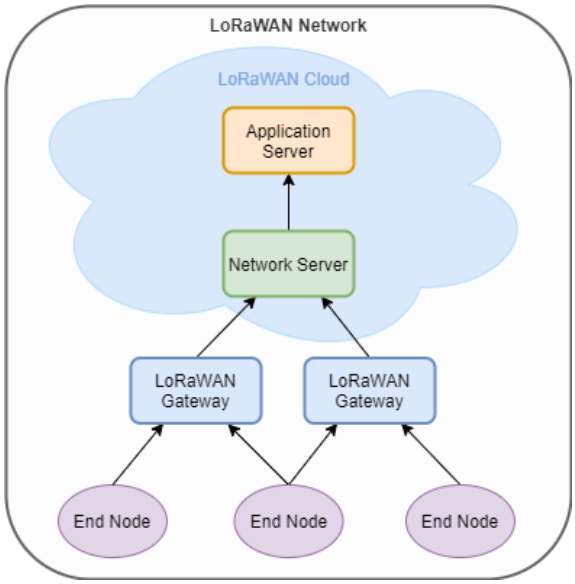


Figure 8: LoRaWAN network structure

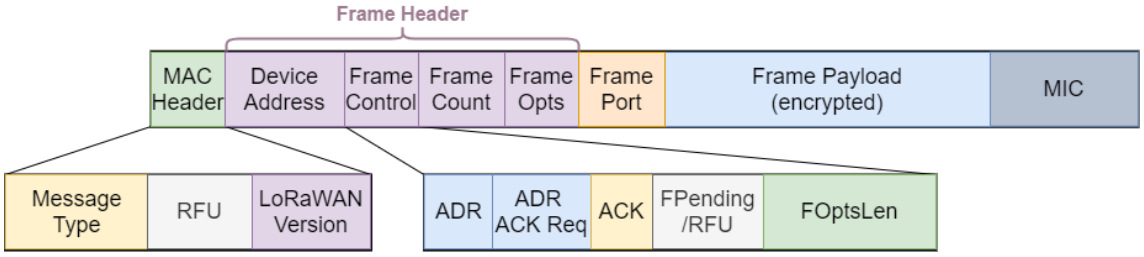


Figure 9: LoRaWAN message format

server and the acknowledge), frame count and frame options, used to piggyback MAC commands to a message. Before the payload, the multiplexing port field is sent. Lastly, the Message Integrity Code is sent.

The LoRaWAN network has a lot to offer, but not for free: aside from the monthly subscription to a LoRa public operator, which can become very expensive if a large number of LoRa nodes are deployed, LoRaWAN gateways can accommodate a large portion of the network deployment costs. Cost conservative alternatives to LoRaWAN usually involve replacing the gateway with a low performance router, by using end device hardware. The main limitation of this approach is the capacity to only receive a packet at a time, at a given frequency channel and spreading factor. Aside from the network limitations, this restrictive configuration has energy implications on the end nodes and causes more interference, by spending unnecessary ToA. In LoRaWAN, ADR mechanisms are adopted in static end nodes to control data rates, air time and energy consumption, by controlling the SNR for the received packets. Recent studies started to take into consideration these parameters and delivered

interesting network management solutions for low cost routers, by enabling the LoRa communication on different spreading factors [2].

2.2.2 CoAP and MQTT-SN

The preferred application layer protocol to provide web services is usually HTTP, but due to its high computation complexity, communication overhead and energy consumption it is not suitable for constrained devices. Therefore, IETF developed several lightweight protocols, including CoAP. It is intended to be used and considered as a replacement for the HTTP protocol on constrained IoT environments.

Being focused on the needs of constrained devices, CoAP protocol packets are much smaller and are easier to parse than its HTTP counterpart. The protocol follows a client/server model, where the clients are provided URI REST methods such as GET, PUT, POST and DELETE, while also extending the HTTP request model with the ability to observe a resource. CoAP runs over UDP, defining retransmission and resource discovery mechanisms to compensate its unreliability. It also allows UDP broadcast and multicast for addressing. Regarding application level QoS, four message types are supported, including confirmable messages, requiring an acknowledge packet to be sent by the receiver. CoAP messages (Fig. 10) are composed by a fixed four byte header, following by the optional token and options, and the payload. Regarding security, CoAP is paired with DTLS, the same security protocol used in UDP communication.

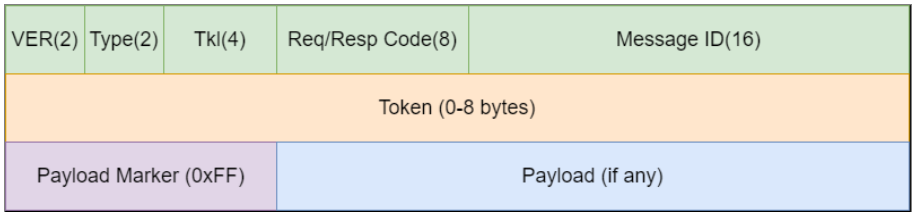


Figure 10: CoAP message format

Other promising protocol for IoT devices is MQTT-SN, a sensor network oriented variant of the popular MQTT, designed to work in the same way. Its architecture (Fig. 11) consists of four components: a client, a gateway, which can be aggregating (all MQTT-SN connections share a single MQTT connection to the broker) or transparent (each MQTT-SN connection contains its own MQTT connection to the broker), an optional forwarder and a MQTT broker. This complex architecture allows inter-operation between both protocols.

MQTT and MQTT-SN are many-to-many communication protocols, by letting clients publish and subscribe messages through the aid of a central broker. CoAP is mainly a one-to-one protocol for communication between a client and a server, even though it supports observing resources.

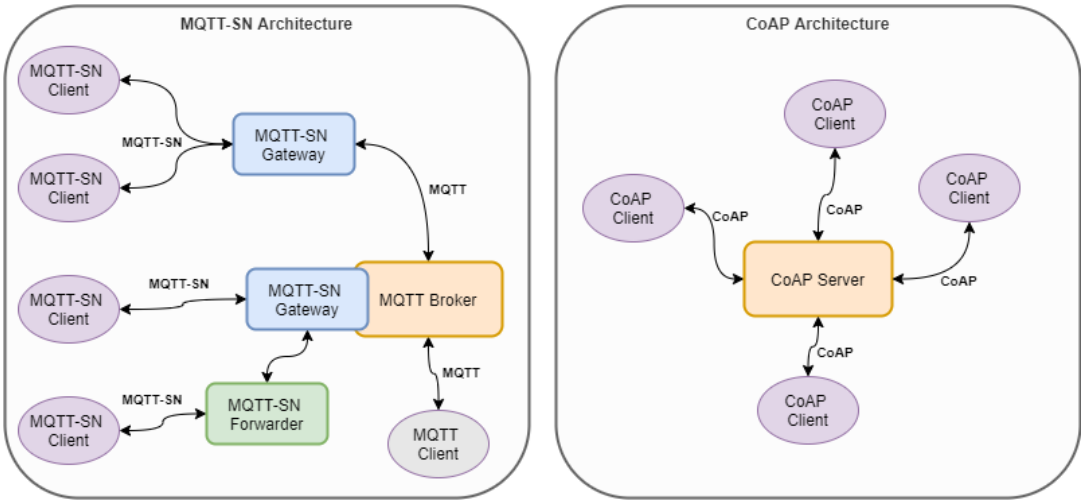


Figure 11: Comparison between MQTT-SN and CoAP architectures

Table 1: Comparison between MQTT, MQTT-SN and CoAP IoT protocols

Protocol	**MQTT**	**MQTT-SN**	**CoAP**
Method	Publish-subscribe	Publish-subscribe	Request-response / publish-subscribe
Transport Layer	TCP	UDP	UDP
Header Size	2 bytes	2 bytes	4 bytes
Application Level QoS	3 levels	5 levels	2 levels
LLN Sustainability	Fair	Good	Good
Intermediaries	Yes	Yes	No
Strengths	Reliability	Reliability	Complexity
Weaknesses	TCP	Architecture	Standard maturity

Both MQTT-SN and CoAP operate over UDP, although, MQTT-SN usually requires a virtual connection to the broker before it can send and receive messages, working similarly to MQTT and its TCP transport layer. MQTT-SN provides better application level QoS than CoAP, adding special publishing QoS levels that do not require a virtual connection, even though these modes do not work with pure gateways. MQTT-SN supports an offline keep-alive feature, buffering messages on the broker and sending them when the client wakes up. When it comes to architecture, CoAP is simpler than MQTT-SN, supported by the friendly RESTful methods, although the standard is not as mature as MQTT and provides more rudimentary QoS features than MQTT. As for the LLN sustainability, MQTT lags behind due to the use of TCP. As for intermediaries, CoAP is the only one that communicates directly with the server. The comparisons of the MQTT, MQTT-SN and CoAP protocols can be observed on the Tab.1.

2.2.3 MERN Stack

The MERN stack is one of the most popular JavaScript stack development frameworks, used to build modern SPA (Single Page Applications) web applications. It provides front-end to back-end development components, combining the following technologies:

- **MongoDB** - A No-SQL, document oriented, database management system, used to store the application data. The data is presented in binary JSON format, allowing fast exchange of data between the client and the server;
- **Express** - A server-side, back-end, lightweight framework that works on top of Node.js, managing the server and its routes, designed to write simplified, fast and secure web applications;
- **React** - A fast and scalable JavaScript library, used to build the UI components, usually for SPA;
- **Node.js** - JavaScript run-time environment, used to run Javascript on a server instead of a web browser. It also provides access to NPM, which hosts a large number of packages, and is based on the event-driven, non-blocking I/O model.

Some advantages of the MERN stack are the coverage of the full development cycle (front-end to back-end) using only JavaScript, support for the MVC architecture, to ensure smooth development flow and good community support.

2.3 Related Work

2.3.1 Monitoring System Using Web of Things in Precision Agriculture

In [30] the implementation of a prototype monitoring system for irrigation of agricultural soils based on WSN, internet integration and SMS alerts is described (Fig.12).

The architecture of the system has three main components: data acquisition, the gateway to the internet and the IoT cloud. The first comprises a ZigBee-based WSN, where each node has a Waspmote microcontroller and a water pressure sensor, without any form of energy harvesting. The gateway forwards the information retrieved by these nodes to the internet, through GPRS/UMTS technology, to the Ubidots cloud platform. From it, the farmer can observe the evolution of the soil moisture for all the locations where devices are configured and define SMS alerts, based on user-defined thresholds for soil moisture.

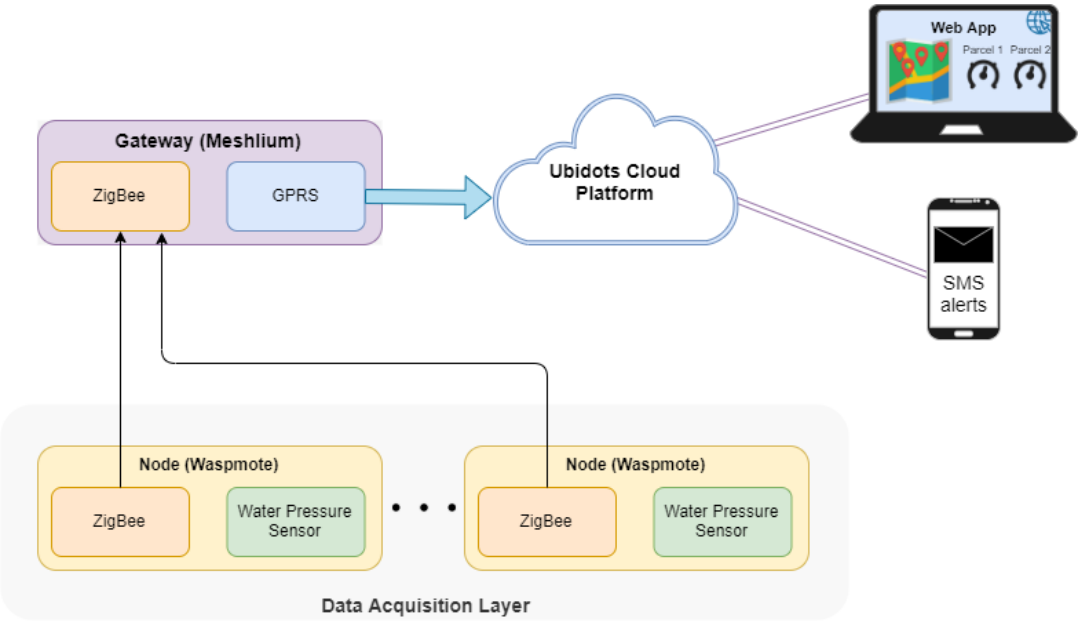


Figure 12: Architecture of the system developed in [30]

2.3.2 Low-Cost Wireless Monitoring and Decision Support for Water Saving in Agriculture

In [9], the development and implementation of a decision support system based on a WSN for water economization in agriculture is described (Fig.13). It resorts to a fuzzy logic algorithm that takes into account the experience of farmers and good irrigation practices, in order to create an irrigation plan adapted to the specific needs of the soil, regarding the weather forecasts and the conditions measured by the WSN, specifically the temperature and moisture of the air and the soil.

The ZigBee-based WSN is comprised of 4 different nodes: sensors, actuators, anchors and the gateway. The sensors measure the water pressure at the root level and have an estimated autonomy of 1 year, not containing any form of energy harvesting. The anchor nodes receive the information from the sensors and forward it to the gateway, on a hybrid star-mesh topology. Due to the frequent transmission caused by the multi-hop communications and the resulting energy expenses, these (gateways and anchor nodes) incorporate solar panels. The actuators, which also contain solar panels for longer autonomy, control irrigation valves, based upon the information received from the gateway.

Lastly, the gateway gathers the information from the WSN and performs the inference of the model. Firstly, the inference of the decision support system is scheduled according to the type of soil and the Turc method for evapotranspiration, using the sensor data as input. Then, the need to irrigate the crop field is evaluated, taking also into account the weather forecasts and the root resistance, which depends on the type of crop. The fuzzy logic model then describes the amount of water necessary,

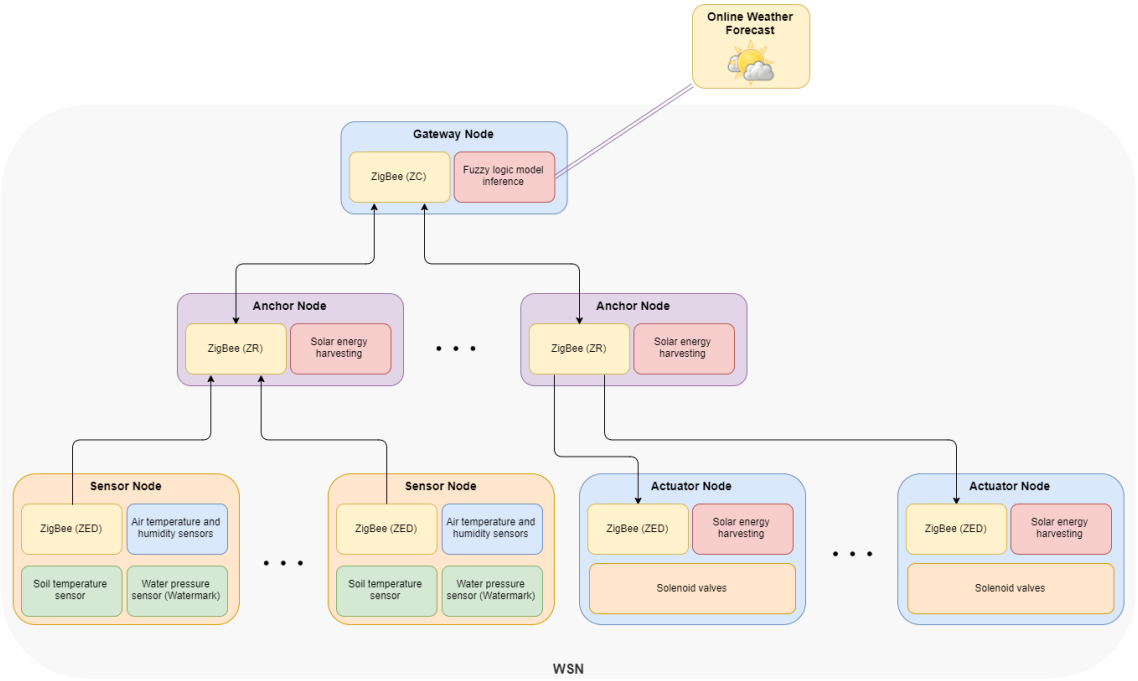


Figure 13: Architecture of the system developed in [9]

providing the user a temporal and spacial distribution of the recommended water for the entire crop field. At this point, the signal is sent to the actuators.

2.3.3 Sensoterra

Sensoterra (Fig. 14) is a solution for real-time soil moisture monitoring with multiple applications in agriculture, horticulture and smart cities ([36]). It claims potential gains of 30% in crop yield, 60% less usage of water on irrigation, 85% time economization and the return of investment in a single crop cycle. Other possible advantages of this product are quick and easy installation, measurements in multiple depths and accessibility of data at any moment.

The product emphasizes on scalability and its battery life lasts for 3 years, not containing any form of energy harvesting. The connectivity is ensured through LoRaWAN. A gateway should be acquired separately, if none are present in the area. The system is certified by the LoRa Alliance. The data retrieved by the system is accessible through mobile and desktop applications, along with Open API, easing its integration with other software systems. The soil moisture measurements are performed by a proprietary volumetric sensor (with patent solicited) that mixes capacitance and soil resistivity, which can be further calibrated according to the soil type, as the system provides multiple calibration curves in the cloud, for most soils. Lastly, there are two variants of the sensor, for single and multiple-depth measurement. Their costs for a single unit are 299$ and 799$, respectively, without extra subscription fees.

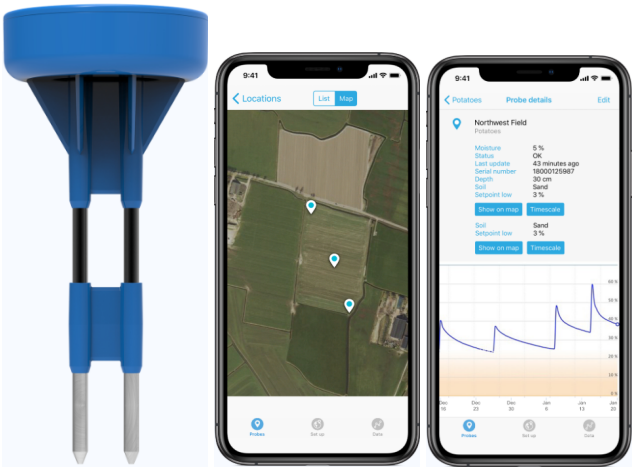


Figure 14: Sensoterra (sensor node and mobile application)

2.3.4 CropX

CropX (Fig. 15) is a data-driven system for the management of agricultural soils ([37]). Its main functionalities are the prescription of irrigation plans based on satellite imaging and soil moisture data, adapted to crop growth and weather forecasts, on a single platform. By resorting to machine learning techniques, the product also aims to help the farmer in the protection against common pests and crop diseases, by analysing crop growth against crop models and the optimization of fertilizer usage, through crop models, satellite imaging and weather forecasts.

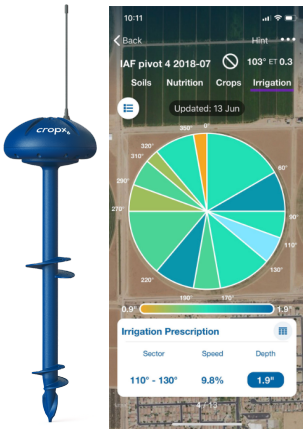


Figure 15: CropX (sensor node and mobile application)

The system is coupled to a sensor node, with two connectivity options available: LoRaWAN or cellular network. Its autonomy is limited, since it does not incorporate any form of energy harvesting. A cloud platform is available to the user in order to access the outputs of the system, which contains different access levels, so that different users have different privileges and notifications.

The sensor incorporates a patented spiral geometry, which enhances precision, reducing biased results based upon the sensor orientation. Besides moisture, the sensor is capable of measuring temperature and electric conductivity of the soil, in order to determine the salinity level. The sensor calibrates itself automatically and its spiral shape aims to simplify the installation in the soil.

Lastly, the system is available in 3 different configurations, with costs between 600$ - 899$ for each sensor node, along with an anual subscription of 275$ per sensor.

3. System Analysis

This work revolves around an optimal irrigation planning engine, which delivers irrigation plans for the next 8 days and is based upon optimal control model developed in Octave. Of course, this engine is not self-contained, it needs many parameters to deliver the irrigation plans, and is not easily accessible by anyone without considerable computer and agricultural knowledge. Hence, the main goal is to develop the infrastructure to support this engine and make it useful to a farmer.

As stated, the optimal irrigation planning engine requires many parameters to provide the irrigation plans, namely: evapotranspiration coefficient of the crop, soil coefficients and irrigation method efficiency, provided by the developers of the engine, weather forecast for the next 8 days, location, elevation, date, soil moisture and the distance to the coast.

Since many parameters can drastically change over the course of 8 days and sometimes within the same day, when considering the weather forecast, the optimal irrigation planning engine must be updated at a regular interval, and an option to perform re-planning should be provided, when necessary.

Lastly, since there is the need to further improve the model, weather forecasts and observations (observed weather conditions on the forecasted days) should be stored for future use.

3.1 Requirements

The requirements of the system are divided in functional and non-functional. The functional requirements describe actions that the system must perform, helping to describe the intended behavior of it. The non-functional requirements define a quality attribute of the system, through a set of criteria and are essential to ensure the usability and effectiveness of the system.

3.1.1 Functional Requirements

The functional requirements that have been identified for the system are:

- Acquire soil moisture data through sensors;
- Provide an individual area for each user through authentication;
- Enable the configuration of new fields and devices/irrigation points;
- Acquire weather forecast data on a web service;

- Periodically provide irrigation plans to all users and respective fields;
- Enable the request for irrigation plan updates;
- Store the necessary data for irrigation planning and additionally the inputs and outputs of the planning algorithm;
- Alert the user when the soil moisture on the field is dangerous to the crops or a device stops working.

3.1.2 Non-functional Requirements

- The system must be easy to use;
- The irrigation plans must be available on different hardware platforms;
- The system must support both single independent end nodes and multiple nodes in a local network;
- The sensors used must have a low cost;
- The sensor nodes must have long battery duration;
- The sensor nodes must have acceptable communication range;
- The sensor nodes' casing must sustain the environmental conditions.

3.2 System Overview

The system (Fig.16) should be flexible enough so that it can be used by both small farmers and massive producers, while taking into account the variability in soil and crops throughout the fields. Hence, the developed system provides two alternative configurations regarding the sensor density deployed in crop fields:

- **Standalone** - A crop field is monitored by one low power end node with energy harvesting capabilities, capable of acquiring soil moisture data from sensors and forwarding the results directly to a web server;
- **WSN** - The monitoring of the crop field is performed by multiple end nodes and an edge device, forming a local network:

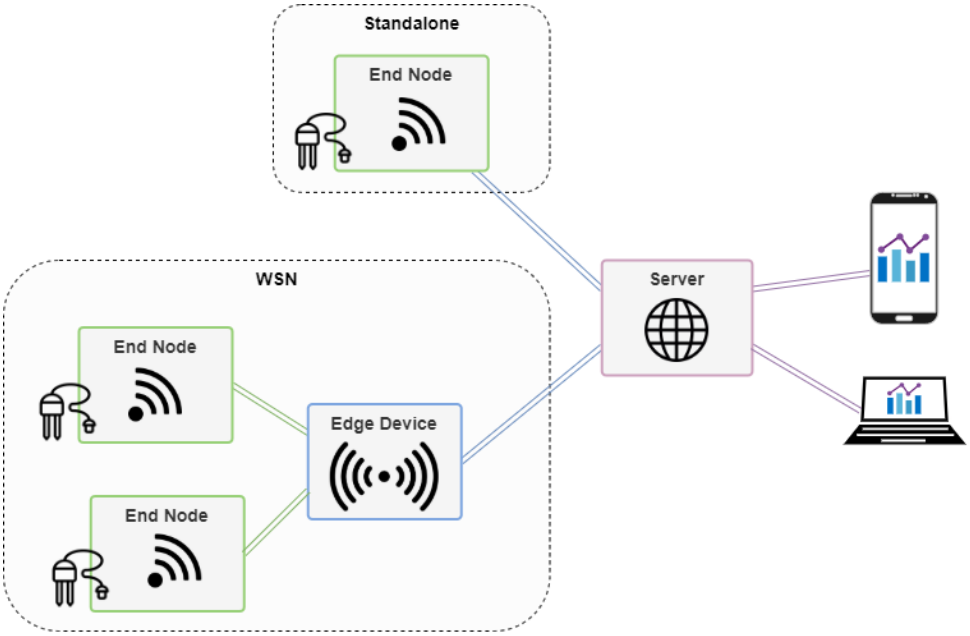


Figure 16: System architecture

- **End Node** - Low power node with the same data acquisition and energy harvesting capabilities as the nodes on the standalone network, forwarding the information locally to an edge device;
- **Edge Device** - Device capable of concentrating the soil moisture information from multiple end nodes and sending it directly to a web server, acting as a packet forwarder.

The main intent behind having two different configurations is to provide a solution suitable for both casual farmers and industrial producers. The standalone configuration, by offering a lower cost for a small number of nodes and a simpler configuration, is better suited for the former, where the latter can benefit from the higher node density, with a more detailed irrigation planning scheme.

The system functionalities are available to the farmer through a web application developed on top of the server, which, by enabling the usage in both handheld and desktop devices, enhances the overall system usability. Therefore, the farmer has easy access and availability in multiple platforms to the irrigation plans.

3.3 System Architecture

The system is supported by two different hardware nodes: the end nodes and edge devices, enabling the architecture described earlier. Beside the aforementioned hardware nodes, a web server is developed. In it, a web application is deployed to provide an interface to the irrigation plans for the farmer and to enter relevant parameters about the crop fields, the information from the nodes is

gathered, along with weather forecasts from web services and the irrigation planning algorithm is run with all this information.

3.3.1 End Node

The end node (Fig.17) is composed by soil moisture sensors, which provide the moisture readings of the soil. The readings of sensors are processed by a low power microcontroller and sent either to an edge device through LoRa technology, if the end node is part of a local WSN ,or through cellular technology (GPRS/UMTS or NB-IoT), if the end node is standalone. NB-IoT is favoured over GPRS/UMTS due to its lower power consumption, even though it is not as well established as the latter. Only one of the technologies is present on each end node. The end node is powered by a battery, which is coupled to an energy harvesting module, in order to improve system autonomy, supporting long periods of functioning without intervention by the farmer to charge the battery.

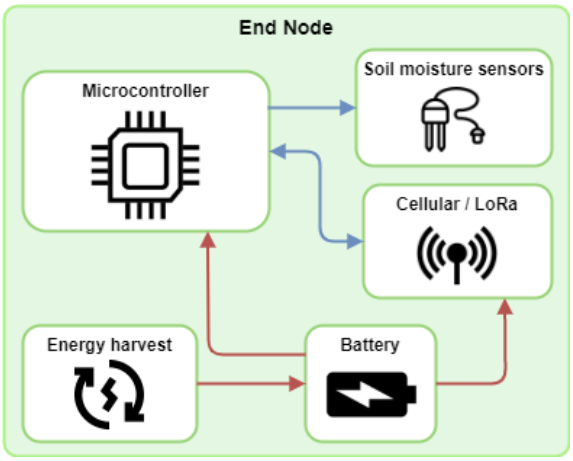


Figure 17: End node hardware architecture

From a software perspective (Fig.18), the end node stays in a sleep mode and wakes up periodically, to then retrieve the soil moisture through one or more sensors (as the soil moisture can vary according to the depth) and send them to either an edge device or directly to the server. Although, one moisture reading per day is usually enough for the optimal irrigation planning engine, an one hour period was defined for research monitoring purposes.

3.3.2 Edge Device

The edge device (Fig.19) consists of a single board computer, which receives the data from the sensor nodes through a LoRa module, forming a peer-to-peer network with the end nodes connected to it. It communicates with the server through a Wi-Fi network, or a cellular network if the former is not

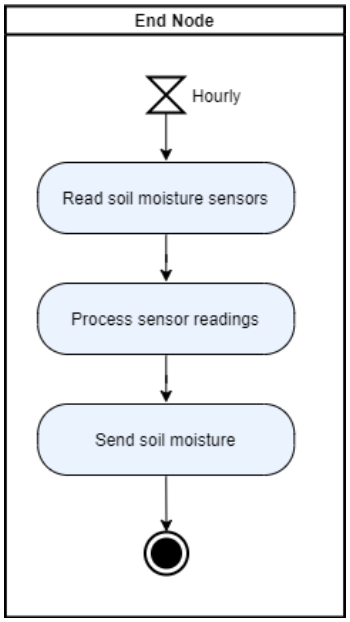


Figure 18: End node main routine

available at the deployment site. Since the edge device needs to be available to receive packets all the time, it requires a constant power source.

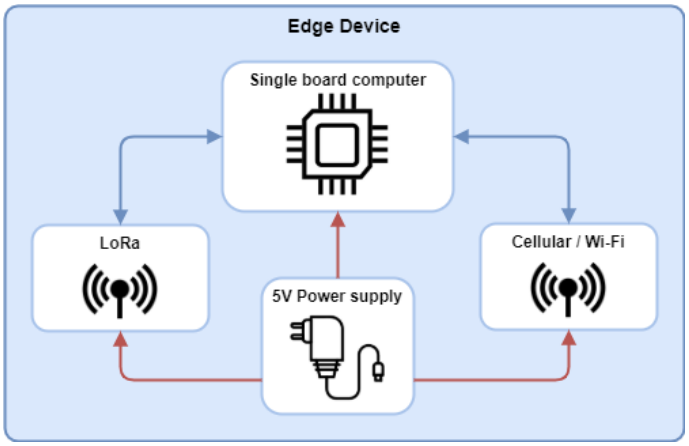


Figure 19: Edge device hardware architecture

The main routine of the edge device (Fig.20) executes when it receives a LoRa packet from an end node. After reception, the packet is sent to the server, through a lightweight protocol suited for IoT communication.

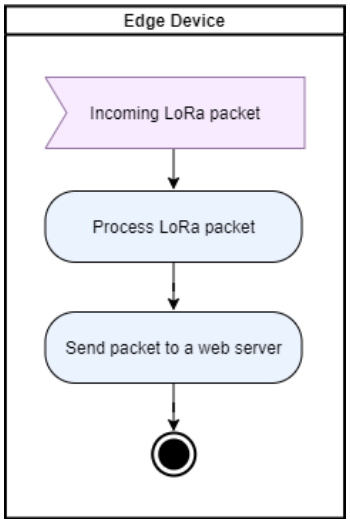


Figure 20: Edge device main routine

3.3.3 Server

Beside the software of the end nodes and the edge device, a server (Fig.21) is necessary to provide an interface for the user to interact with, commonly regarded as the front-end and the necessary services to connect it to the other parts of the server, such as the database and the optimal irrigation planning engine. These aforementioned services are part of the back-end, along with the other services that bring the system to life, that collect soil moisture information from the irrigation points, weather forecasts and perform the irrigation planning.

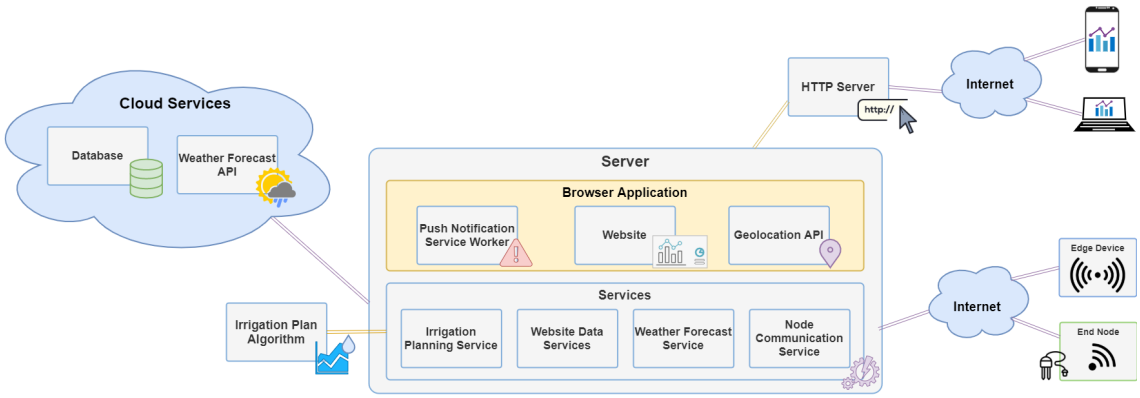


Figure 21: Server software architecture

The server contains the browser application and the necessary services that bring the system to life. Within the browser application, there is a website, the means of interaction with the user. The browser application also contains a Geo-location API, making use of the location of the device and a service worker, which runs on the background of the device even when the browser is closed, to provide

the push notification functionality. The services contained within the server provide the website with the necessary data, run the irrigation planning algorithm present in the server computer, enable the communication with the irrigation points on the crop fields and retrieve the weather forecast from a web API. The database is hosted in the cloud and also accessed through APIs. Lastly, a HTTP server is in charge of serving the website to the user.

3.4 Use Cases

In the following use case diagram (Fig.22), we can verify how the farmer and time interact with the system.

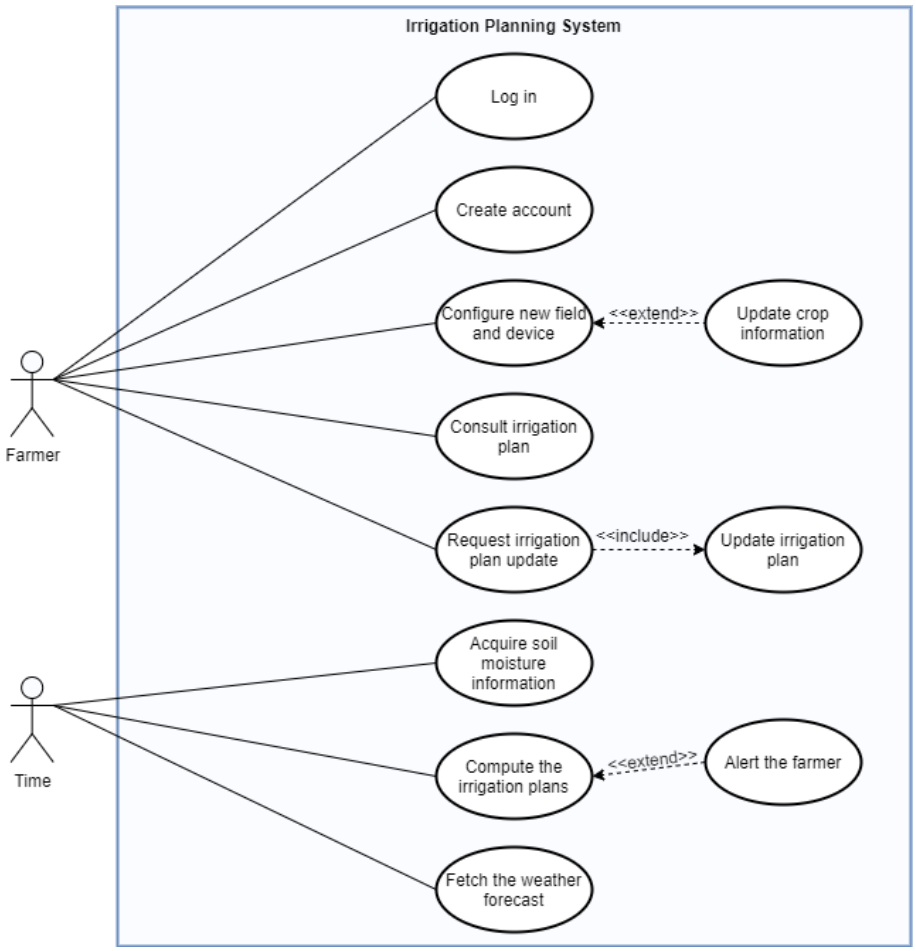


Figure 22: Use cases of the system

The farmer needs to create an account and log in to make use of the system. After creating the account, the configuration of the irrigation areas and the deployed data acquisition devices become accessible. Once an irrigation point is configured and the respective end node sends data to the

server, the farmer can access the respective crop irrigation plans and also request an update for all crop fields.

Time is a decisive actor since the system periodically performs soil moisture data acquisition, computation of the irrigation plans for all locations and the fetching of weather forecasts for a predefined number of days (for all locations).

3.5 Sequence Diagrams

In the following diagram (Fig.23), the sequence of events that take place when the farmer configures a new standalone node or changes the configuration of an existing one. The web application responds to the request asking the farmer to fill in required parameters for device configuration. These parameters include the type of crop and the crop itself, the type of soil, irrigation method and the device identifier. Then, if all the parameters are correct and the farmer is the owner of the device, the web application saves the configuration and indicates a successful configuration and the user can turn on the device; otherwise, the configuration fails. The configuration of an edge device only differs on the required configuration parameters, which do not include crop, soil and irrigation information.

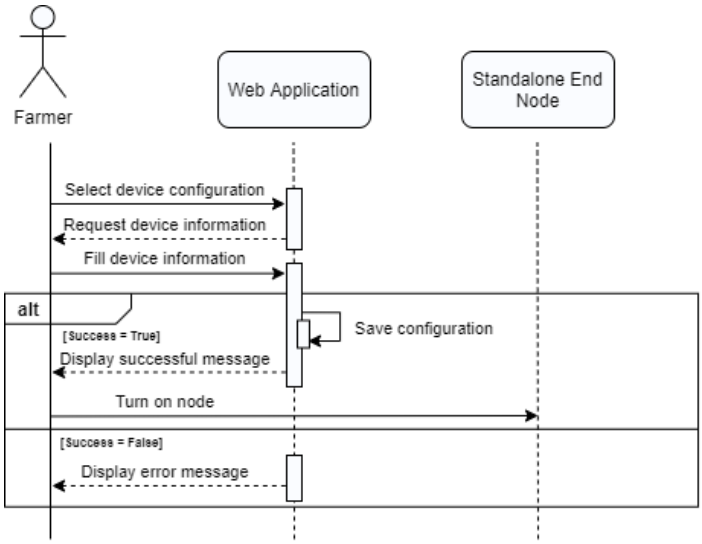


Figure 23: Sequence diagram for configuration of a standalone end node

The following sequence diagram (Fig.24) illustrates the configuration of a LoRa capable end node. From the farmer's perspective, the process is rather similar to the preceding, apart from having to designate one of his edge devices to connect the end node to. Then, the rest of the configuration is handled by the edge device, which configures the necessary local network parameters for communication with the end node.

The sequence diagram on the Fig.25 shows the process of data acquisition by an end node, which happens periodically. It starts when an internal time based event wakes up the node from a power-

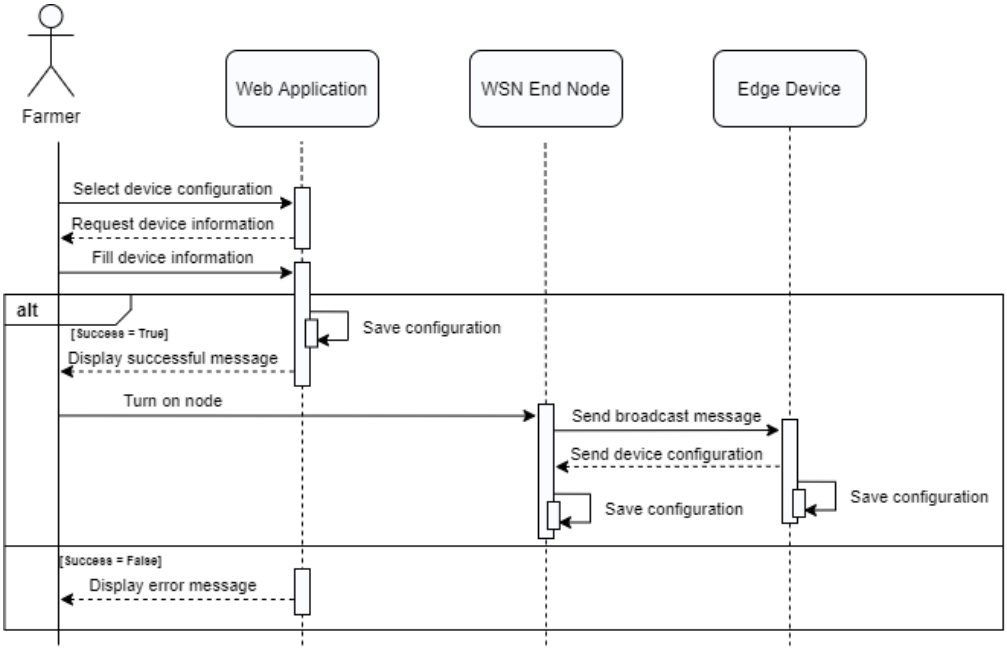


Figure 24: Sequence diagram of a LoRa end node configuration

down mode. Then, the node initializes the necessary peripherals and retrieves the soil moisture through the sensors. After gathering and processing the sensor data, it is sent to an edge device or directly to the server, depending on the node connectivity. Upon completion, the node re-enters a low power state.

The following sequence diagram (Fig.26) shows the irrigation plans computation process. Once a day, an internal timer event on the server triggers the planning computation for all active irrigation points. For each irrigation point, the latest weather forecasts and soil moisture data are fetched from the database, to then compute the respective optimal irrigation plan. When the crop conditions are dangerous, the web application sends a push notification to the farmer. In the end of the process, the new computed plan is saved on the database.

The sequence diagram on the Fig.27 demonstrates the events that take place when the user requests an update of the irrigation plans. Firstly, the user requests through the web application interface an update of the plans. Then, the same process as the daily calculation takes place. Since the plans will be displayed immediately to the user, no push notifications are sent.

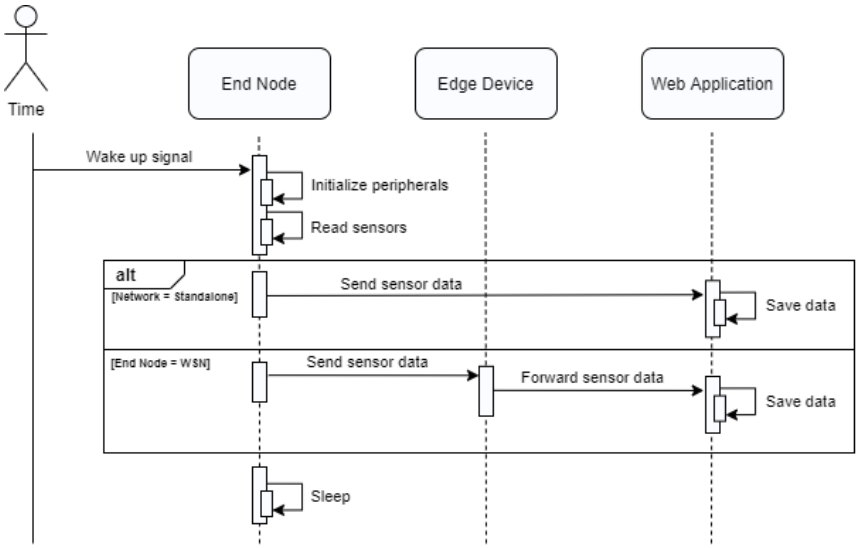


Figure 25: Sequence diagram of data acquisition

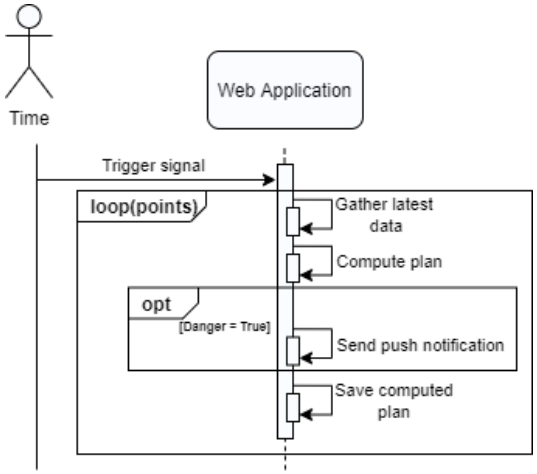


Figure 26: Sequence diagram of plan computation

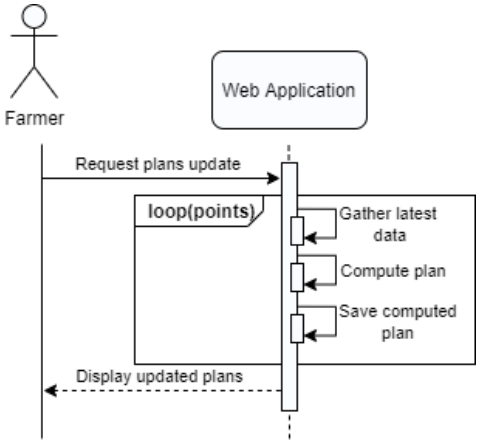


Figure 27: Sequence diagram to update the irrigation plans

4. System Design

4.1 Local Network

Since a pure LoRa gateway is relatively expensive for the number of devices a farmer usually needs in a crop field, a transceiver was used in the edge device, which is cheaper, although, very limited in communication capabilities, only being able to listen to a single frequency and spreading factor at once.

The local network employs the LoRa physical layer to achieve a reasonable ratio between communication range and energy spent on the communication. Due to the limitations of the edge device, only a packet with a single frequency channel and spreading factor can be received at a time. By fixing the SF, energy consumption or range will be compromised, since higher spreading factors lead to longer ToA and energy consumed, even on smaller distances, where the smallest SF would probably suffice. By switching spreading factors when listening to incoming LoRa packets, this problem would be mitigated. By adding a preamble long enough, package loss can be minimized, although another issue would occur: since LoRa has imperfect orthogonality, adjacent spreading factors could be detected and the packet would be lost.

So, as in [2], the SF selection algorithm represented in Fig.28 is adopted. It performs channel activity detection (CAD). If a preamble is not detected, it switches the SF until one is detected. If a preamble is detected, then it uses the same spreading factor and performs CAD again. At the third consecutive successful detection, it either prepares to receive the LoRa packet or performs CAD detection on an adjacent SF, until the current SF is not able to find a valid preamble. This step is necessary on some spreading factors, because of the imperfect orthogonality between adjacent spreading factors. In the latter case, the SF is decreased by one, since it is the last valid candidate, and the LoRa transceiver is placed in reception mode. By adopting this algorithm, false SF selection is mitigated, solving the orthogonality problem stated before.

The preamble length for each SF needs to be high enough so that a minimum amount of packets are missed, which can be calculated by analysing the worst case scenario. For SF=7 and SF=10 the preamble length needs to be approximately an entire cycle of cumulative CAD detection periods (which can be obtained in the transceiver datasheet) until it successfully detects the right SF. This process still has to take into account the necessary time to process the CAD detection interrupt service routine, which may include the reading of a register through SPI or an input GPIO pin, along with the clearing of the necessary flags, for every CAD detection cycle. The worst case scenarios for packet detection with the ASFS mechanism can be calculated as follows, for the spreading factors

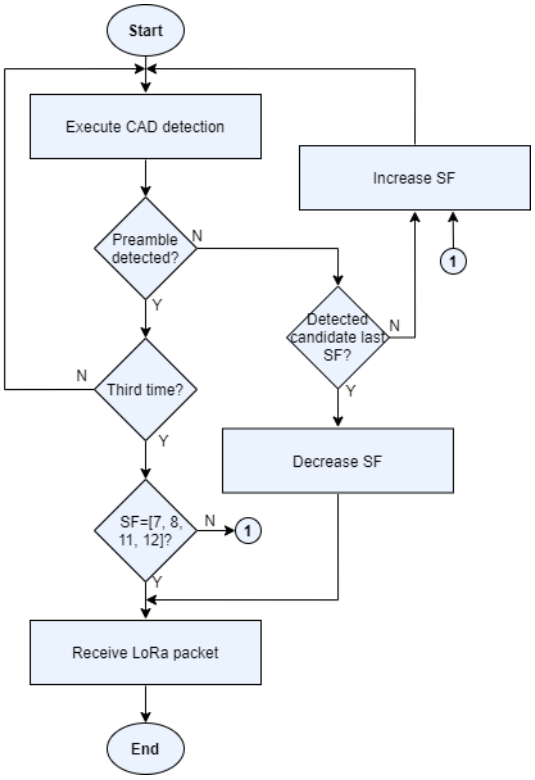


Figure 28: Flowchart of the ASFS mechanism

that do not suffer from orthogonality (SF=7) and for those that do (SF=10), where T_{CADsf} is the time taken by the CAD detection period for the given SF and T_{ISR} the period taken to process the interrupts to control the ASFS algorithm:

$$3T_{CAD7} + T_{CAD8} + T_{CAD9} + T_{CAD10} + T_{CAD11} + T_{CAD12} + 8T_{ISR}, SF = 7$$

$$T_{CAD7} + T_{CAD8} + 3T_{CAD9} + 3T_{CAD10} + 2T_{CAD11} + T_{CAD12} + 11T_{ISR}, SF = 10$$

The aforementioned LoRa communication parameters, coupled with the appropriate preamble lengths for each SF, allow the calculation of the ToA, which has a direct impact on both the end node autonomy and network range. While a longer ToA negatively impacts the energy life of an end node, it also allows for a wider communication range.

Lastly, the communication parameters of the end nodes are calculated by the edge device similarly to the ADR mechanism of LoRaWAN, by defining a margin and relating the maximum SNR of the received packets to the SNR limit for each SF:

$$margin(dB) = \max(SNR_{packet}) - SNR_{limit}(SF) - margin_{default}$$

Another important factor is the timing of communications, taking into account that an increasing number of nodes communicating in the same area would result in packet collisions, degrading network performance. Hence, a collision avoidance mechanism should be employed, but not any one, since methods such as CSMA would degrade the system autonomy. An algorithm was developed to order the communication of all nodes connected to same edge device, and thus avoiding collisions. The edge device allocates a time slot for each irrigation point, which results in the timing distribution shown in Fig.29, where the number indicates the order of configuration of the device in the network.

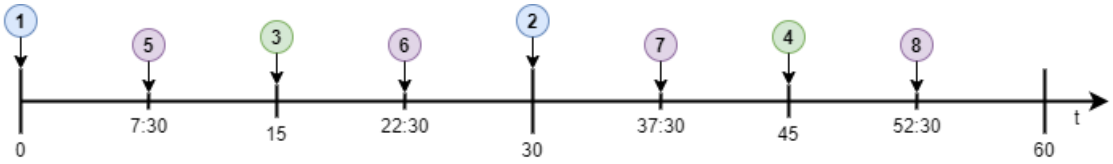


Figure 29: WSN end node configuration timeline (mm:ss)

The algorithm creates an interval between node communications that is as large as possible. For that, it takes into account the number of devices in the network and that they all need to communicate during a one hour time span. A calibration value is attributed in the configuration of the node, to align the device timing with the one used by the edge device. By setting a maximum of 254 nodes on the same local network, the smallest interval between each node is approximately 14 seconds, which is more than enough for communications without collision, even at the maximum SF.

Since the LoRa MAC layer is not used, a messaging protocol is developed on top of the physical layer. The messaging communication protocol developed is shown in Fig.30 and defines two types of uplink messages and two types of downlink messages, for configuration and communication of soil moisture data. The end node uplink configuration contains its ID (which is also located inside the application database) leaded by 32 bits of zeros, to be uniquely identifiable. Its downlink response contains the source address of the assigned edge device, followed by 8 bits of zeros, the assigned device address (of 8 bits, limiting the number of nodes to 254, where 0 is reserved) and configuration parameters. Time sync is used to calibrate the wake-up time with the time slot selected by the edge device, in minutes and seconds. Since soil moisture changes slowly overtime, this has no negative implications on the data validity. Ack sync defines the synchronization mechanism for acknowledgement messages, because downlink messages are limited by the ISM regulation. The soil moisture data message contains the two addresses, the soil moisture of the irrigation point and options, where an ack can be required. The downlink acknowledgement message configures the transmission power and SF.

Lastly, the configuration of new end nodes must be performed with the edge device within reach, otherwise the configuration might fail. The operation uses the smallest spreading factor, so that if the configuration fails, the process can be restarted fast enough, while respecting the government duty cycle regulations applied to ISM bands.

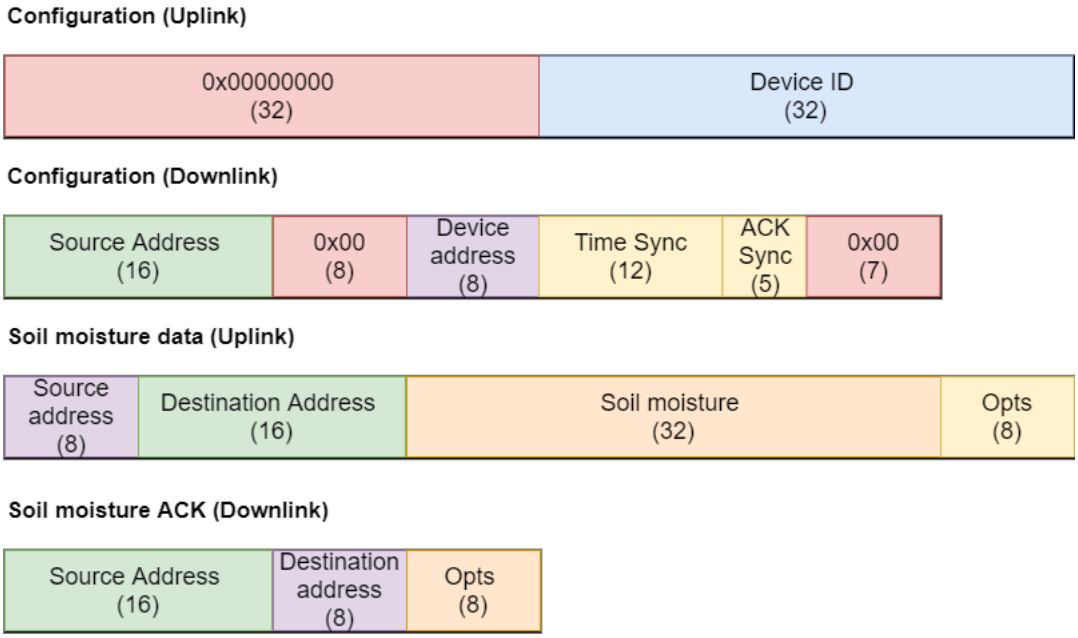


Figure 30: LoRa-based developed messaging protocol

4.2 Communications with the Server

The standalone end nodes and the edge devices communicate the soil moisture information to the server through CoAP. Since soil moisture changes occur at a slow rate, devices send data with a one hour frequency and irrigation plans are calculated once every day, a lost message overtime is not a concern. Therefore, devices send CoAP non-confirmable messages to the server. The messages are not encrypted and are defined in Tab.2.

Table 2: CoAP requests made by the system

Device	Method	Type	URI	Payload
End Node	POST	NON	device/{id}/moisture	4

4.3 Behavior

The configuration of standalone end nodes and edge devices is depicted by the Fig.31. After choosing that option, the user fills in the necessary configuration parameters on the web application. After submitting the form, the server makes sure the device is owned by the respective user, by querying the database. If so, the web application saves the configuration, instructs the user to power on

the device and the operation is successful, since these devices do not require further configuration. Otherwise, the process fails and the form data is discarded.

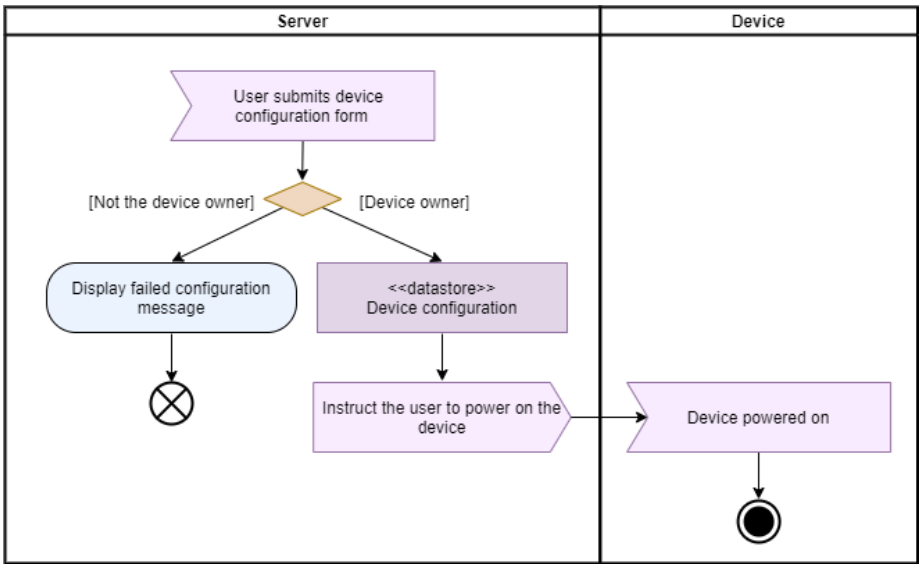


Figure 31: Activity diagram of the configuration of standalone end nodes and edge devices

The configuration of LoRa end nodes (Fig.32) starts in the same manner, but since these nodes are not connected to the internet, the server verifies that the network created by the selected edge device is not full. When the end node is powered on, it sends an uplink configuration request according to the messaging protocol defined in Fig.30, to which the edge device responds (after having computed the algorithm defined in Fig.28) with a downlink message with the necessary configuration parameters.

The data acquisition process of the standalone end node (Fig.33) starts every hour, when the internal RTC from the microcontroller wakes the system from sleep mode. Then, the system runs its wake up routine, where the peripherals are enabled for operation. Then, all the soil sensor values are obtained and the conversion function for them is applied. When the output soil moisture is obtained, the CoAP message defined in Tab.2 is sent to the server, containing the soil moisture value of the respective irrigation point. Then, the device enters its sleep routine.

The data acquisition activity of the LoRa end nodes (Fig.34) is similar to the former, differing on the communications. After sending the soil moisture packet through LoRa to the edge device, the end node either waits for an acknowledgement (once a day) or goes directly to sleep mode. When an acknowledgement is requested in the LoRA packet, the LoRa module is put into reception mode. In case of a reception timeout, the SF and transmission power are increased to the maximum, maximizing the chances of reception in further iterations. If the ack packet is received, the node changes the transmission configuration, according to the values computed by the edge device.

The main activities executed by an edge device are the forwarding of soil moisture data received from end nodes (Fig.35 - left) and the reception of new LoRa packets (Fig.35 - right). The soil moisture

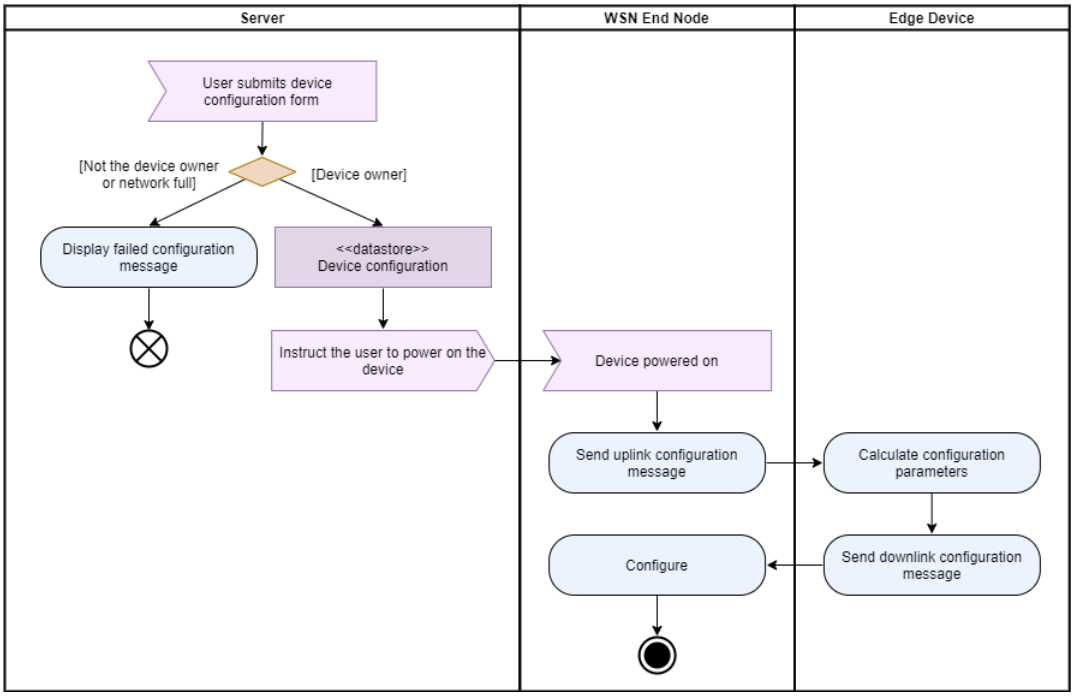


Figure 32: Activity diagram of the configuration of LoRa end nodes

data is sent through a CoAP POST request after processing the received data from the LoRa end nodes. The reception of new LoRa packets starts by detecting incoming packets through the ASFS algorithm. In case of a successful preamble detection, the packet is received and processed, by verifying the CRC contained in the packet. Then, the process repeats.

The activities related to irrigation plans are the daily execution of the irrigation model to compute plans for all end nodes (Fig.36 - left) and to respond to each irrigation plan update requested by an user (Fig.36 - right). The daily model planning starts by the obtaining from the database the location, soil moisture and crop information for each active irrigation point. The locations are filtered through a threshold of latitude and longitude, and the weather forecast is fetched for each filtered location, using the DarkSky API. Then, the irrigation planning algorithm is run for each point and the necessary database documents and excel files are updated. If dangerous conditions are verified, a push notification is sent through the respective API endpoint. In the latter activity (Fig.36 - right), the same process takes place for the irrigation point whose plan the user requested to be updated. Since the farmer will observe the results on the screen, only the database documents are updated. Lastly, the newly calculated irrigation plans are displayed to the user.

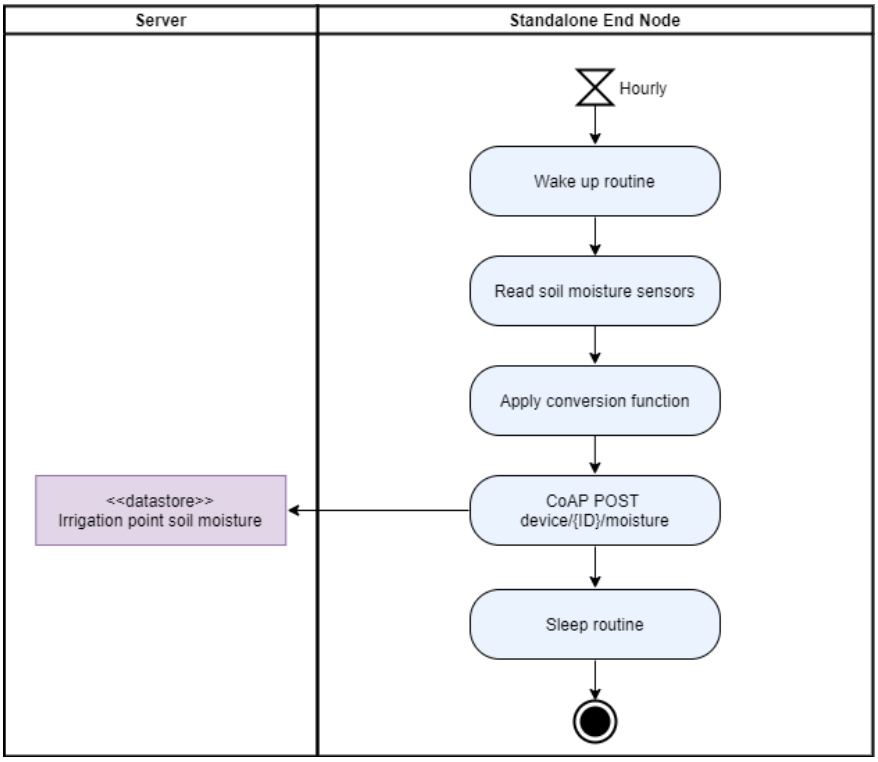


Figure 33: Activity diagram of the data acquisition of the standalone end nodes

4.4 End Nodes

4.4.1 Hardware Description

- **STM32L072CZ** - 32-bit low power microcontroller with the necessary peripherals and communication interfaces, such as a 12-bit ADC up to 16 channels, 4 USART and 6 SPI peripheral interfaces and 11 timers, including an RTC. In order to speed up development, the B-L072Z-LRWAN1 board was used (Fig.37), which already incorporates the SX1276 LoRa module.
- **RFM95W** - LoRa transceiver used by the WSN end node, based on the SX1276 chip, featuring a receiving current of 10.3mA, preamble detection, 127dB RSSI indicator, 14dBm PA and CRC calculation capabilities. It provides SPI and GPIO interfaces (see Fig.38) to talk to the microcontroller. The DIO pins are configured as RxDone / TxDone and RxTimeout, respectively.
- **SIM7000E** - GPRS / UMTS and NB-IoT module used by the standalone end node, supporting UDP, controlled via AT commands and containing multiple power-down modes, including a PSM mode of 9μA available with NB-IoT and a sleep mode of 1mA. The UART and GPIO interfaces are used by the microcontroller to communicate with the module, which is powered

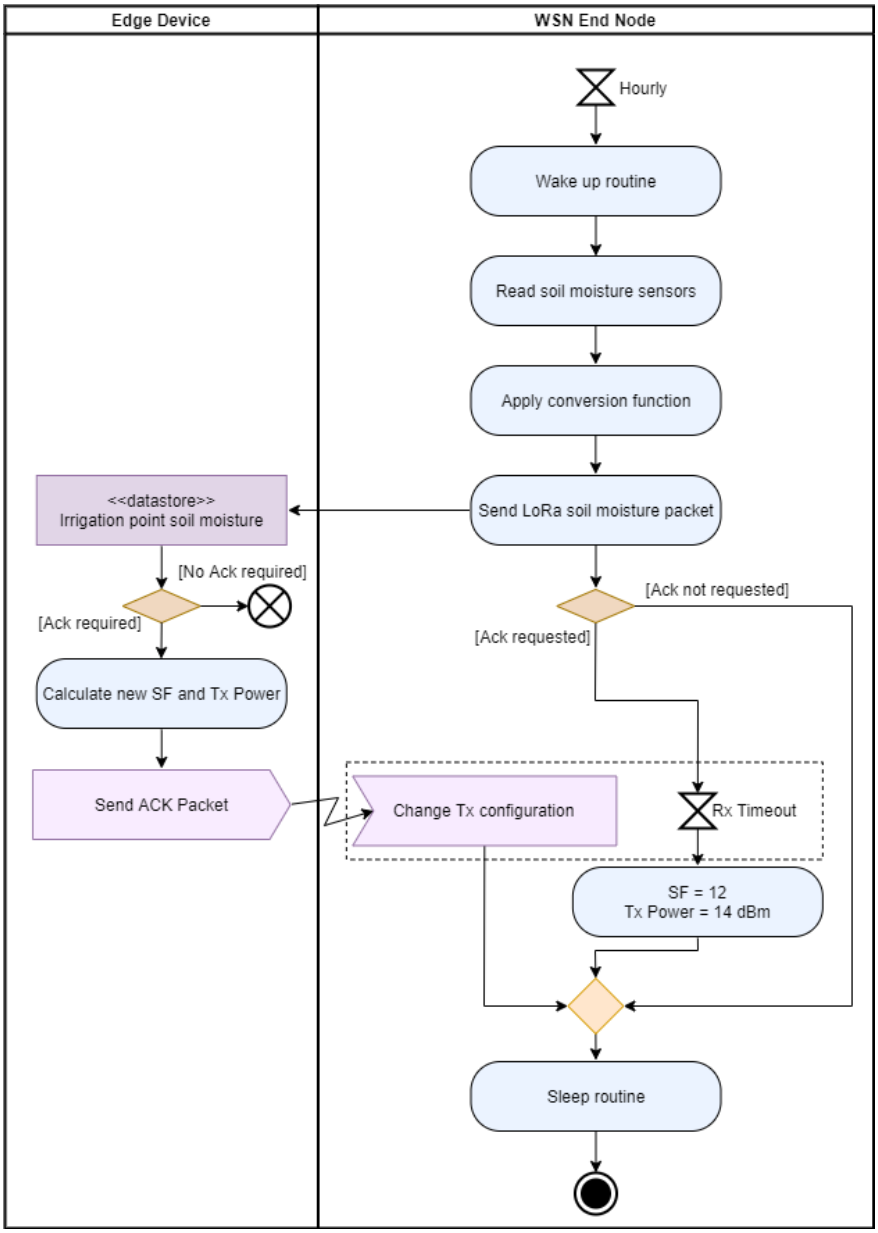


Figure 34: Activity diagram of the data acquisition of the LoRa end nodes

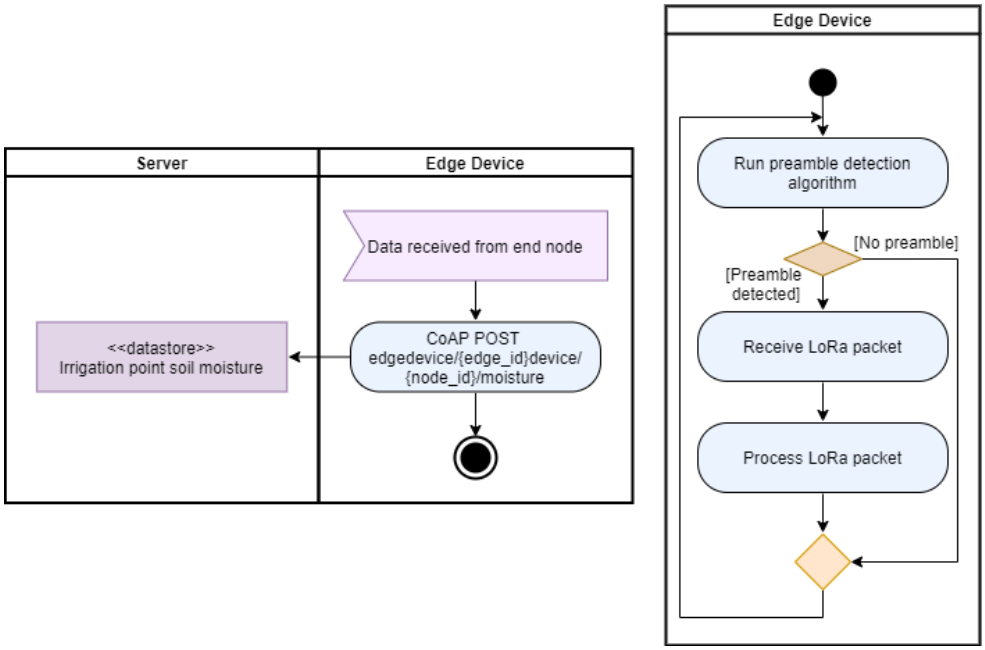


Figure 35: Activity diagrams of the normal operation of an edge device

by the step-up converter. Lastly, the converter should be more powerful than the one present on the WSN end node, since this module draws power from it (Fig.39).

- **Gravity: Analog Capacitive Soil Moisture Sensor - Corrosion Resistant** - Low cost soil moisture sensor, with 3.3-5V operating voltage, an analog output of 1.2-2.5V and a current of 5mA. Besides the aforementioned features, some studies have explored the sensor capabilities and provided calibration equations for it, as in [43], which is used in this work to obtain the best results. The ADC interface by the microcontroller is used to obtain up to 6 sensor readings, along with GPIO and extra circuitry to power-off the sensors when they are not in use, since the current consumed by them is considerable. To accommodate any voltage drops on the control circuitry, 5V are used to power the sensors (Fig.40).

- **PCB** - In order to connect the microcontroller to the sensor and the transceiver, a PCB was developed with the Altium Designer software. It also connects to a Li-Po battery, photovoltaic panel and step-up converter headers, a solar energy harvesting circuit, aside from providing a circuit to power-on or off the sensors and the ST-Link header, in order to program the microcontroller. SMD components were preferred to minimize the PCB size, which is 35.7mm by 40mm (Fig.41).

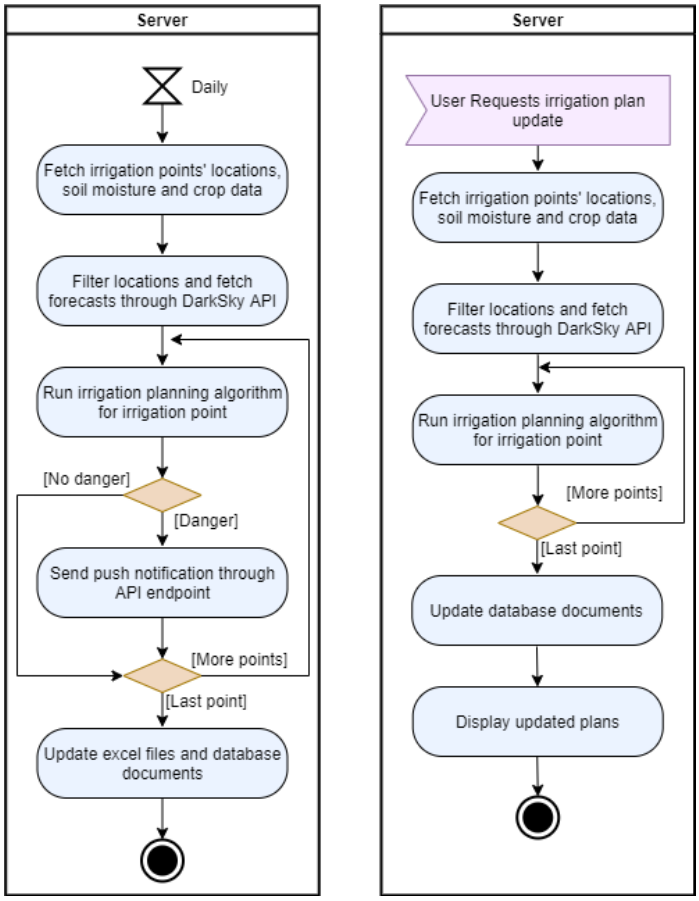


Figure 36: Activity diagrams of the irrigation planning

4.4.2 Tools/COTS

FreeRTOS, the market-leading real-time operating system for microcontrollers, caracterized by its robustness, small kernel footprint and wide support, was used.

4.4.3 Software Architecture

Three tasks are performed over the FreeRTOS scheduling system, which communicate between them and with the microcontroller peripherals through message queues and semaphores.

The WSN end node tasks manage the power, soil moisture sensing and LoRa communication (Fig.42). The PwrManagement task handles the RTC of the microcontroller, synchronizing it to the network, according to the message received from the LoRa task. The soilSensors thread samples the soil moisture when the RTC interrupt signals it through a semaphore. Then, it turns on the sensors and waits for a timer to signal the sensors are ready to use. Then, it samples the moisture and waits for the ADC signal, meaning the measurements are ready, when the conversion function

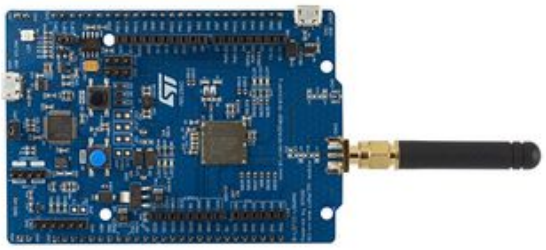


Figure 37: B-L072Z-LRWAN1 discovery kit

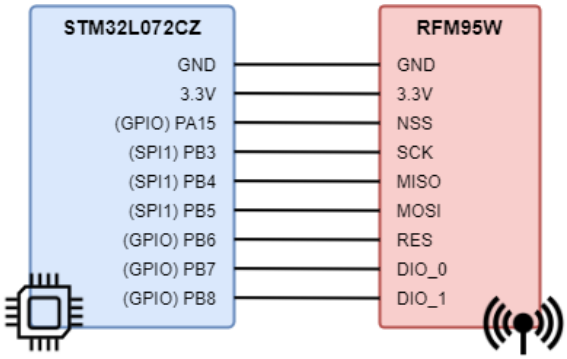


Figure 38: RFM95W connection diagram on the end node

is applied and sent to the LoRa thread, via a message queue. Lastly, the LoRa thread configures and manages the transceiver, implementing the communication functionalities. It makes use of a semaphore, to signal when the device is ready for operation after reset or after SPI operations. It also uses a message queue to receive status from the GPIO interrupts, where timeouts or successful reception or sending of packets are sent to the thread.

The standalone end node tasks are similar to the ones of the WSN end node, with minor changes that reflect the different functionalities, namely a CoAP thread instead of a LoRA one, see Fig.43. The soilSensors functions the same way as in the WSN end node. The CoAP thread sends the soil moisture to the web application, by running a CoAP client over the cellular communication module. The module status is sent to the thread through a status message queue, signaling when either an AT command is sent, a response is received, or a timeout or another error occurs. The pwrManagement thread no longer has to configure the timing of the first wake-up triggered by the RTC, so, unlike the WSN end node, the communication and pwrManagement threads are independent from each other.

The application running on the end nodes makes use of the C++ programming language and the aforementioned FreeRTOS kernel. From a structure perspective, the software can be regarded as a bundle of three layers: the hardware specific to the microcontroller, the scheduling, based on FreeRTOS and the tasks that give the end node its functionalities.

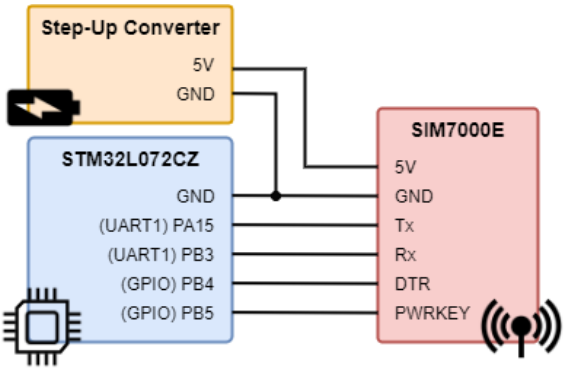


Figure 39: SIM7000E connection diagram on the end node

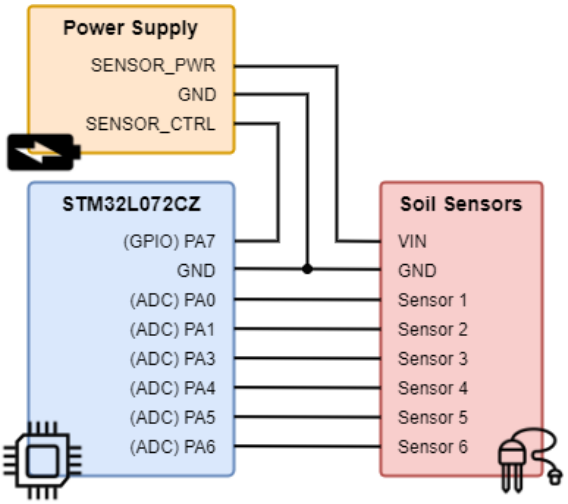


Figure 40: Soil moisture sensors connection diagram on the end node

4.4.4 Software Detailed Design

Hardware Abstraction

Template metaprogramming is used to implement classes that represent the microcontroller hardware, providing highly abstract yet efficient code, along with many other advantages. The developed class templates are shown in the UML class diagram of Fig.44 and provide access to the GPIO, UART, SPI, RTC, Timer and ADC peripherals.

The GPIO class overloads three different initializers, to configure pins with or without interrupt handler (which needs to be declared static because the microcontroller interrupts requires C language linkage) and to configure alternative functions in GPIO pins, necessary for SPI, UART and ADC configuration. It also includes the three necessary pin output functions: set, reset and toggle.

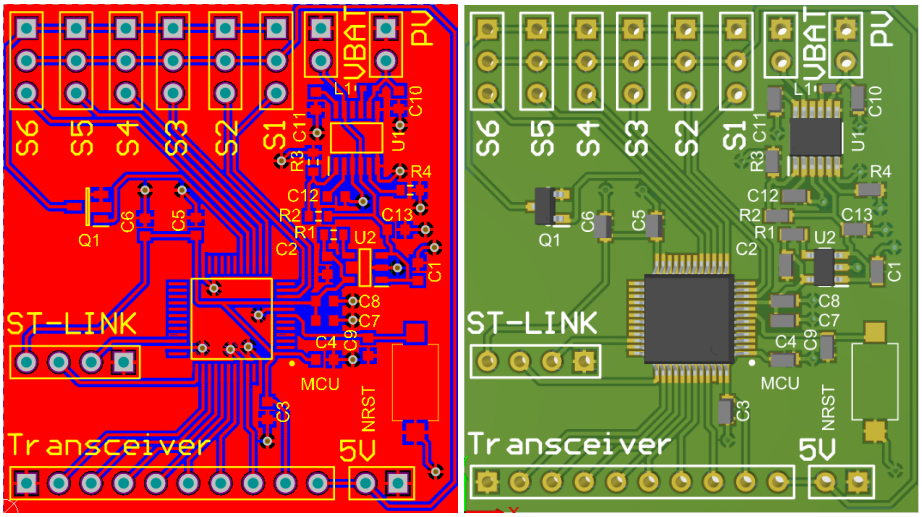


Figure 41: Layout of the developed PCB

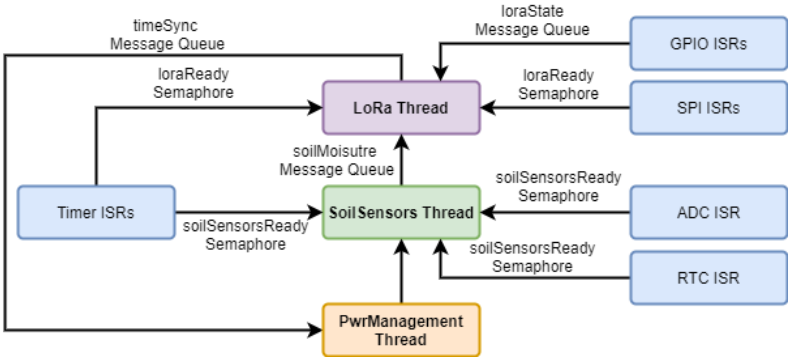


Figure 42: Communication between tasks and interrupts of the WSN end node

The RealTimeClock class allows the configuration of an alarm, by setting the respective hours, minutes and seconds, along with providing an interrupt handler.

The Timer class supports timers, by supplying a timeout in μs and an interrupt handler. Enable and disable functions are also provided to shutdown the timers before entering sleep mode. The LPTimer class works in a similar manner, but since it is used as the ticking source for the RTOS, it never turns off, even in lower power modes.

The ADConverter class helps to use ADC channels. Initialization functions for both the converter and the channels are provided, where the GPIO class is used to configure the alternate function on the corresponding GPIO pins. The interrupt handler is shared across all channels and the results from the conversion can be accessed by the adcBuffer. Lastly, it also contains enable and disable functions, to save energy.

The SPI class provides access to the peripheral of the same name, where read and write functions are provided for single byte and burst operations, and a buffer is available to access the received

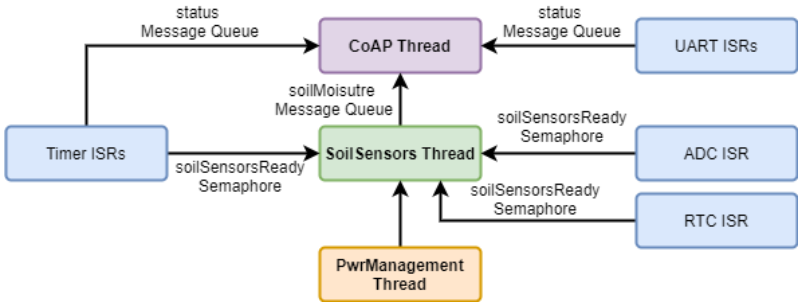


Figure 43: Communication between tasks and interrupts of the standalone end node

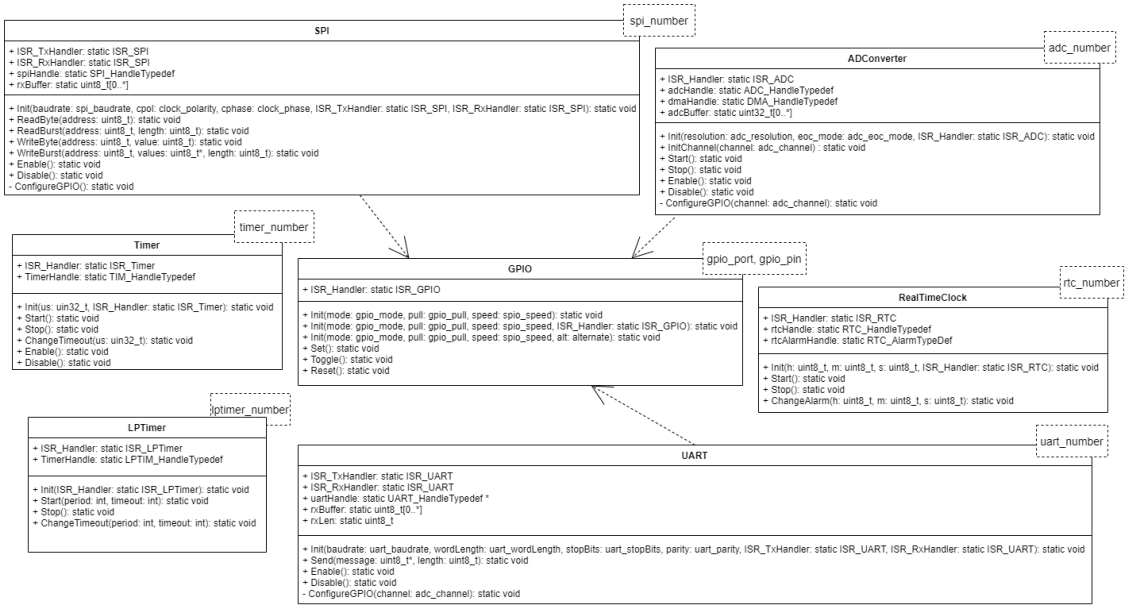


Figure 44: End node class templates abstracting hardware

data. Enable and disable functions are also provided by it. Lastly, it makes use of the GPIO class to configure its pins.

The UART class includes send and receive functions to communicate, providing the reception buffer and the length of the received data. It also provides enable and disable functions, and makes use of the GPIO class to configure the hardware pins.

Scheduling

Since FreeRTOS is a C library, three classes were developed around it to wrap the functionalities required by the system, see Fig.45. The Semaphore class allows the configuration of a binary or counting semaphore, and supplies the necessary take and give functions. The MessageQueue class

allows the creation of a message queue of any type, containing also the push and pop functions. The Thread class enables the instantiation of threads, controlling their priority and stack size.

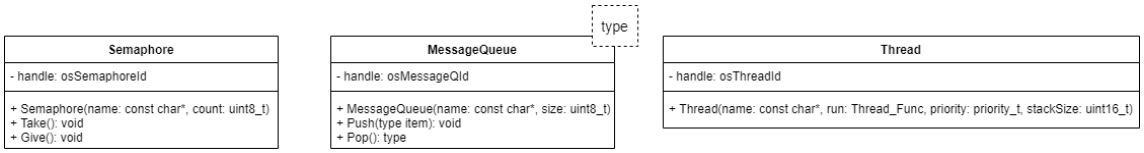


Figure 45: End node scheduling-related class templates

Tasks

In order to bring the application to life, each thread is contained in a class, with the necessary hardware peripherals, interrupt service routines and thread communication/synchronization mechanisms. Since there should be only one class instance for each thread, the singleton design pattern is used, which also allows access to the instance variables inside the static interrupt handlers.

The LoRa class of WSN end nodes (Fig.46), contains the necessary peripheral objects to operate the RFM95W transceiver and the functions that implement the protocol depicted in the Fig.30: sending and parsing of received packets, communication management and entry on the local network.

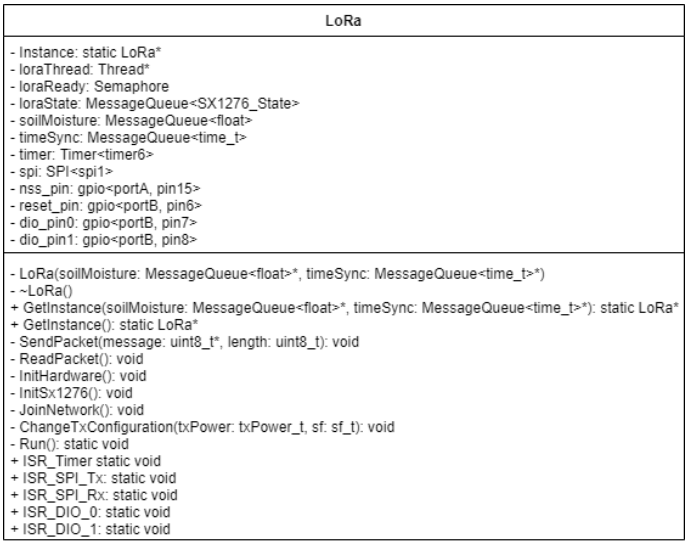


Figure 46: End node LoRa thread class

The CoAP class of the standalone end nodes (Fig.47), builds up on the Cellular class, which provides the underlying UDP interface for the protocol, by controlling the SIM7000E module and makes use of the CoAPPacket class, that contains the CoAP packet structure. In addition, the CoAP class implements the protocol operations, with the exception of DTLS operation.

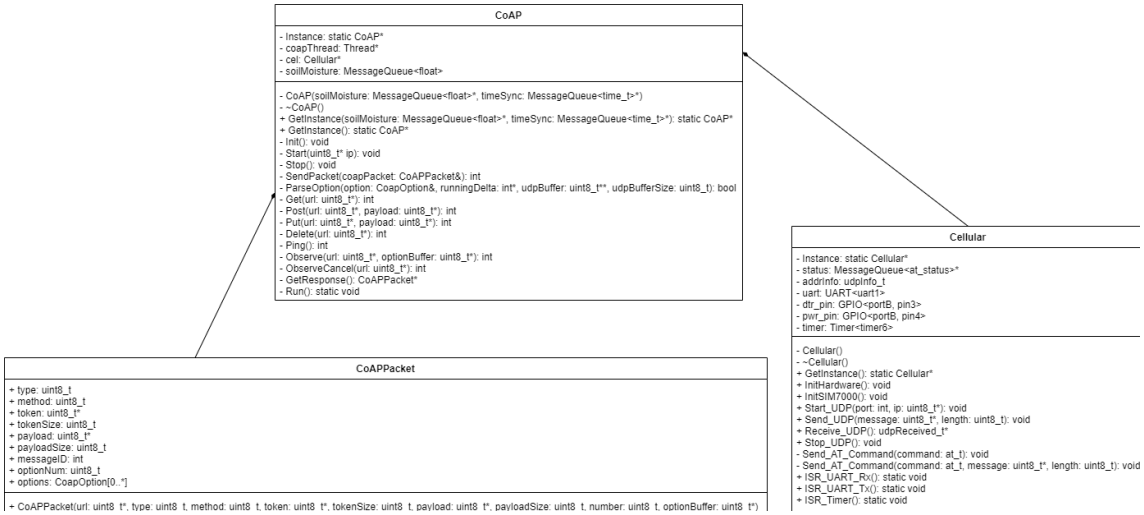


Figure 47: End node CoAP thread class

The SoilSensors class (Fig.48) controls the power to the sensors and extracts the soil moisture values from them. It also converts these values into VWC units, ready to be sent to the server irrigation planning algorithm.

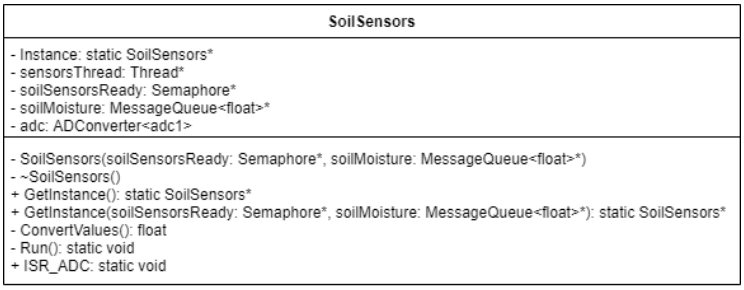


Figure 48: End node SoilSensors thread class

Lastly, the PwrManagement class (Fig.49) manages the RTC and triggers the execution of the SoilSensors thread.

4.4.5 Power Saving Measures

In order to ensure long deployment periods, some measures have to be adopted to save as much power as possible. Starting with the peripherals, the sensors are switched off via a low side switch, implemented by using a N-MOSFET. This way, the microcontroller can turn on the sensors only when needed, by outputting 3.3V in a GPIO pin. Regarding the radios, everything can be performed by software, by issuing AT commands (on the SIM7000 module) or configuring the respective SPI registers (on the RFM95W transceiver) to place them into a sleeping state when they are not in use.

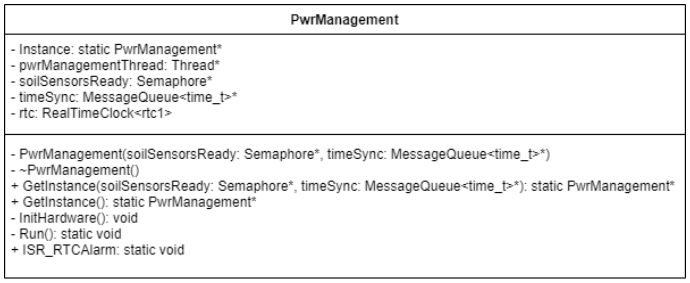


Figure 49: End node PwrManagement thread class diagram

Regarding the app running on the microcontroller, there is a power-saving feature on FreeRTOS, the tickless idle, which places the microcontroller in a lower power state and suppresses ticks from the RTOS and automatically adjusts the tick count in the end. The problems with this approach are that the system would be in that state for little periods of time, when compared to the hourly execution of the main routine, and all clocks and peripherals are still kept running, along with the ARM Cortex M0 core. So, the power savings are little.

The FreeRTOS idle hook functionality allows to place code inside the lowest priority task and it was used to decide whether the system should go into the stop mode. By not using the tickless idle, the system has to be ticking all the time, so the ticking source still needs to be available in this mode. There are two options: the RTC and the low power timer. Since the RTC is already in use, the low power timer was chosen. The timer was also configured to its maximum timeout when it comes to this mode, since its interrupts will still wake up the system from the deep sleep mode. In the stop mode, the core is stopped, along with a set of peripherals and high speed clocks, allowing for deeper power savings. The idle hook follows the logic presented in Fig. 50, where the order to place the system into the lower power state is given by the sending task, when the main routine ends.

Besides the aforementioned measures, some other precautions were taken, when it comes to the peripherals of the microcontroller. Firstly, all unused pins are configured as analog, which is recommended by ST to save power, and all the peripherals are disabled when not in use.

4.5 Edge Device

4.5.1 Hardware Description

- **Raspberry Pi Zero W** - Single board computer with an onboard Wi-Fi module, aside from the essential interfaces for the edge device operation. Besides the Wi-Fi capabilities, it contains a 1GHz single-core CPU, 512MB of RAM and can be powered via micro USB, in its small form factor (Fig.51).

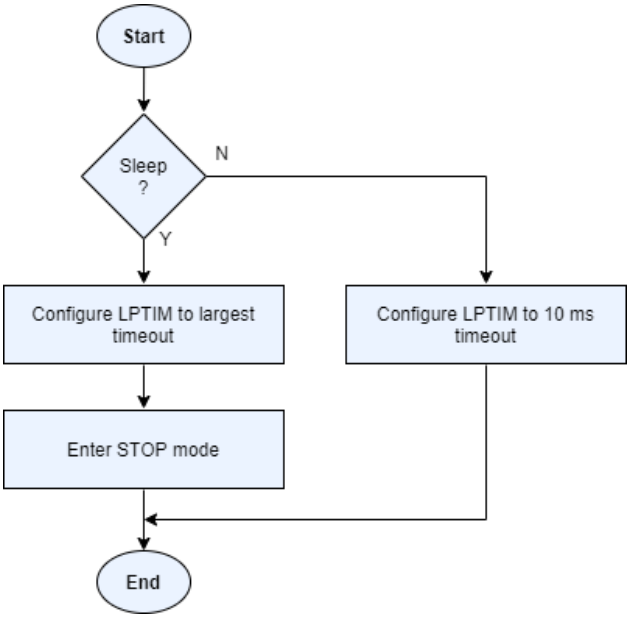


Figure 50: Flowchart of the idle task hook

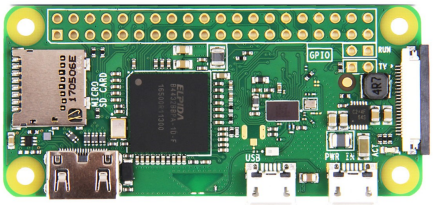


Figure 51: Raspberry Pi Zero W single board computer

- **RFM95W** - The SPI and GPIO interfaces are used by the Raspberry Pi in order to connect to the LoRa transceiver (Fig.52). The DIO pins are configured as RxDone / TxDone, RxTimeout and CadDone, respectively.
- **SIM7000E** - The Raspberry Pi makes use of the UART and GPIO interfaces to communicate with the module (Fig.53).

4.5.2 Tools/COTS

- **Raspbian Lite** - Official supported operating system for the Raspberry Pi. The Lite version is preferred due to the smaller image size and memory usage and the absence of a GUI.
- **Wiring Pi** - GPIO access library for the Raspberry Pi written in C, supporting UART, SPI and other peripherals on the SoC. Its support for GPIO interrupts and fast execution speed make this the favoured library among many others.

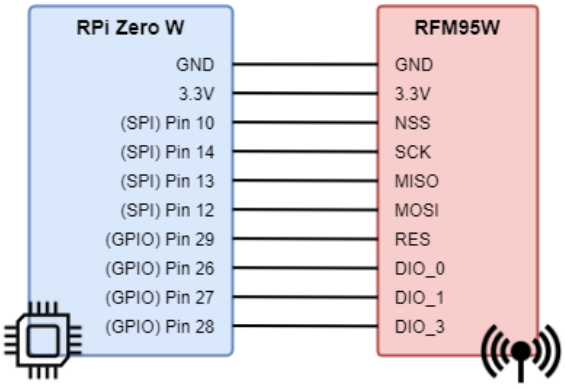


Figure 52: RFM95W connection diagram on the edge device

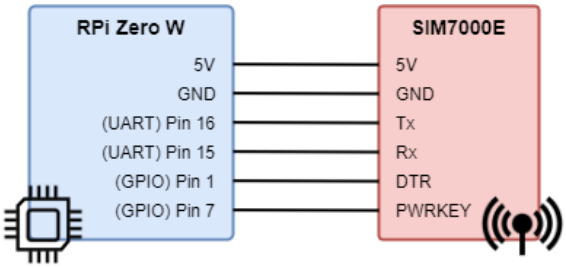


Figure 53: SIM7000E connection diagram on the edge device

4.5.3 Software Architecture

Two tasks are performed by the edge device (Fig.54), related through both sides of the packet forwarding: The CoAP and LoRa threads. The LoRa thread runs the ASFS algorithm and manages the network, receiving the status from the LoRa modem through a message queue, which can be of CAD timeout or detection, reception timeout or successful reception and sending of packets. The processed packets are messaged through a queue to the CoAP thread, running the CoAP client over either Wi-Fi or through the cellular communication module, forwarding the packets to the web application.

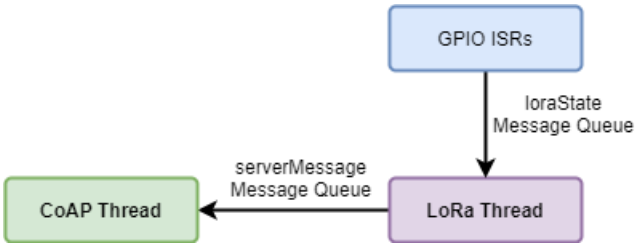


Figure 54: Communication between tasks and interrupts of the standalone edge device

As the end nodes, the application developed for the edge device makes use of C++, using the C++ standard library threads as the multi-threaded environment. Aside from the underlying libraries, the classes were designed as similar as possible to the end nodes, in order to enhance portability between both platforms. Hence, the application can also be regarded as the same bundle of three layers.

4.5.4 Software Detailed Design

Hardware Abstraction

The hardware is also represented with template metaprogramming, enabling access to the GPIO, SPI and UART peripherals (Fig.55).

The GPIO class allows the configuration of both input (with interrupt) and output pins, with the necessary pin functions.

The SPI class enables access to the peripheral, with functions to read and write for both single bytes and in burst mode. It requires the GPIO class to control the NSS pin.

The UART class provides access to the peripheral, along with the necessary send and receive functions, where the latter returns a boolean when no data is available in the serial port.

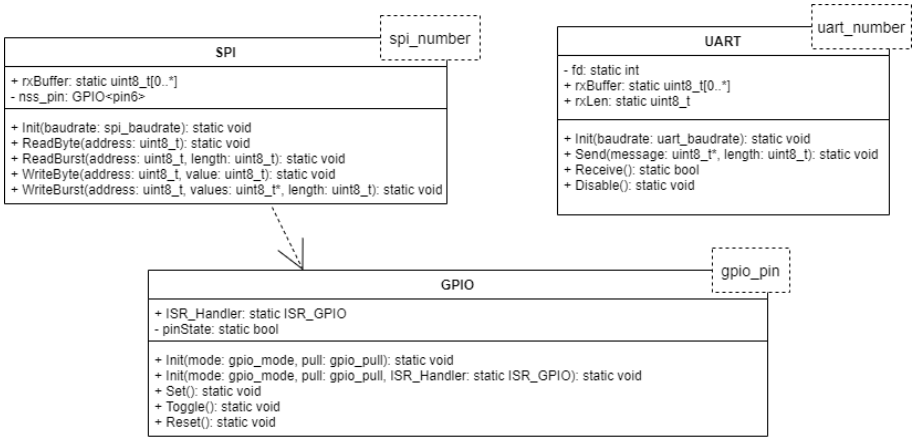


Figure 55: Edge device hardware specific class diagram

Scheduling

Since C++ standard library does not incorporate message queue support, a class was developed to provide its functionality, through the queue STL container and the mutex libraries (Fig.56).

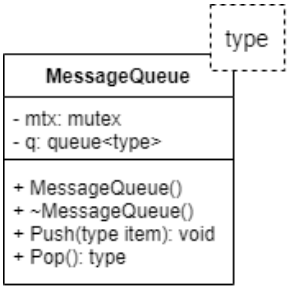


Figure 56: Edge device message queue class diagram

Tasks

The application is concentrated in two threads, which can be regarded as the local network / packet forwarding duo. As stated before, its structure is based upon the same patterns as the application developed for the end nodes.

The LoRa class (Fig.57) implements the local network functionalities, where packets are received, nodes configured and the network is managed. It revolves around the RFM95W transceiver, making use of the necessary peripherals to control the device. Over the peripheral, it also implements packet reception (via the ASFS algorithm), packet sending and network management, by storing the information on the end nodes in a vector container (defined by the EndNode class), along with the SNR of the received packets, to tweak the link between each node and the edge device automatically on a daily basis. Lastly it also includes the configuration and reconfiguration of end nodes, where the latter restores the end node link parameters, if the end node is turned off.

The CoAP class (Fig.58) uses the same CoAP implementation as the standalone end node, differing on the inderlying UDP implementation. Aside from the necessary architectural changes on the Cellular class, there is also provided an UDP class, making use of the Wi-Fi module on the Raspberry Pi and the Linux implementation of the protocol as an alternative.

4.6 Server

4.6.1 Web Application

The web application is divided in seven different pages (Fig.59):

- **Login** - Index page of the website, from which the user can log into the dashboard or go to the register page.
- **Register** - Allows the registration of new users into the application.

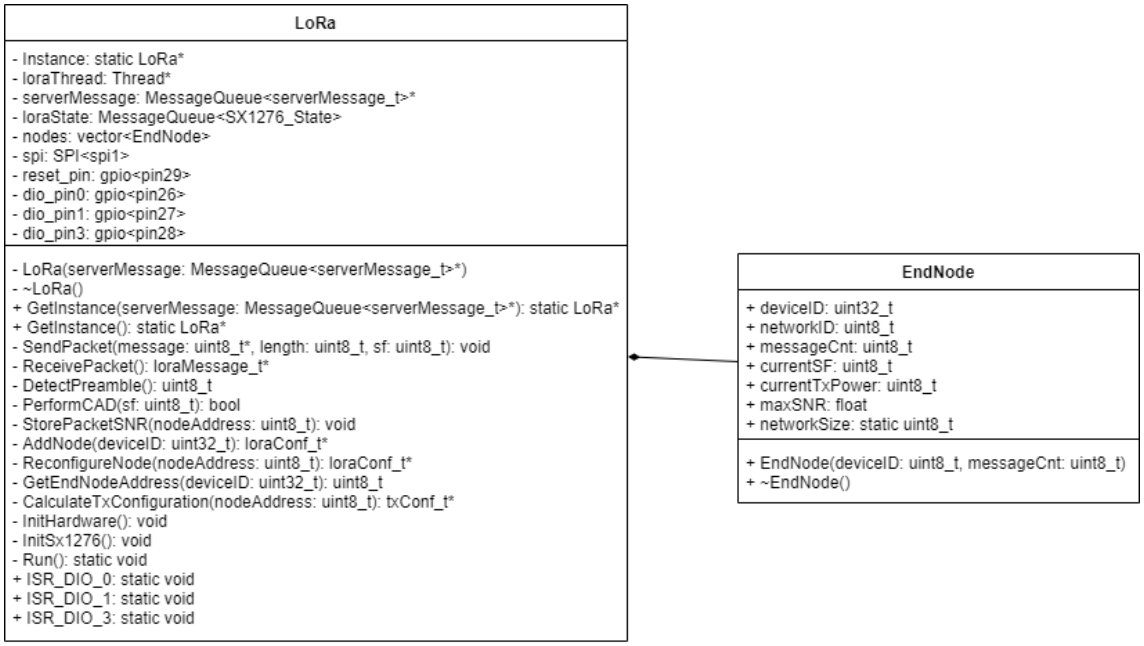


Figure 57: Edge device LoRa thread class diagram

- **Summary** - Provides an insight over the irrigation points, including alerts regarding the irrigation plans or the devices, the location of the irrigation points and the latest irrigation plans. As all pages in the authenticated section, it contains a top bar that provides easy access to all other pages in the application.
- **Detailed View** - Shows a detailed version of a specified field (bundle of irrigation points) / irrigation point, where the irrigation plan is magnified and the weather forecasts for the locations can be observed.
- **Manage Devices** - Enables the management of devices, including the insert, edit and delete operations. Some manageable parameters are the device location, crop, soil, irrigation and association with a field.
- **Manage Fields** - Allows the management of fields, to allow the easy grouping of irrigation points in an arbitrary way entirely up to the farmer's preference.
- **Account Info** - Displays the user information relevant to the application.

4.6.2 Tools/COTS

- **MERN Stack** - Javascript technological stack used to build the SPA web application.

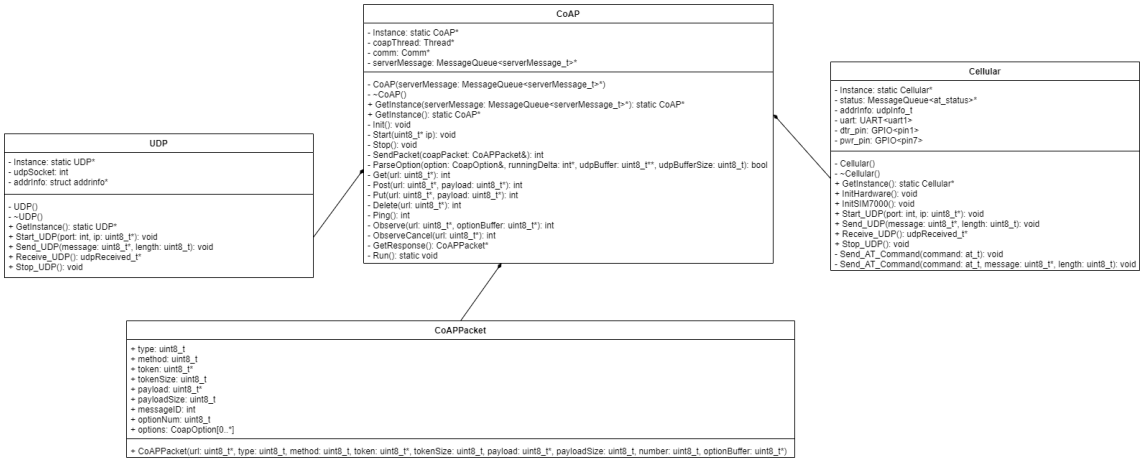


Figure 58: Edge device CoAP thread class diagram

- **Nginx** - Lightweight HTTP server and proxy / reverse proxy server, known for its high performance, stability, simple configuration and low resource consumption.
- **OpenSSL** - Robust, commercial-grade toolkit for the TLS and SSL protocols.
- **Material-UI** - React UI framework that implements Google's Material Design components.
- **Redux** - A predictable state container for Javascript apps, providing a centralized space for the website state, allowing for interesting features such as state persistence. It also contains official bindings with React and powerful debugging capabilities through the Redux DevTools browser extension.
- **Node-CoAP** - CoAP client and server library for Javascript.
- **Web-Push** - Library that implements a push server to dispatch the push notifications to the user.
- **Dark Sky API** - Free weather API, providing current weather conditions, daily forecasts of up to seven days, along with many other features.
- **Open Topo Data API** - Free elevation API based on elevation data sets.
- **Global Self-consistent, Hierarchical, High-resolution Geography Database (GSHHG)** - High resolution geography data set, containing geographical information on shorelines and lakes for the entire world.
- **BabelEdit** - Translation editor for web applications, supported by the React framework, easing the development of multi-language websites.

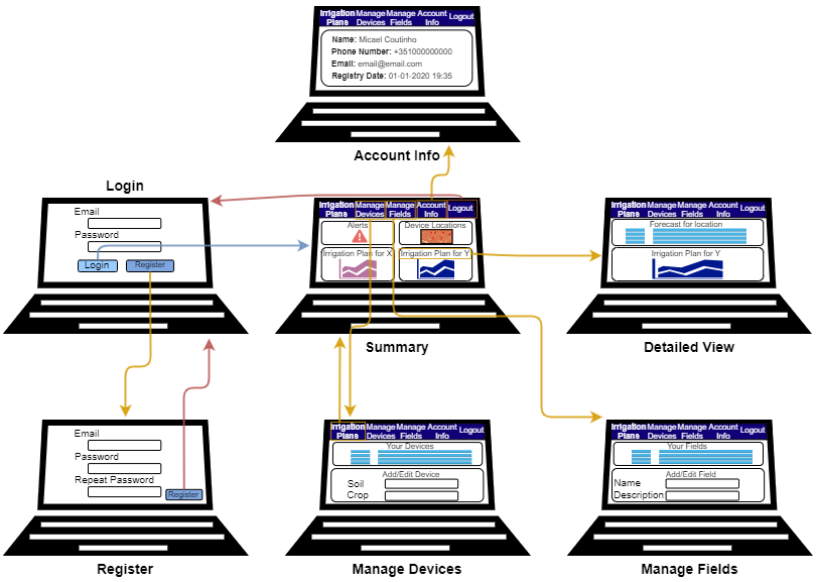


Figure 59: Web application structure and navigation (simplified)

- **Postman** - Platform for API development, allowing easy testing of REST, SOAP and GraphQL requests.

4.6.3 Irrigation Planning Algorithm Interface

In order to perform the irrigation planning, the necessary inputs are required by the optimal control algorithm:

- **Altitude** - Gathered via the Open Topo Data API, when the user configures the location of the irrigation point.
- **Soil Moisture** - Latest soil moisture value retrieved from the sensors in the soil.
- **Soil, crop and irrigation variables -** The coefficients are stored in the database and associated with the irrigation point when configured. There are in total 12 soil, 43 crop and 6 different irrigation options for the user to choose from.
- **Weather Forecast** - Retrieved daily by the server, through the Dark Sky API. Since the weather does not vary in very small location differences, the forecasts are always spaced 0.05 latitude and longitude-wise (7 to 8 kilometers), so that only the necessary forecasts are fetched.
- **Coastline** - Calculated through a Python script when an irrigation point is configured, verifying if the point is located over or under 20 kilometers away from the coast.

The inputs and outputs to the irrigation planning alghorithm can be found below on the Fig.60

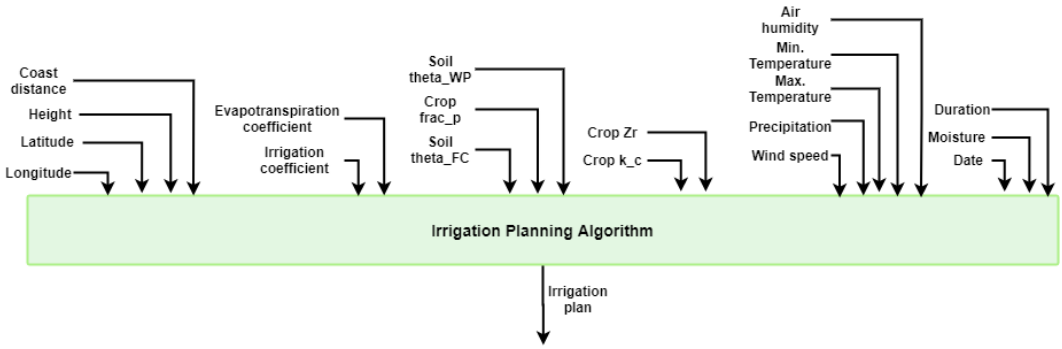


Figure 60: Inputs and outputs of the irrigation planning algorithm

4.6.4 Coastline Detection Algorithm

The flowchart in the Fig.61 specifies how the distance from the coast is calculated. Firstly, it loads a pickle file with the dataset of coastline points for the entire planet. Then, it makes use of the K-Nearest Neighbors algorithm to find the closest coastline point to the device, by feeding the dataset to it and the coordinates of the device. After finding the closes point (neighbor), the distance between them is calculated through the Haversine function, which is more precise than the euclidean distance because it takes into account the curvature of Earth.

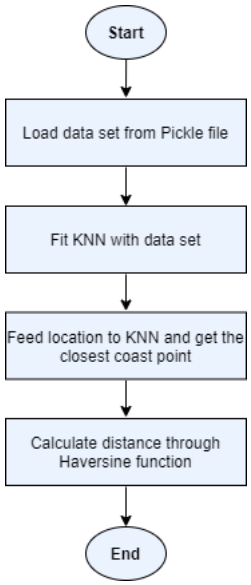


Figure 61: Flowchart of the coastline detection algorithm

4.6.5 Data Management

The web application stores its data in two different manners: a No-SQL database and csv files. The database contains all data necessary for the web application and the irrigation planning algorithm, while the csv files provide easy access to the weather forecasts, that can be used in other research works to further improve the irrigation planning algorithm.

The database (Fig.62) is divided in 8 different collections:

- **Field** - Contains the field information (name, description and owner) used to aggregate multiple related irrigation points.
- **User** - Incorporates the user information, along with the push notification subscription data.
- **Soil** - Includes the soil types and the respective variables that influence the irrigation plans.
- **Irrigation** - Contains the irrigation and its respective efficiency.
- **Crop** - Includes crop information relevant for plan calculation.
- **Plan** - Contains the irrigation plans for an irrigation point, along with the starting date.
- **Device** - Features the device information, such as the type, location, height, coast distance and moisture, along with references to the respective crop, irrigation, soil, owner and if there is an issue with the device.
- **Forecast** - Includes the weather forecast for a respective location, over the course of a week, necessary for plan calculation. The relevant variables are temperature, wind speed, air humidity and liquid precipitation rate.

The database is updated with new entries through the back-end API's, which are triggered by the front-end, when an user performs an action, daily, when the optimal irrigation planning algorithm runs or the weather forecast is acquired or when the devices in the fields send soil moisture information.

The csv files are separated into two different categories, per location and forecast variable:

- **Observations** - Record of the values observed in each day by the weather forecast API.
- **Forecasts** - Information regarding the evolution of a forecasted variable for a respective day, throughout the span of seven days.

The csv files are updated daily through the back-end API that fetches the weather forecast, while performing the irrigation planning activity. When the forecasts are fetched, the values for the respective day are stored in a file, for each weather-related variable. The forecasts for each day are also stored, when it makes part of the eight day interval of the forecast. The files are stored for each location, where the former belong to the observations folder and the latter to the forecasts.

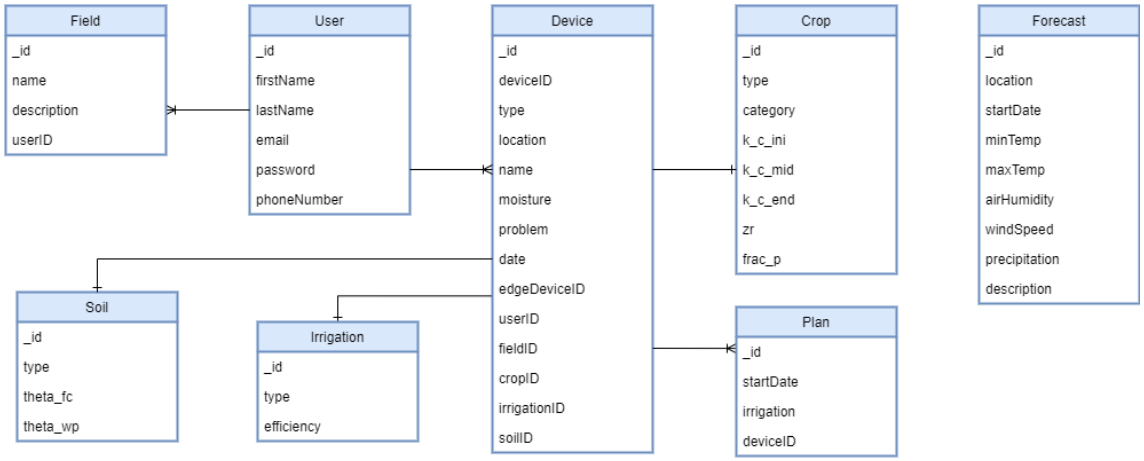


Figure 62: Database entity-relationship diagram

4.6.6 Backend Endpoints

The web API endpoints provided by the back end can be split into five different categories, according to their goal.

The API's for the web application include all the necessary ones to deliver and retrieve information to and from the website (Tab.3):

The API's for the optimal irrigation planning algorithm connect the engine to the web application (Tab.4):

The API's for weather forecast fetch and arrange the weather forecast data (Tab.5):

Table 5: API endpoints of the web application related to the weather forecast

Method	URI	Description
GET	/forecastprovider	Get the weather forecast from the weather API and store it

The notification API's bring the push server to life, which works with the notifications service worker to send push notifications to the farmer (Tab.6):

Table 3: API endpoints of the web application related to the website

Method	URI	Description
GET	/crops	Get crops from the database
GET	/forecast?location={l},{l2}	Get weather forecast for the specified locations from the database
GET	/irrigations	Get irrigations from the database
POST	/login	Verify the login information and emit a JSON web token to log in the user
POST	/register	Encrypt the password and register the new user in the database
GET	/soils	Get soils form the database
GET	/user/{id}/account	Get the user account information from the database
GET	/user/{id}/devices	Get the user devices from the database
POST	/user/{id}/device	Insert a new device in the database
DELETE	/user/{u_id}/device/{d_id}	Remove a device from the database
PUT	/user/{u_id}/device/{d_id}	Edit the location, field, crop, soil and irrigation of a device
GET	/user/{id}/fields	Get the user fields from the database
POST	/user/{id}/field	Insert a new field in the database
DELETE	/user/{u_id}/field/{f_id}	Delete a field from the database and unlink its devices
PUT	/user/{u_id}/field/{f_id}	Edit the name and description of a field
GET	/user/{id}/plans	Get the irrigation plans from the user devices in the database

Table 6: API endpoints of the web application related to the push notifications

Method	URI	Description
POST	/subscription/{id}	Create push notification subscription for the user and store it in the database
GET	/subscription/{id}	Send a push notification to the user through the push server

The maintenance API's clean up the oldest entries from the database, so that the database size is kept small (Tab.7):

Table 4: API endpoints of the web application related to the irrigation algorithm

Method	URI	Description
POST	/plans	Calculate the irrigation plans for one user devices or all devices

Table 7: API endpoints of the web application related to database maintenance

Method	URI	Description
DELETE	/devices/old	Delete device entries older than a month
DELETE	/plans/old	Delete irrigation plans older than a month
DELETE	/forecasts/old	Delete weather forecasts older than a month

5. System Implementation

5.1 Local Network

With a pre-established maximum ToA and the preamble lengths for each SF, all other variables can be tweaked for optimal performance. In this case, the physical layer LoRa parameters were calculated by creating a Python script, which relates them to the ToA and link budget (Tab.8).

Table 8: LoRa communication parameters

Parameter	Value
BW	41.7kHz
CR	4/5
Header Mode	Explicit
CRC	On

The aforementioned parameters provide different results according to the spreading factor in use, for an 8 byte payload and a transmission power of 14dBm (Tab.9):

Table 9: LoRa communication results for an 8 byte payload

SF	Preamble Length	ToA	Link Budget	Tmin Between Messages
7	127	481.56ms	138.8dB	0:48
8	64	569.76ms	142.2dB	0:57
9	32	677.46ms	145.1dB	1:08
10	32	1354.93ms	148dB	2:13
11	16	1878.56ms	150.3dB	3:08
12	12	3364.22ms	152.8dB	5:36

The aforementioned configuration (Tab.9) yields a maximum link budget of 152.8dB. As for comparison, the LoRaWAN protocol, which in its strongest configuration in Europe (SF=12, BW=125kHz) grants a total link budget of 148dB. This bigger link budget enables a longer range of communication, at the cost of a longer ToA.

5.2 End Nodes

5.2.1 Hardware Deployment

The outer shell of the WSN (Fig.63 - left) and standalone (Fig.63 - right) end nodes contains the antenna for the transceiver (respectively for LoRa or cellular communication), the connected sensors and a solar panel, for energy harvesting. Their casing measures 120mm of width, 200mm of length and 90mm of height and the material is ABS plastic, characterized by its sturdiness and hardness.

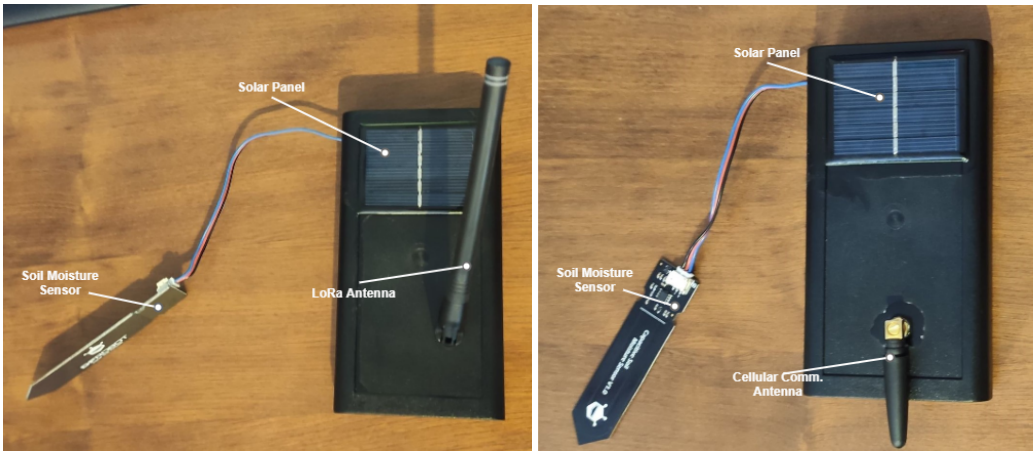


Figure 63: End nodes hardware deployment

On the inside (Fig.64), the PCB, battery and step-up converter can be found on the bottom, while the transceiver is at the top.

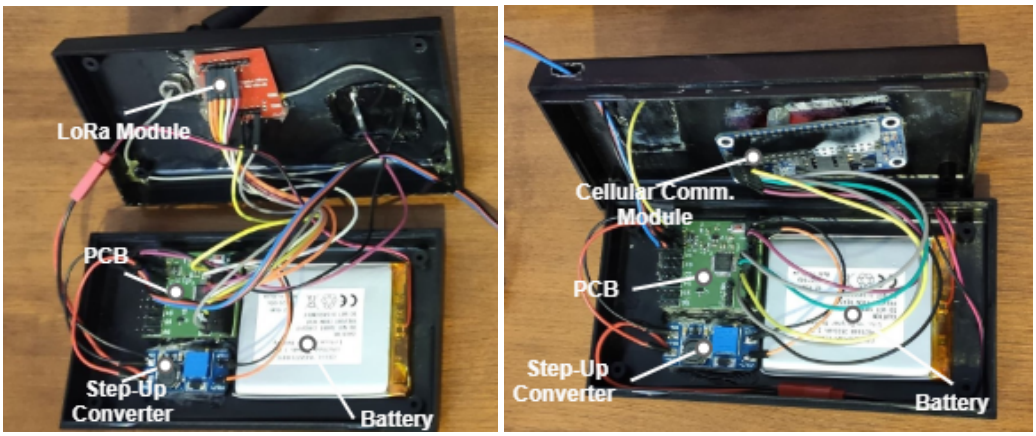


Figure 64: End nodes hardware deployment (inside view)

The resulting PCB (Fig.65) can be observed in greater detail below:

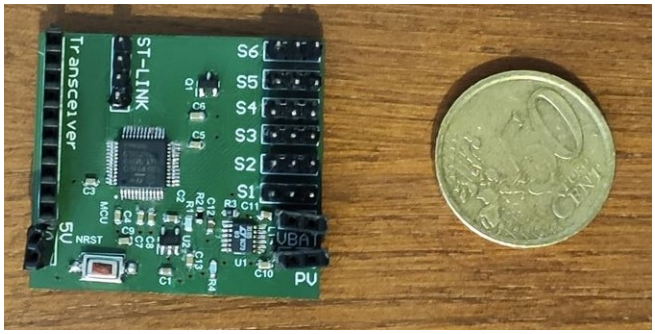


Figure 65: Developed PCB for the end nodes

5.2.2 Tests and Results

The communication capabilities of the end nodes are better observed on their receptors, the edge device and web application, respectively. As such, the communication tests performed on the WSN end node can be observed in the Edge Device and the standalone end node on the Web Application sections of this chapter.

Regarding the system autonomy, the most important variable is the power drawn by the end nodes (Tab.10), which shows how much energy is needed for the system to become self-sustainable with the solar energy harvesting and how long the nodes can be deployed without it.

Table 10: End nodes autonomy estimation without energy harvesting

End Node	Mean Current	Autonomy Without Energy Harvest
Standalone (GPRS)	4.71 mA	88.46 days (10 000 mAh)
WSN (SF=7)	251.65 μA	579.5 days (3500 mAh)
WSN (SF=8)	252.94 μA	576.55 days (3500 mAh)
WSN (SF=9)	254.24 μA	573.6 days (3500 mAh)
WSN (SF=10)	263.24 μA	553.83 days (3500 mAh)
WSN (SF=11)	269.8 μA	540.52 days (3500 mAh)
WSN (SF=12)	289.25 μA	504.18 days (3500 mAh)

To calculate the energy used by the nodes when in active mode (performing the main routine), the microcontroller current in run and stop modes was summed to the power drained from six sensors and the LoRa module when in sending, idle and sleep modes (for the multiple spreading factors) and multiplied by the time in that mode. The same was done for the standalone end node with GPRS capabilities, differing on the transceiver and microcontroller currents. The power consumption was not calculated for the NB-IoT standalone node, since the technology was unavailable for testing. For the energy spent sleeping, the procedure is similar, where the currents taken into account are for

the respective sleeping state. The resulting autonomy is an estimate for the number of days the corresponding battery can power the end nodes in a single charge, without considering the energy harvesting capabilities of the system.

An experiment was also conducted to validate the theoretical values, by placing each of the nodes with a 3500 mAh and monitoring their battery voltage before and after 12 hours. The discharge curve was then taken into account to estimate the battery discharged. The solar panel was also disconnected. From these results (Tab.11), the power consumption of WSN end node is very small, as the calculations stated. Regarding the standalone end node, its consumption was slightly higher than expected. This can be due to the power and network status LED's in the SIM7000E module, which are always on and blinking, respectively. Another reason is that the DTR pin output needs to stay high in order for the module to be in sleep mode.

Table 11: End nodes autonomy experiment without energy harvesting

End Node	Initial Voltage	Voltage after 12 hours	Estimated Discharge
Standalone (GPRS)	4.06 V	3.86 V	5 %
WSN (SF=12)	4.12 V	4.12 V	0 %

From these results, we can verify that the WSN end node requires a smaller battery than the standalone end node, due to the energy spent by the module when using GPRS/UMTS. By switching to a NB-IoT card, the value is expected to drastically decrease, due to a smaller current when sending and in the sleep mode (caused by the PSM mode available on NB-IoT).

5.3 Edge Device

5.3.1 Hardware Deployment

The outside of the edge device (Fig.66 - left) contains the antennas for both the LoRa and the cellular communication modules, along with the 5V power supply. On the inside (Fig.66 - right), the Raspberry Pi Zero W can be found on the bottom, connected to the power supply. On the top, both the LoRa and cellular communication modules are present. Its casing is the same as the one found on the end nodes.

Lastly, the preferred location to deploy the edge device is on a slightly higher ground than the end nodes, as close to them and as further away from obstacles as possible, specially if the end nodes are at a considerable distance.

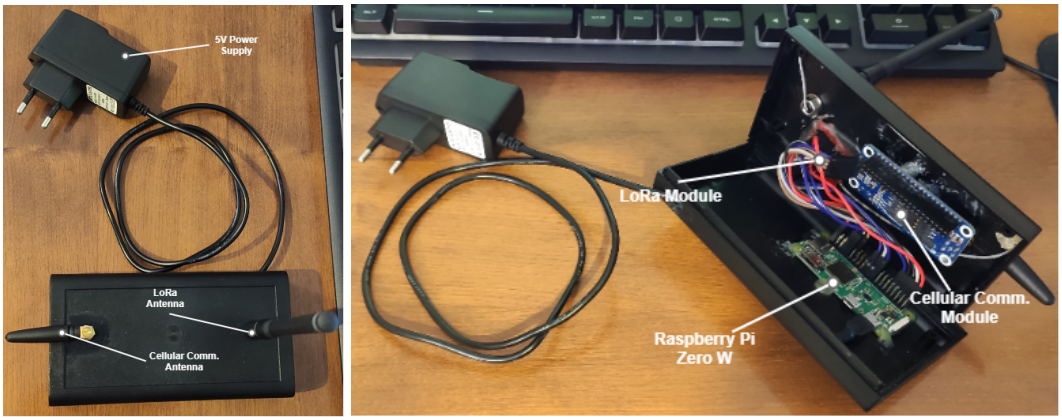


Figure 66: Edge device hardware deployment

5.3.2 Tests and Results

Regarding the end node, the most important tests performed were the end node communication / configuration and the packet forwarding capabilities, where the latter is shown on the Web Application section.

The end node communication/configuration test (Fig.67) involves the configuration of a new end node on the local network and the reception of a soil moisture packet. It can be observed on a terminal, connected to the Raspberry Pi via SSH and running the app manually. The process starts when the end node sends a join message, which is received by the end node by continuously performing CAD detection and notifying the LoRa thread through a message queue. After the ASFS algorithm detects a valid preamble three times, the packet is received at the found spreading factor and parsed. Then, the edge device computes the necessary join parameters and sends them to the end node, saving the configuration. Later on, a soil moisture packet is sent, which the edge device forwards to the CoAP server and stores the SNR of the received packet, to later control the communication parameters.

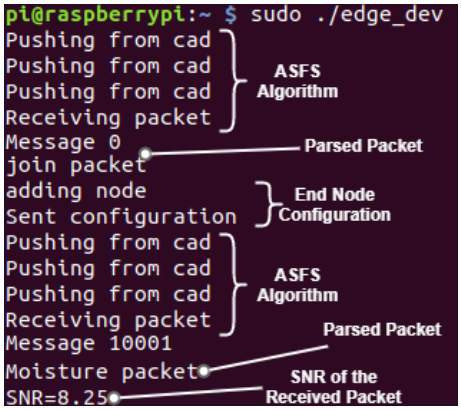


Figure 67: Edge device LoRa communication results

5.4 Server

5.4.1 Web Application

The web application was developed with multiple platforms in mind, making it practical to use from smartphones to desktop computers. After authenticating, the user is redirected to the summary tab (shown in Fig.68, where the left version is for desktop and the right for mobile phones), being able to observe alerts related to the devices, the location of the irrigation points on the map, along with a table summarizing them, and the irrigation plans for each point, grouped by field. By clicking on a device name, the user accesses the detailed view, which specifies the forecast for the location of the point, along with a broader plot of the irrigation plan (which can also be updated through the push of a button). On the manage fields tab, the user is allowed to create fields and group irrigation points to it, editing and removing them. On the device management tab, the user can edit, remove or add devices. For the end nodes, the user can specify the crop, irrigation and soil types, which are taken into account for the plan calculation.

Figure 68: Web application summary page

When clicking in an irrigation point or crop field, the user is directed to the detailed view (Fig.69). In it, there is detailed information on the irrigation points that are associated with the crop field, the irrigation plans, which are absent when problem occur, such as loss of communication between the server and the irrigation point, and the weather forecast for the respective device locations.

In the Fig.70, we can take a closer look at the device management. On top, the list of devices is shown, which allows for the editing or removal or each one. It is followed by a form for adding a new device, where crop, field and soil types are inserted, along with the ID of the new device and its location.

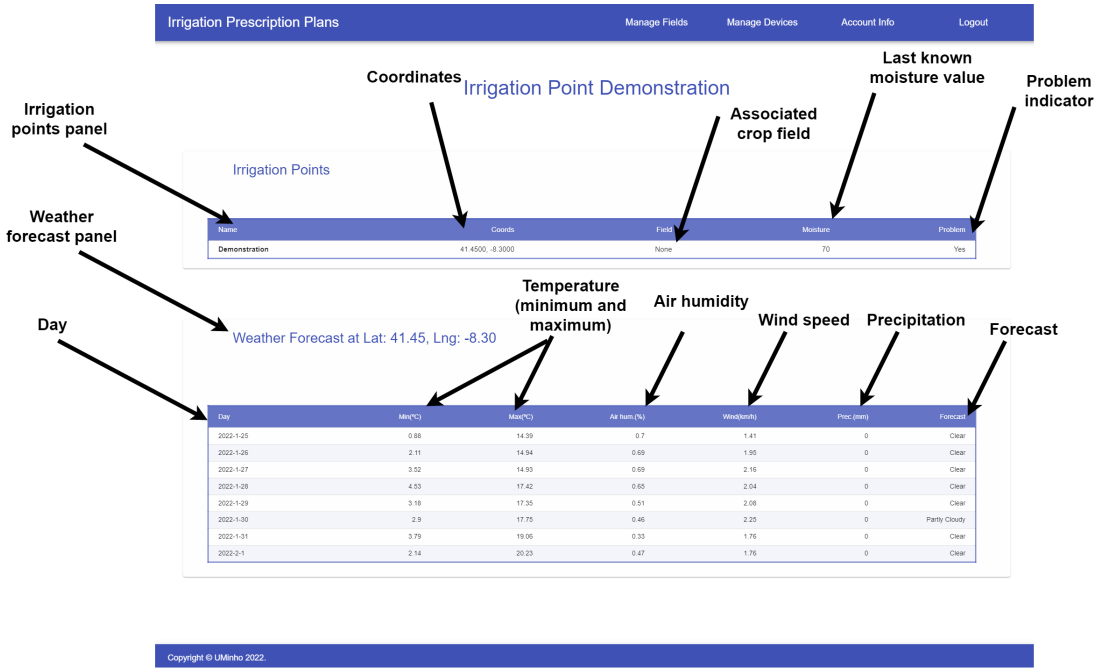


Figure 69: Web application detailed view page

Besides the design, some important features of the web application are:

- Available in multiple languages (Portuguese and English). Other languages can be easily added.
- Persistent client storage of the application data, only updated once a day or when the user executes an operation that requires it.
- Optimized for both desktop and handheld devices.
- Push notification alerts.

5.4.2 Irrigation Planning Algorithm

In order to run the irrigation planning algorithm from the server, the Octave script has to be called within the back-end, passing the necessary arguments for each irrigation point and then retrieving the results. Firstly, the Octave package was installed on the server. Then, the child_process module from the Node API was used to call the irrigation planning algorithm with the necessary inputs. Lastly, the script was adapted to parse the command line arguments and print the resulting irrigation plan for each day when the planning is performed.

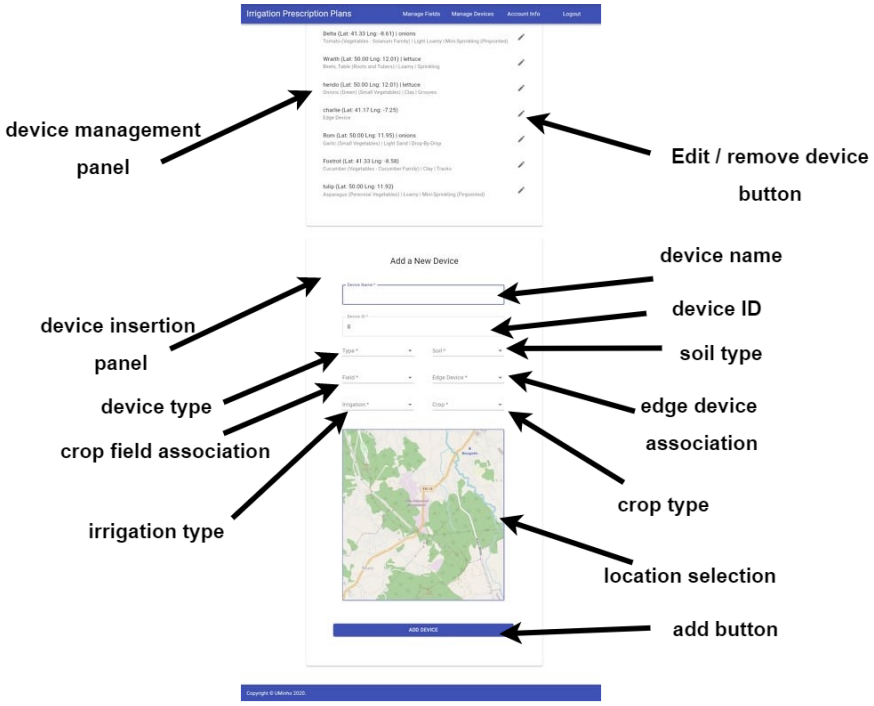


Figure 70: Web application device management page

5.4.3 Coastline Detection Algorithm

Before the coastline detection was developed, a Python script was developed to convert the world coastline dataset into a more manageable set of coordinates. To examine the trade-off between the coastline precision and the time required to perform a detection, five datasets with different coastline resolutions were converted.

The conversion script was used to read the shape files for each of the five resolutions of coastline and converting them into a set of points, through the help of the Cartopy package. Then, each of the resolutions was inspected using Matplotlib, plotting them over the Portuguese coastline (see Fig.71), and saved through the Pickle package, to be used for coastline detection.

The Python script for coastline detection makes the inference of the K-Nearest Neighbors algorithm, using the Scikit-Learn implementation, according to the Pickle file generated by the conversion script. Then, the first neighbor is found and the Haversine function is used to calculate the distance, using the Math library. The returned result is true or false, according to the defined distance threshold.

In the end, the conversion of the intermediate resolution dataset was chosen, due to the time necessary to perform the coastline detection script on a personal computer and the satisfying results it can provide, as shown in Tab.12. Like the planning algorithm, the coastline detection script is interfaced with by the back-end through the child process library of Node.js.

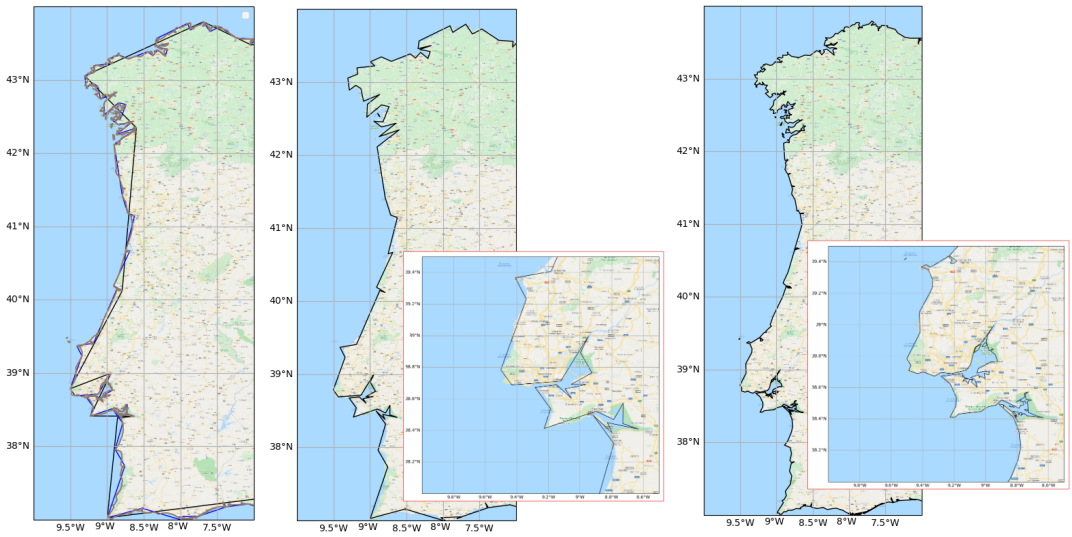


Figure 71: Coastlines for all (left), low (middle) and intermediate (right) resolutions

Table 12: Number of points, pickle file size and execution time for each coastline data set resolution

Resolution	Number of Points	Pickle File Size	Execution Time
Crude	~7 700	185kB	<0.1s
Low	~50 000	1.4MB	<0.1s
Intermediate	~340 000	8.6MB	~0.7s
High	~1 600 000	41.3MB	~4.5s
Full	~9 500 000	240MB	~35s

5.4.4 Tests and Results

The packet forwarding capabilities of the edge device and the standalone end node (Fig.72) can be observed on the server, again through a SSH connection to a computer. The packet received by the server contains in its URI the end node device ID. On the payload, there is the soil moisture of the respective end node.

Besides the communication capabilities of the back-end, the alerting, weather forecast fetching and irrigation planning capabilities were also assessed.

To observe the irrigation alerts in action (Fig.73), the website connection with the user needs to be secured with HTTPS. When the irrigation planning is performed and either a device has stopped communicating or the irrigation plan shows a dangerous condition for the crops, a push notification is sent and it will be available as soon as the user connects to the internet. By clicking the notification, the user is directed to the website, where the issues can be observed in detail.

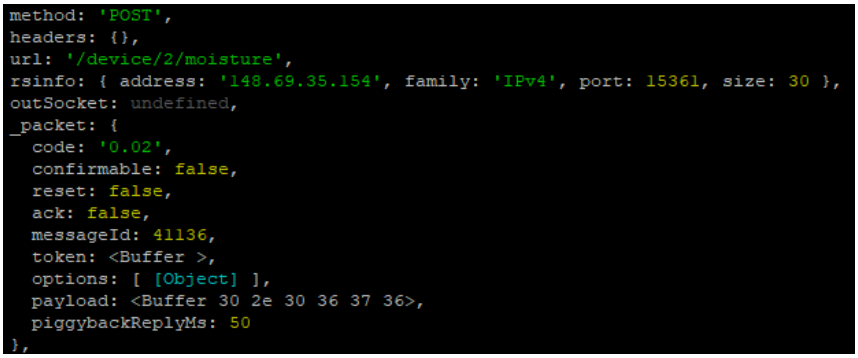


Figure 72: CoAP results for the standalone end node

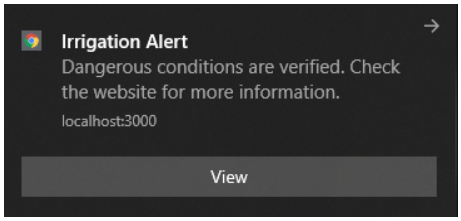


Figure 73: Web application push notification results

The results from fetching the weather forecast can be observed in both csv files, present in the server and in the cloud database. In the former, the observations and forecasts directories for each location can be found (Fig.74 - left). Inside them, the observed and forecasted values for each variable are present (Fig.74 - right).

In the database, the weather forecasts are present for each location (Fig.75), which from there are used by both the irrigation planning algorithm and the web application.

In the database, the irrigation are present for each irrigation point (Fig.76), by containing the deviceID to identify which device it corresponds to and the starting date of the plan.From there, it can be displayed to the user in the form of a plot and a table, in the web application.

5.4.5 Deployment

To deploy the web application on the server computer, Nginx was used alongside with OpenSSL, to generate the SSL certificates in order to establish a HTTPS connection, which is mandatory since the Google location API and the push notification service require it.

Besides the generation of self-signed certificates, an optimized production build of the React front-end was deployed. Then, Nginx was configured to listen to the port 443 (HTTPS) with the necessary certificates, serve the website and create a proxy to the port 4000, where the Node.js back-end has been deployed under the PM2 process manager.

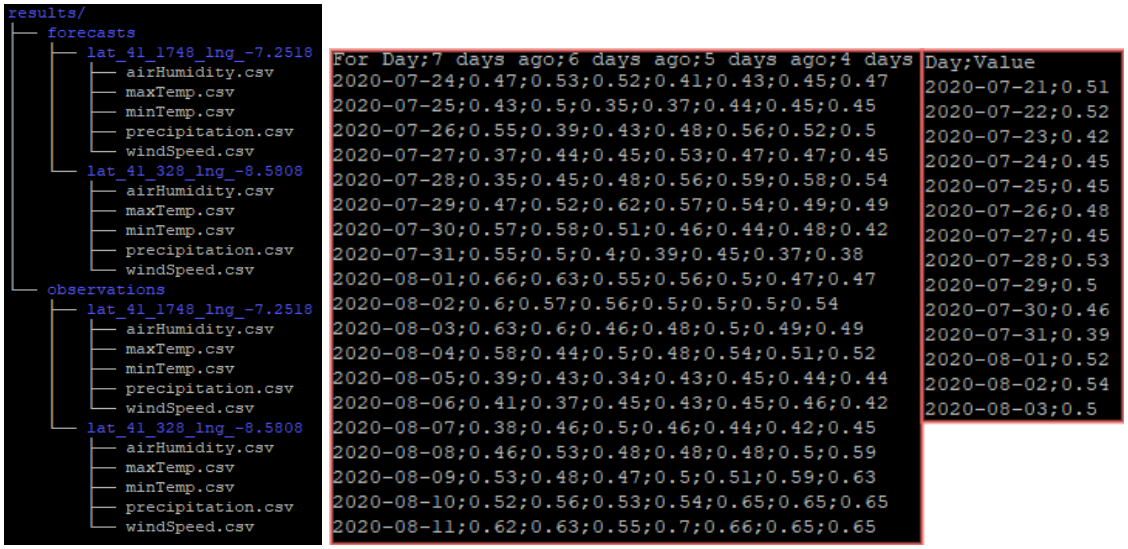


Figure 74: Generated weather storage directories and files

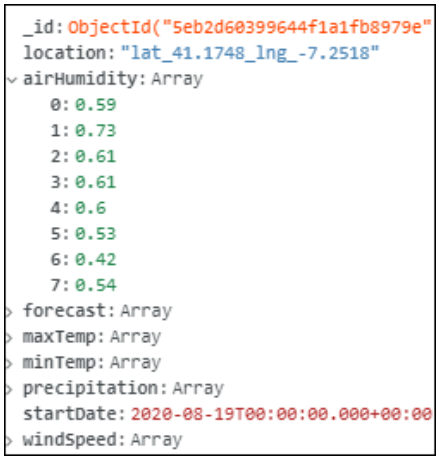


Figure 75: Weather forecast database storage

Lastly, both the Octave irrigation planning algorithm and the Python script were copied to the server, along with the necessary dependencies to make them work, including the installation of Octave in the server for the former and python packages for the latter.

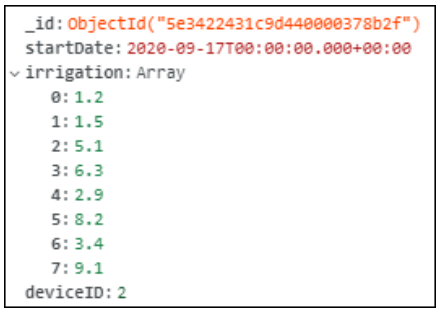


Figure 76: Irrigation plans database storage

6. Conclusion

6.1 Conclusions

The main goal of this work was to develop a solution capable of prescribing irrigation plans for agricultural soils, based on the irrigation planning algorithm developed in [1]. Multiple steps were taken in order to bring this system to life:

- Research the latest irrigation monitoring systems from multiple standpoints, such as the sensing technology employed, the network technology applied, the form of interaction with the farmer and the monitoring mechanism used;
- Establishment of the requirements and the architecture of the system according to the previous research of similar systems;
- Design of the system according to the research performed on the state of the art technologies and the outlined architecture;
- Implementation of the designed system, resorting to the selected technologies;
- Undertaking of unit, functional, integration and end-to-end tests on the developed systems and extraction of conclusions and future improvements involving the developed work.

The developed system is composed by a network of irrigation points, that retrieve the soil moisture and a web application which receives the soil moisture data from the network, retrieves the weather forecasts from an API and performs the irrigation planning. Besides it stores the relevant data and provides the user with a web interface, from where plans can be observed, devices and crops fields can be managed and the farmer can be alerted, when dangerous conditions are verified.

The resulting system is cost-conservative, when compared to the systems studied. The production cost for each node is located around 100€ (for 1 sensor), while the edge device costs 110€ to replicate. When comparing to the products studied in the second chapter, this value is around 40 percent lower. Lastly, the solution which can be applied in multiple scenarios, such as farms, homes or even by city halls to irrigate public parks and gardens.

The developed work consisted of the design and implementation of hardware (namely the PCBs for the field devices), communications (with the LoRA local network above the physical layer), and a modern web application. The work in these three different fields is based on state of the art

technologies. Besides the contributions that the developed system is as a whole, some more specific contributions are worthy of pointing out:

- An energy saving scheme controlling wake up and sleep modes using FreeRTOS on the STM32 platform;
- A template-based hardware abstraction layer that simplifies the development of C++ in these microcontrollers, while using scheduling with FreeRTOS;
- A LoRa data link/network layer that balances communication range and energy consumption, while using a single transceiver as the concentrator;
- An algorithm that calculates the closest distance to the coast for any point in the planet.

6.2 Future Work

Even though all the requirements of the system are fulfilled, some improvements can be performed in the system, in order to improve the developed solution, such as:

- Further explore the NB-IoT technology for the standalone end nodes, in order to improve their energy autonomy;
- Develop an actuator to irrigate the system automatically according to the plans, closing the loop in the system;
- Further tweaking of the LoRa network developed, in order to enhance its performance;
- Secure the CoAP and LoRa-based communications, using DTLS and AES-128 encryption, respectively;
- Verify the necessity of soil-specific calibration.

Bibliography

[1] S. F. Lopes, R. M. S. Pereira, S. O. Lopes, M. Coutinho, A. Malheiro and V. Fonte, "Yet a smarter irrigation system", pp. 337–346, 2020. doi: 10.1007/978-3-030-51005-3_28.

[2] S. Kim, H. Lee and S. Jeon, "An adaptive spreading factor selection scheme for a single channel lora modem", Sensors (Switzerland), vol. 20, no. 4, 2020, issn: 14248220. doi: 10.3390/s20041008.

[3] F. Liedmann, C. Holewa and C. Wietfeld, "The radio field as a sensor-A segmentation based soil moisture sensing approach", 2018 IEEE Sensors Applications Symposium, SAS 2018 - Proceedings, vol. 2018-Janua, no. March, pp. 1–6, 2018. doi: 10.1109/SAS.2018.8336755.

[4] H. M. Jawad, R. Nordin, S. K. Gharghan, A. M. Jawad, M. Ismail and M. J. Abu-Alshaeer, "Power reduction with sleep/wake on redundant data (SWORD) in a wireless sensor network for energy-efficient precision agriculture", Sensors (Switzerland), vol. 18, no. 10, 2018, issn: 14248220. doi: 10.3390/s18103450.

[5] F. Capraro, S. Tosetti, F. Rossomando, V. Mut and F. V. Serman, "Web-based system for the remote monitoring and management of precision irrigation: A case study in an arid region of Argentina", Sensors (Switzerland), vol. 18, no. 11, 2018, issn: 14248220. doi: 10.3390/s18113847.

[6] V. M. Juan Núñez, R. Faruk Fonthal and L. M. Yasmín Quezada, "Design and Implementation of WSN and IoT for Precision Agriculture in Tomato Crops", 2018 IEEE ANDESCON, ANDESCON 2018 - Conference Proceedings, 2018. doi: 10.1109/ANDESCON.2018.8564674.

[7] S. Katyara, M. A. Shah, S. Zardari, B. S. Chowdhry and W. Kumar, "WSN Based Smart Control and Remote Field Monitoring of Pakistan's Irrigation System Using SCADA Applications", Wireless Personal Communications, vol. 95, no. 2, pp. 491–504, 2017, issn: 1572834X. doi: 10.1007/s11277-016-3905-5.

[8] M. F. Işik, Y. Sönmez, C. Yilmaz, V. Özdemir and E. N. Yilmaz, "Precision Irrigation System (PIS) using sensor network technology integrated with IOS/Android Application", Applied Sciences (Switzerland), vol. 7, no. 9, 2017, issn: 20763417. doi: 10.3390/app7090891.

[9] F. Viani, M. Bertolli, M. Salucci and A. Polo, "Low-Cost Wireless Monitoring and Decision Support for Water Saving in Agriculture", IEEE Sensors Journal, vol. 17, no. 13, pp. 4299–4309, 2017, issn: 1530437X. doi: 10.1109/JSEN.2017.2705043.

[10] M. N. V. Juan, F. R. Faruk and Y. M. Quezada, “Design and implementation of WSN for precision agriculture in white cabbage crops”, Proceedings of the 2017 IEEE 24th International Congress on Electronics, Electrical Engineering and Computing, INTERCON 2017, pp. 1–4, 2017. doi: 10.1109/INTERCON.2017.8079671.

[11] P. Visconti, C. Orlando and P. Primiceri, “Solar powered WSN for monitoring environment and soil parameters by specific app for mobile devices usable for early flood prediction or water savings”, EEEIC 2016 - International Conference on Environment and Electrical Engineering, 2016. doi: 10.1109/EEEIC.2016.7555638.

[12] G. Mitralexis and C. Goumopoulos, “Web based monitoring and irrigation system with energy autonomous wireless sensor network for precision agriculture”, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9425, pp. 361–370, 2015, issn: 16113349. doi: 10.1007/978-3-319-26005-1_27.

[13] J. Gutierrez, J. Villa Medina, A. Nieto-Garibay and M. Porta-Gándara, “Automated irrigation system using a wireless sensor network and gprs module”, Instrumentation and Measurement, IEEE Transactions on, vol. 63, pp. 166–176, Jan. 2014. doi: 10.1109/TIM.2013.2276487.

[14] J. Jao, B. Sun and K. Wu, “A prototype wireless sensor network for precision agriculture”, Proceedings - International Conference on Distributed Computing Systems, no. July, pp. 280–285, 2013. doi: 10.1109/ICDCSW.2013.10.

[15] R. Zhang, L. Chen, J. Guo, Z. Meng and G. Xu, “An energy-efficient wireless sensor network used for farmland soil moisture monitoring”, IET Conference Publications, vol. 2010, no. 575 CP, pp. 2–6, 2010. doi: 10.1049/cp.2010.1017.

[18] A. Islam, K. Akter, N. J. Nipu, A. Das, M. Mahbubur Rahman and M. Rahman, “IoT Based Power Efficient Agro Field Monitoring and Irrigation Control System : An Empirical Implementation in Precision Agriculture”, 2018 International Conference on Innovations in Science, Engineering and Technology (ICISET), no. October, pp. 372–377, 2019. doi: 10.1109/iciset.2018.8745605.

[19] R. Mulenga, J. Kalezhi, S. K. Musonda and S. Silavwe, “Applying Internet of Things in Monitoring and Control of an Irrigation System for Sustainable Agriculture for Small-Scale Farmers in Rural Communities”, 2018 IEEE PES/IAS PowerAfrica, PowerAfrica 2018, pp. 841–845, 2018. doi: 10.1109/PowerAfrica.2018.8521025.

[20] A. A. Alfin and R. Sarno, “Soil irrigation fuzzy estimation approach based on decision making in sugarcane industry”, Proceedings of the 11th International Conference on Information and Communication Technology and System, ICTS 2017, vol. 2018-Janua, pp. 137–142, 2018. doi: 10.1109/ICTS.2017.8265659.

[21] K. Sirohi, A. Tanwar, Himanshu and P. Jindal, "Automated irrigation and fire alert system based on hargreaves equation using weather forecast and ZigBee protocol", 2nd International Conference on Communication, Control and Intelligent Systems, CCIS 2016, pp. 13–17, 2017. doi: 10.1109/CCIntelS.2016.7878191.

[22] T. Savic and M. Radonjic, "Proposal of solution for automated irrigation system", 24th Telecommunications Forum, TELFOR 2016, no. Tiar, pp. 115–118, 2017. doi: 10.1109/TELFOR.2016.7818867.

[23] I. Mat, M. R. Mohd Kassim, A. N. Harun and I. Mat Yusoff, "IoT in Precision Agriculture applications using Wireless Moisture Sensor Network", ICOS 2016 - 2016 IEEE Conference on Open Systems, pp. 24–29, 2017. doi: 10.1109/ICOS.2016.7881983.

[24] R. L. Pascual, D. M. R. Sanchez, D. L. E. Naces and W. A. Nunez, "A Wireless Sensor Network using XBee for precision agriculture of sweet potatoes (Ipomoea batatas)", 8th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management, HNICEM 2015, no. December, 2016. doi: 10.1109/HNICEM.2015.7393212.

[25] N. Hema and K. Kant, "Local weather interpolation using remote AWS data with error corrections using sparse WSN for automated irrigation for Indian farming", 2014 7th International Conference on Contemporary Computing, IC3 2014, pp. 478–483, 2014. doi: 10.1109/IC3.2014.6897220.

[26] J. Co, F. J. Tiausas, P. Aldrin Domer, M. L. Guico, J. Claro Monje and C. Oppus, "Design of a Long-Short Range Soil Monitoring Wireless Sensor Network for Medium-Scale Deployment", IEEE Region 10 Annual International Conference, Proceedings/TENCON, vol. 2018-Octob, no. October, pp. 1371–1376, 2019, issn: 21593450. doi: 10.1109/TENCON.2018.8650541.

[27] T. Savić and M. Radonjic, "WSN architecture for smart irrigation system", 2018 23rd International Scientific-Professional Conference on Information Technology, IT 2018, vol. 2018-Janua, pp. 1–4, 2018. doi: 10.1109/SPIT.2018.8350859.

[28] N. Fahmi, S. Huda, E. Prayitno, M. U. H. A. Rasyid, M. C. Roziqin and M. U. Pamenang, "A prototype of monitoring precision agriculture system based on WSN", 2017 International Seminar on Intelligent Technology and Its Application: Strengthening the Link Between University Research and Industry to Support ASEAN Energy Sector, ISITIA 2017 - Proceeding, vol. 2017-Janua, pp. 323–328, 2017. doi: 10.1109/ISITIA.2017.8124103.

[29] X. Zhang, J. Zhang, L. Li, Y. Zhang and G. Yang, "Monitoring citrus soil moisture and nutrients using an IoT based system", Sensors (Switzerland), vol. 17, no. 3, pp. 1–10, 2017, issn: 14248220. doi: 10.3390/s17030447.

[30] F. Karim, F. Karim and A. Frihida, “Monitoring system using web of things in precision agriculture”, Procedia Computer Science, vol. 110, pp. 402–409, 2017, issn: 18770509. doi: 10.1016/j.procs.2017.06.083. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2017.06.083.

[31] G. Nisha and J. Megala, “Wireless sensor Network based automated irrigation and crop field monitoring system”, 6th International Conference on Advanced Computing, ICoAC 2014, pp. 189–194, 2015. doi: 10.1109/ICoAC.2014.7229707.

[32] J. John, V. S. Palaparthy, S. Sarik, M. S. Baghini and G. S. Kasbekar, “Design and implementation of a soil moisture wireless sensor network”, 2015 21st National Conference on Communications, NCC 2015, 2015. doi: 10.1109/NCC.2015.7084901.

[33] Santoshkumar and R. Y. Udaykumar, “Development of WSN system for precision agriculture”, ICIIECS 2015 - 2015 IEEE International Conference on Innovations in Information, Embedded and Communication Systems, pp. 0–4, 2015. doi: 10.1109/ICIIECS.2015.7192904.

[34] M. Mafuta, M. Zennaro, A. Bagula and G. Ault, “Successful deployment of a Wireless Sensor Network for precision agriculture in Malawi”, Networked Embedded Systems for Every Application (NESEA), 2012 IEEE 3rd International Conference on, pp. 1–7, 2012.

[35] A. Gloria, C. Dionisio, G. Simoes, P. Sebastiao and N. Souto, “WSN Application for Sustainable Water Management in Irrigation Systems”, pp. 833–836, 2019. doi: 10.1109/wf-iot.2019.8767278.

[38] N. G. P. M. Nico, “Development of Low-cost LoRaWAN Gateway for Private Deployments”, no. November, p. 58, 2017.

[39] P. Divya Vani and K. Raghavendra Rao, “Measurement and monitoring of soil moisture using Cloud IoT and android system”, Indian Journal of Science and Technology, vol. 9, no. 31, 2016, issn: 09745645. doi: 10.17485/ijst/2016/v9i31/95340.

[40] M. Zhang, M. Li, W. Wang, C. Liu and H. Gao, “Temporal and spatial variability of soil moisture based on WSN”, Mathematical and Computer Modelling, vol. 58, no. 3-4, pp. 826–833, 2013, issn: 08957177. doi: 10.1016/j.mcm.2012.12.019. [Online]. Available: http://dx.doi.org/10.1016/j.mcm.2012.12.019.

[43] Radi, Murtiningrum, Ngadisih, F. S. Muzdrikah, M. S. Nuha and F. A. Rizqi, “Calibration of Capacitive Soil Moisture Sensor (SKU:SEN0193)”, Proceedings - 2018 4th International Conference on Science and Technology, ICST 2018, vol. 1, pp. 1–6, 2018. doi: 10.1109/ICSTC.2018.8528624.