Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Development of a Wireless Sensor Network for agricultural monitoring for internet of things(IoT)

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Gopal Lal Rajora

ADVISOR: José Polo Cantero

DATE: January, xxth 2020

Title: Development of a Wireless Sensor Network for agricultural monitoring for internet of things (IoT)

Author: Gopal Lal Rajora

Advisor: José Polo Cantero

Date: January xx, 2020

Abstract

Monitoring of the agricultural environment has become an important area of control and protection which provides real-time system and control communication with the physical world. This thesis focuses on Development of a wireless Sensor Network for agricultural monitoring for Internet of things (IoT) to monitor environmental condition.

Among the various technologies for Agriculture monitoring, Wireless Sensor Networks (WSNs) are perceived as an amazing one to gather and process information in the agricultural area with low-cost and low-energy consumption. WSN is capable of providing processed field data in real time from sensors which are physically distributed in the field. Agriculture and farming are one of the industries which have a late occupied their regards for WSNs, looking for this financially acute innovation to improve its production and upgrade agribusiness yield standard. Wireless Sensor Networks (WSNs) have pulled in a lot consideration in recent years.

The proposed system uses WSN sensors to capture and track information pertaining to crop growth condition outside and inside greenhouses. 6LowPAN network protocol is used for low power consumption and for transmitting and receiving of data packets.

This thesis introduces the agricultural monitoring system's hardware design, system architecture, and software process control. Agriculture monitoring system set-up is based on Contiki OS while device testing is carried out using real-time farm information and historical data.

I gratefully acknowledge to my Professor, Family, friends and UPC.

Table of Contents

INTRODUCTION **1**
CHAPTER 1. Literature Review **3**
1.1. Internet of Things (IoT) 3
1.1.1. IoT architecture 4
1.1.2. IoT Protocols 5
1.1.3. IoT Network Protocols: 6
1.1.4. IoT Data Protocols 11
1.2 Wireless Sensor Network 14
1.2.1 IPv6 15
1.2.2 6LoWPAN 16
Chapter 2. System Component **18**
2.1. Software Part 18
2.1.1. Open Source Operating System (ContikiOS) 18
2.1.2. InfluxDB 19
2.1.4. Ubidots 19
2.1.5. Raspbian 20
2.2. Hardware Component 20
2.2.1. Zolertia Re-Mote 20
2.2.2. Temperature and Humidity Sensor (DHT22): 21
2.2.3. Raspberry Pi 22
Chapter 3. DESIGN AND IMPLEMENTATION **24**
3.1. Instant Contiki installation 24
3.2. Raspbian Installation 24
3.3 6LBR 25
3.4 Design of the Solution 27
3.4.1. Functional Design 27
3.4.2 Node Design and Programme architecture 29
3.4.3 Database 31
3.4.4 Front End for IoT 32
CHAPTER 4. EVALUATION AND RESULTS **34**
4.1. Laboratory scenario 34
4.2 Errors and solutions 36
4.3 Results 37
CHAPTER 5. CONCLUSION AND FUTURE WORK **39**
5.1. Conclusions 39

5.2 Future Work 40
Bibliography 41

INTRODUCTION

Agriculture plays a vital role in the economy of the country and generates a large-scale job to the people. However, agriculture depends heavily on weather and climate. For example, changes in temperature, humidity, soil moisture, Carbon dioxide can lead to low crop yield. To monitor crop growth and increase the yield of agricultural production, monitoring environmental parameters is critical. Sensed information is important not only for decision-making, but also for evaluating the environmental impacts of agricultural practices.

Research in the field of wireless sensor networks (WSN) environmental monitoring has taken on a new dimension in recent years. The Wireless Sensor Network (WSN) is the ideal choice for delivering effective and economically viable solutions for a wide range of applications from health monitoring, agriculture, environmental monitoring to military operations. WSN is an advanced innovation that coordinates the information on sensors, automation control, and digital network transmission, storage of information and processing of information. It is estimated that the WSN market will increase from $0.45 billion in 2012 to $2 billion in 2022.

In addition, with the advancement of Internet of Things (IoT) technology, particularly sensor technology and cloud services, agriculture will profit far more than before. Recent technological advances lead to small sensor development, data storage, and communication, providing cost, size, power, and flexibility advantages.

Because of these advantages, we are proposing a low-cost WSN device capable of tracking various environmental parameters such as temperature and air and soil humidity. We will explain the basic components of the systems in the next section. Then we present the software design in the following sections. The approach taken here is one of a wireless agriculture monitoring system for environmental sensing.

The system can support multiple nodes at a time. The findings of this study will be introducing a point-to-point network at this point with the ability to extend the number of nodes. The document's material follows the next structure:

1. The impact of technology on the Internet of Things and Wireless Sensor Networks. Telecommunications history over the past few years with the introduction of radio networking and electronic development demonstrates the protocol overview of the network with the advantages and disadvantages of each type of technology with respect to the final application. The last section of this chapter will clarify how communication protocols have been chosen in relation to their direct impact on this project, and the index order is intended to follow the OSI model.
2. Selection of devices and technology for monitoring implementation. The key aspects to consider when implementing a Wireless Sensor Network are the elements ' electrical characteristics, the basic device information, and the recommendations of the manufacturer. The IEEE and IETF

(Internet Engineering Task Force) (Beinschob & Reinke, 2015) standardize the communication between motes. Configuration of the operating system and mandatory software requirement for network development. The transceivers used in this project were previously selected for the quality of the laboratory and the disposal of the instruments based on the implementation of previous projects with these motes.

3. Design and implementation of Wireless Sensor Network follow with installation of software and configuration of the nodes.

4. Testing of on-field device and discussion over the functionality of the end. The result obtained in this thesis are from UPC greenhouse where the network is built.

The last chapter will concentrate on conclusion and future work showing the potential impact of business on the agricultural market and network reliability.

CHAPTER 1. Literature Review

Literature review is important to have a thorough knowledge of one's intended area of research and to learn more about topics. Essential to conduct a literature review is to define the area of research, update state-of - the-art and learn the area that needs more investigation or contribution. The aim of this literature review is to learn more about the study area of the subject we will be studying and to learn from previous work done by other researchers in the field.

The available methods (available solutions in market) of testing the parameters in agriculture are manual methods. The farmers themselves test all the parameters in the system and measure the readings. Our solution focuses on designing devices and tools for controlling, viewing and alerting users using the advantages of a wireless network of sensors. This focuses on making agriculture smart using technology and IoT technologies.

1.1. Internet of Things (IoT)

The Internet of Things (IoT) is a collection of interrelated computing devices, mechanical and virtual machines, objects, animals or people with unique identifiers and the ability to transfer information over a network without needing communication between humans and computers**.** (IoT Agenda, 2020)The key goal of the Internet of Things (IoT) is to create a connected world where physical, digital and virtual converge and create smart ecosystems that provide more information for power, safety, transportation, towns, manufacturing, buildings and many other areas of our everyday lives. Potential IoT applications are common, from smart homes to smart cities, many of which require real-time and low-power communications.

McKinsey Global Institute predicted that the IoT market's financial impact on the global economy could reach up to $11.1 trillion by 2025. On the other hand, the total number of smart connected devices around the world is rapidly rising, with many organizations like IHS estimating 20.3 trillion connected "Things" going to be in use in 2017, growing to over 75 billion by the end of 2025 [ref]. As it has been shown in Figure 1.1 Based on population information and total number of connected devices, we anticipated that by the end of 2025 there would be more than 9 smart devices per person. The number of interconnected devices will pose a lot of challenges. (Reinbacher, Leon, & Wee, 2018)

Taking into account the exponential increase in the number of connected IoT devices, various international standardization bodies such as Internet Engineering Task Force (IETF), IEEE, and the 3rd Generation Partnership Project (3GPP) have developed new protocols to meet IoT specifications and emerging applications. Reliability is one of IoT communication protocol most critical criteria. In other words, data transmission through volatile and loss connections in harsh environments is inherently unreliable, leading to unnecessary retransmission, large amounts of energy consumption, and long latency in the common wired and wireless media. Therefore, reliability plays an important role not only in providing data in IoT infrastructures, but also in IoT

devices efficiency and energy consumption due to their side effects. (Monazzah, Safaei, Bafroei, & Ejlali, 2017)

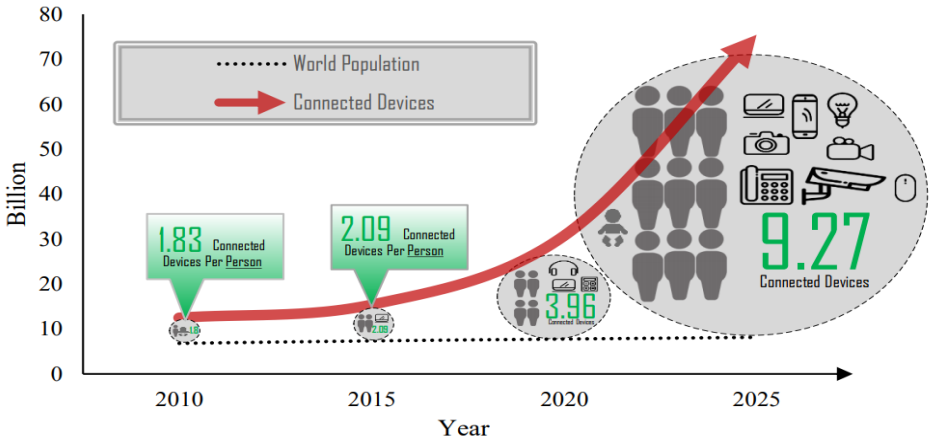


Figure 1.1 Estimated Number of Connected Devices per Person

(Monazzah, Safaei, Bafroei, & Ejlali, 2017)

1.1.1. IoT architecture

In contrast to the current internet iteration, IoT was designed to provide its benefits to very diverse objects, both in terms of functionality and technology. A unified approach to the development of IoT technologies is needed to enable such a diverse range of devices to communicate with each other.

While each IoT system is different, the basis for each architecture of Internet of Things as well as its general flow of data processing is essentially the same. First of all, it consists of the Things, which are objects connected to the Internet that can feel the world around them by means of their embedded sensors and actuators and collect information that is then passed on to the IoT gateways. (IoT architecture, 2019) The next stage is IoT data acquisition systems and gateways that capture the large mass of unprocessed data, turn it into digital streams, filter it and pre-process it so it's ready for analysis. Edge devices responsible for further processing and enhanced data analysis form the third layer. This layer is also where machine learning and visualization technologies will join. The data is then moved to data centers that can either be locally deployed or cloud-based. This is where the data for actionable insights are processed, handled and evaluated in detail. The Layers of IoT architecture shown in Figure 1.2.

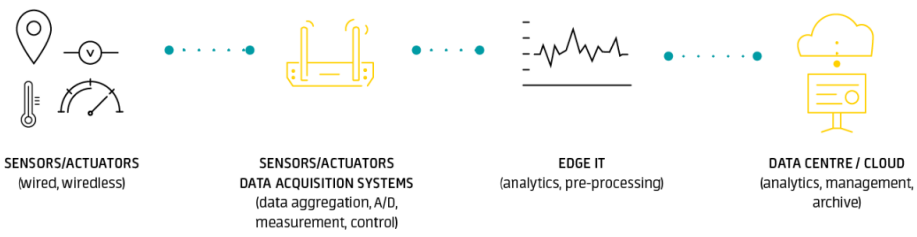


Figure 1.2 Layers of IOT architecture

(Sikder, Petracca, Aksu, Jaeger, & Uluagac, 2018)

These are the four layers of IoT architecture described in detail: (Sikder, Petracca, Aksu, Jaeger, & Uluagac, 2018)

A. **Sensing layers:** The sensing layer's main purpose is to detect any abnormalities in the peripheral devices and to obtain data from real world. This layer consists of several sensors. One of the key advantages of IoT systems is the use of multiple sensors for applications. Typically sensors are incorporated via sensor hubs in IoT devices. For multiple sensors that collect and forward sensor data to a device's processing unit, a sensor hub is a common connection point. For data flow between sensors and applications, a sensor hub uses multiple transport mechanisms (Inter-IntegratedCircuit (I2C) or Serial Peripheral Interface (SPI). Such transport mechanisms rely on IoT devices and establish a communication channel for collecting sensor data between the sensors and the applications.
B. **Network Layer:** The network layer serves as a conduit for transferring data to other connected devices, collected in the sensing layer. The network layer is implemented in IoT devices using different communication technologies (e.g., Wi-Fi, Blue-tooth, Zigbee, Z-Wave, LoRa, cellular network, etc.) to allow data flows within the same network between other devices (Sikder, Petracca, Aksu, Jaeger, & Uluagac, 2018).
C. **Data Processing Layer:** The data processing layer consists of the main data processing unit of IoT devices. The layer of data processing takes data obtained in the layer of sensing and analyzes the data to make the result-based decisions. In some IoT devices (e.g. smartwatch, smart home hub, etc.), the data processing layer also saves the previous analysis result to enhance user experience. This layer may share data processing results through the network layer with other connected devices.
D. **Application Layer:** To achieve diverse implementations of IoT systems, the application layer incorporates and displays the outputs of the data processing layer. The framework layer is a user-centered layer that executes different user tasks. There are numerous IoT applications, including smart transportation, smarthome, personal care, health care, and so on.

1.1.2. IoT Protocols

As the Internet of Things grows rapidly, many heterogeneous smart devices connect to the Internet. IoT communication protocols are communication methods that protect data exchanged between connected devices and ensure optimum protection.

Table 1 Protocol in different layers

Application layer	HTTP, CoAP, EBHTTP, LTP, SNMP, Ipfix, DNS, NTP, SSH, DLMS, COSEM, DNP, MODBUS
Network/Communication layer	IPv6/IPv4, RPL, TCP/UDP, uIP, SLIP, 6Lowpan
PHY/MAC layer	IEEE 802.11 Series, 802.15 Series, 802.3, 802.16, WirelessHART, Z-WAVE, UWB,IrDA, PLC LonWorks, KNX

IoT protocols and standards can be broadly classified into two separate categories.

1.1.3. IoT Network Protocols:

To link devices over the network, IoT network protocols are used. These are usually used over the Internet as a set of communication protocols. End-to-end data communication within the network framework is allowed using IoT network protocols. IEEE 802.15.4, low-power WiFi, 6LoWPAN, RFID, NFC, Sigfox, LoraWAN and other proprietary protocols for wireless networks are the leading networking technologies used in the IoT world. 6LoWPAN protocol will be explained in the section 1.2.2. The following are the common protocols for the IoT network:

1.1.3.1 ZigBee:

ZigBee is an IoT protocol that allows the communication of smart objects. For home automation, it is widely used. ZigBee is used with devices that allow low-rate data transfer across short distances, more common for industrial settings.

Zigbee communication is specifically designed for wireless personal area networks (WAPNs) control and sensor networks in compliance with IEEE 802.15.4 standard and is the product of the Zigbee alliance. These Zigbee's WPANs operate at frequencies of 868 MHz, 902-928MHz and 2.4 GHz.

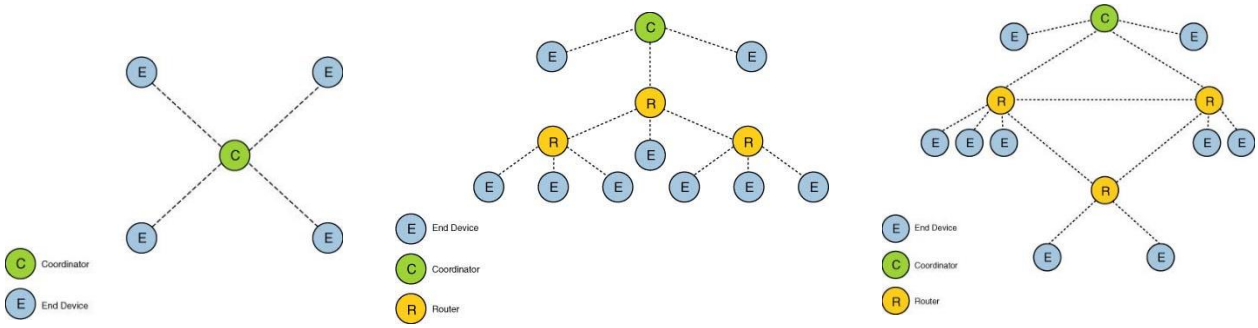
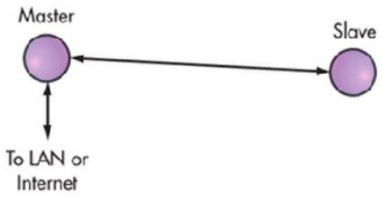


Figure 1.3 (a) Star Topology Figure 1.3 (b) Tree Topology Figure 1.3 (c) Mash Topology **(Hao & Foster, 2008)**

In a Zigbee network there are three types of devices: FFD (Fully Functional Device), RFD (Reduced Functional Device), and one coordinator for Zigbee. Therefore, an FFD node will serve as a router. It supports a wide range of network topologies including star, Mesh, or Tree shown in figure 1.3. The scheme of routing depends on topology. Certain features of Zigbee include route identification and maintenance, network connection / leaving support for nodes, short 16-bit addresses, and multihop routing.

1.1.3.2. Bluetooth Low Energy(BLE):

The Bluetooth Special Interest Group has created Bluetooth Low Energy also known as “Bluetooth Smart." It is a short-range communication protocol with PHY and MAC layer. The stack of the BLE protocol is similar to the stack used in traditional Bluetooth systems. It has two parts: host and controller. The controller deploys the physical and communication layer. The controller is normally a radio based SOC (System on Chip). BLE is not compatible with classic Bluetooth.



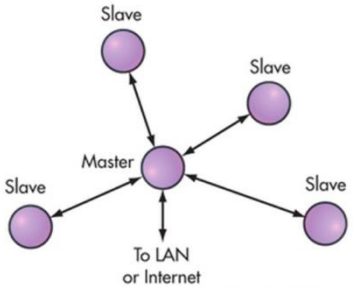


Figure 1.4 (a) Point to Point Topology. Figure 1.4(b) Star Topology

(“Topology Options | Bluetooth Technology Website.”, 2019)

The main difference is that data streaming is not supported by BLE. Instead, with a data rate of 1 Mbps, it supports fast transfer of small data packets (packet size is small).

BLE has two types of devices: master and slave. The master functions as a central device capable of connecting with specific slaves. The Bluetooth version 5 released new features such as low power consumption, mesh communications stack, and distance range for the Internet of Things. BLE's energy efficiency is 2.5 times higher than Zigbee. (kim, Evans, & Iversen, July, 2008)

Typically, the topologies shown in Figure 1.4 are used in these types of networks, where all devices have to communicate with a coordinator. Other available topologies are: peer-to-peer, broadcast and mesh. (“Topology Options | Bluetooth Technology Website.”, 2019)

The connection is on all the time in classic Bluetooth, even if there is no data transfer. It also supports 79 data channels (1 MHz channel bandwidth) and a data rate of 1 million symbols /s, while BLE supports 40 channels with 2 MHz channel bandwidth and 1 million symbols/s data rate. BLE embraces low-duty cycle requirements because its packet size is small and it requires as little as 80s to send the smallest packet.

1.1.3.3. Wi-Fi

Wi-fi is a family of wireless networking systems, based on the popular IEEE 802.11 specification, widely used for computer networking and internet access in the local area. The term Wi-Fi is commonly synonymous with wireless access, despite being a particular trademark held by the Wi-Fi Alliance, a group dedicated

to certifying that Wi-Fi devices follow the 802.11 wireless standards set by the IEEE.

Such specifications, with names such as 802.11b and 802.11ac, form a specification family that started in the 1990s and is still growing today. The technological evolution is shown in figure 1.6.

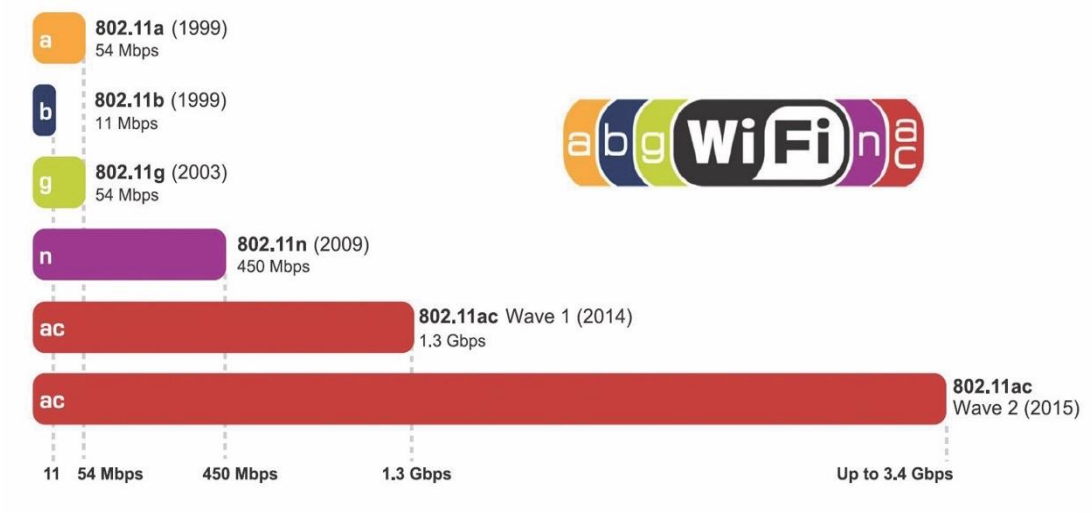


Figure 1.5 802.11 standard amendments **("The evolution of WIFI", 2017)**

The 802.11 specifications codify changes that increase wireless performance and Coverage as they become usable, as well as the use of new frequencies. They also address new technologies that reduce the consumption of energy.

Different IEEE 802.11 protocol specifications describe the different versions of Wi-Fi, with the various radio systems that define frequency bands and the maximum ranges and speeds that can be reached. The 2.4 gigahertz (120 mm) UHF and 5 gigahertz (60 mm) SHF ISM radio bands are most frequently used by Wi-Fi; these bands are split into several channels. Channels can be shared between networks, but at any point in time only one transmitter can transmit locally on a channel ("The evolution of WIFI", 2017)

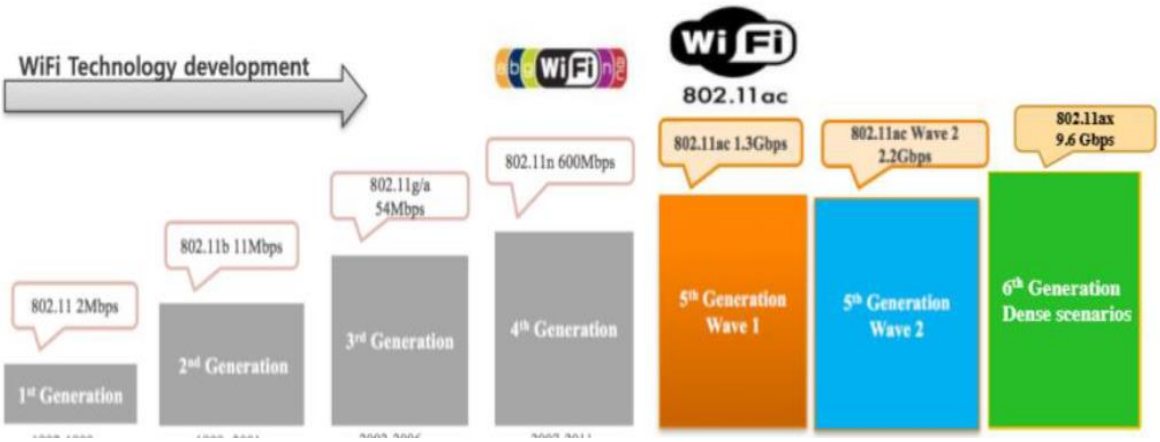


Figure 1.6 Wi-Fi technological evolution **(Templeton, Carlson, Leon-Garcia, & Widjaja, 2018)**

1.1.3.4. Thread

Thread is an IPv6-based networking protocol designed in an IEEE 802.15.4-2006 wireless mesh network, commonly known as a Wireless Personal Area Network (WPAN), for low-power Internet of Things devices. It is independent of other 802.15.4 mesh networking protocols, such a ZigBee, Z-Wave, and Bluetooth LE.

The main features of Thread include:

- Simplicity — Easy Setup, Start-up, and Operation
- Safety — All devices in a Thread Network are authenticated and all messages are encrypted
- Reliability — Self-healing mesh networking, without a single point of failure and spread-spectrum techniques to provide immunity to interference
- Efficiency — Low-power Thread devices can sleep and run on battery power for years
- Scalability — Thread networks can cover hundreds of devices

Standards employed in this protocol are observed in the Figure 1.7.

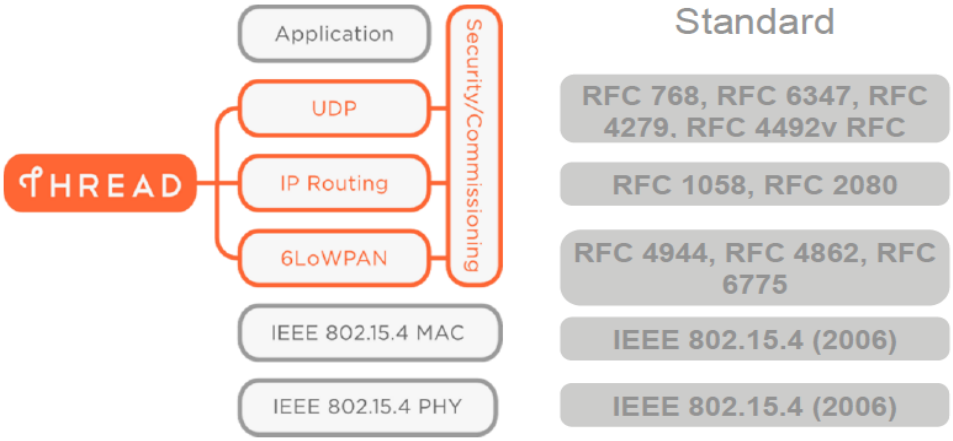


Figure 1.7 Stack Module of Thread **(Internet of Things in 5 Days, 2015)**

Thread uses 6LoWPAN, which, like Zigbee and other devices, also uses the IEEE 802.15.4 wireless protocol for mesh communication. Thread however, with cloud access and AES encryption, is IP-addressable. Nest has also released a BSD licensed open-source Thread implementation (called "OpenThread") (Home, 2020)

1.1.3.5. Cellular Networks (2G, 3G & 4G)

This network protocol has become the most popular paid communications service for long distances. Cellular networks are high-speed, high-capacity voice and data communication networks with enhanced multimedia and seamless roaming capabilities for supporting cellular devices. Such networks are used for

more than just entertainment and phone calls, with the increasing popularity of cellular devices.

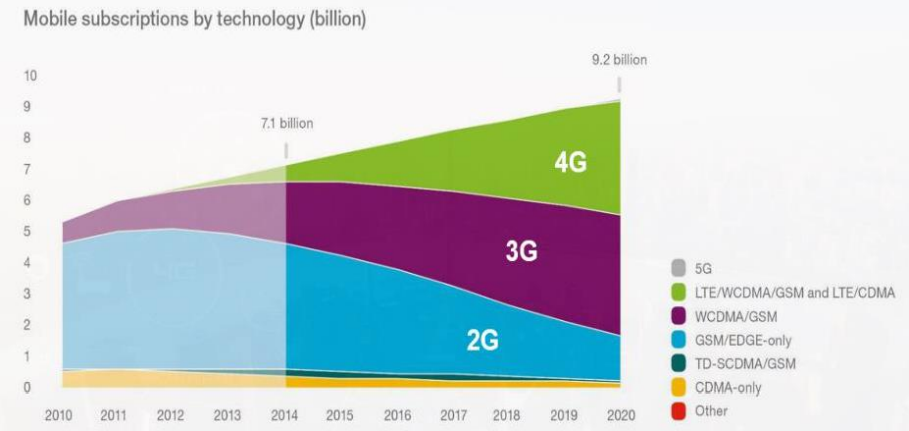


Figure 1.8 Number of subscribers **(Matilla, 2018)**

Due to streaming of video and data sharing between mobile devices, the number of subscriptions and data rate consumption (Figure 1.8) increased. Every technology evolution deals with the data rate for coverage, the cellular area is reduced, allowing more people to use the Internet Service Provider (ISP) bandwidth effectively.

1.1.3.6. LoRaWAN:

The Long Range (LoRa) wide area network is the stack protocol for the LoRa modulation defined by the LoRa alliance. It is a Low Power, Wide Area(LPWA) networking protocol designed to wirelessly interface battery worked 'things' to the internet in local, national or worldwide systems, and targets key Internet of Things (IoT) necessities, for example, bi-directional communication, end to end security, portability and localization services. It defines the communication protocol and system architecture for the network. And also responsible for managing the communication frequencies, data rate, and power for all devices. Network structure for LoRaWAN we can see in figure 1.9.

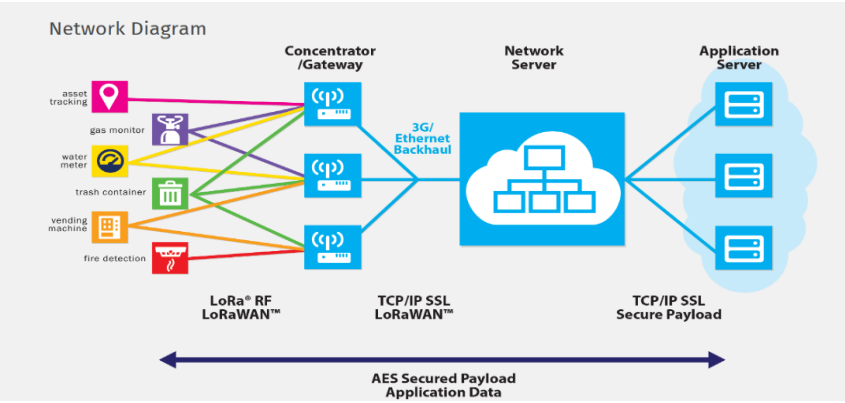


Figure 1.9 Network Diagram for LoRaWAN

(Paredes-Parra, Garcia-Sanchez, Mateo-Aroca, & Molina-Garcia, 2019)

LoRaWAN network architecture is deployed in a star-of-stars topology for managing communication between LPWAN gateways and end-node devices as a routing protocol, maintained by the LoRa Alliance.
LoRaWAN has three different classes of end-point devices to address the different needs reflected in the wide range of applications:

- Class A - Lowest power, bi-directional end-devices
- Class B - Bi-directional end-devices with deterministic downlink latency
- Class C - Lowest latency, bi-directional end-devices

Communication is distributed across a variety of frequency channels and data rates between end devices and gatewaysó LoRaWAN data rates vary from 0.3 kbps to 50 kbps.Choosing the data rate is often a trade-off between the period of the message and the range of communication.

1.1.3.7. Narrowband IoT(NB-IoT):

Narrowband Internet of Things (NB-IoT) is a standard radio technology Low Power Wide Area Network (LPWAN) developed by 3GPP to support a wide range of cellular devices and services.The specification was frozen in 3GPP Release 13 (LTE Advanced Pro) and NB-IoT is, among others, also referred to as LTE Cat-NB. Other terms such as LTE Cat-NB1 and Cat N1 apply to the NB-IoT specification released in 2016. (Internet of Thing, 2020)

NB-IoT focuses on indoor coverage, low cost, long battery life, and high density of link. NB-IoT uses an LTE standard version, which restricts the spectrum to a single 200kHz narrow band. This uses downlink communication with OFDM modulation and uplink communication with SC-FDMA.

NB-IoT: less powerfull than LTE-M: Of the two primary cellular LPWAN standards now available in networks (namely NB-IoT and eMTC / LTE-M), NB-IoT is the least ' strong ' in terms of speed, data transfer capabilities, support for cases of genuine mobile use and more; in other ways, it is stronger.

Latency of NB-IoT: NB-IoT is less ideal for situations where network latency needs to be very low when compared to LTE-M. Usually, latency is equivalent to or less than 10 seconds (about 1.6 to 10 seconds).

NB-IoT: focus on energy efficiency: NB-IoT theoretically offers up to or even little over 10 years of battery life. It is better than LTE-M in this regard.

1.1.4. IoT Data Protocols

To link low-power IoT devices, IoT data protocols are used. Such protocols provide point-to-point communication with the user-side hardware without any connection to the Internet. IoT data protocol communication is via a wired or cellular network. Some of the protocols for IoT data are:

1.1.4.1. Message Queue Telemetry Transport (MQTT)

MQTT gathers data from various electronic devices and supports remote device control as one of the most popular protocols for IoT applications. It is a subscribe / publish protocol running over Transmission Control Protocol (TCP), meaning it supports event-driven exchange of messages through wireless networks. It is designed for remote location connections where a "small code footprint" is required or where the bandwidth of the network is limited. The high level architecture of MQTT is shown in Fig. 1.10.

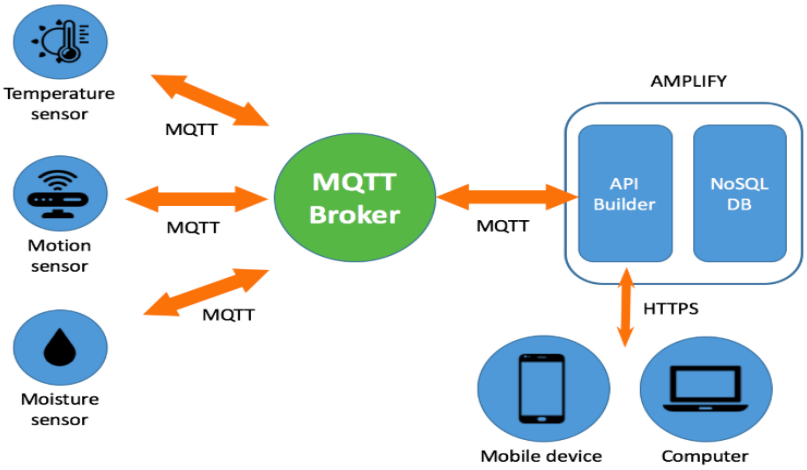


Figure 1.10 MQTT architectures **(Brenman, 2018)**

Different customers can subscribe to a central broker to accumulate topics generated by the device in MQTT using a model for exchanging messages. This message exchange model is resource-sensitive and dose not follow a particular structue of data. When a message is posted by the device, a client receives a message from the broker it subscribed to about a topic. The broker's aim is to ensure transmission of messages by simplifying management and enabling networked IoT devices. It is well suited for machine-to-machine communication. It also provides security even if the connection breaks down with untrustworthy connections for the transmitted messages.

1.1.4.2. Constrained Application Protocol (CoAP)

CoAP is a protocol for Internet utility for gadgets that are constrained. The client send a request to the server using this protocol and the server can send the res ponse back to the client in HTTP. This uses UDP (User Datagram Protocol) for lightweight implementation and reduces space usage the protocol uses the EXL (Efficient XML Interchanges) binary data format. CoAP is best suited for low-energy consumption sensors to utilize RESTful services within their power limitations. The architecture of CoAP is shown in Figure 1.11.

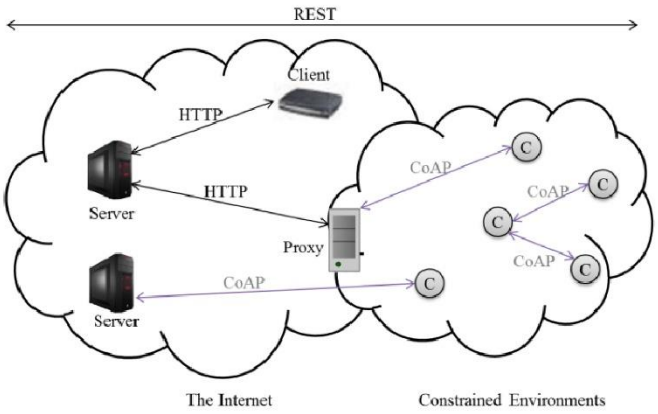


Figure 1.11 Architecture of CoAP **(What is CoAP protocol IoT | CoAP Architecture, message format, n.d.)**

CoAP consists of two layers called Messaging and Application/ Response. The messaging layer is responsible for reliability and accuracy of any message, while request / response layer function is networking and contact handling. CoAP also promotes multi-cast messaging and supports asynchronous message exchange.

The CoAP protocol is commonly used in automation, mobile and microcontroller applications. The protocol sends a request to the endpoints of the application, such as home appliances, and returns the response of the application services and resources.

1.1.4.3. Advanced Message Queuing Protocol (AMQP)

AMQP is a messaging-oriented middleware system software layer protocol that offers routing and queuing. It is used for point-to-point reliable connection and supports seamless and secure data exchange between the connected devices and the cloud. AMQP consists of three different Exchange, Message Queue, and Binding elements. These three elements ensure that messages are shared and stored securely and effectively. It also helps to build the connection between one message and the other. The Basic architectue of AMQP is show in the figure 1.12.

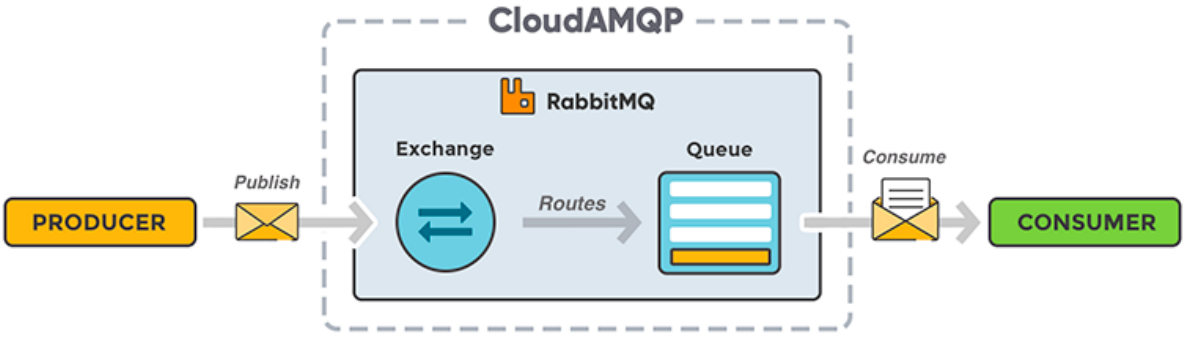


Figure 1.12 Architecture of AMQP **(RabbitMQ for beginners, 2015)**

The AMQP protocol is used mainly in the banking sector. The protocol records the message whenever a message is sent by a server until each message is transmitted without fail to the intended users / destinations.

1.2 Wireless Sensor Network

A WSN is a network of small and inexpensive sensor nodes that monitor and communicate with each other by means of radio signals. WSN's scope is to coordinate data collection that is transmitted to a central location. Sensor nodes are basically small devices with extremely basic processing power, limited storage / memory capacity and low energy consumption.

Though rapid interest and research in WSN fields have taken place only recently but, use of sensors for specialized services is not new. Sound Surveillance System (SOSUS), which used acoustic sensors, was used to track silent Soviet submarines during the Cold War.

Wireless Sensor Networks (WSN) emerged from advancements in the areas of micro-electromechanical system (MEMS) technology, wireless communication, and digital electronics. WSNs devices are small in size, low cost, and require low power to work. Below is shown the basic structure of the WSN sensor nodes (Figure 1.13).

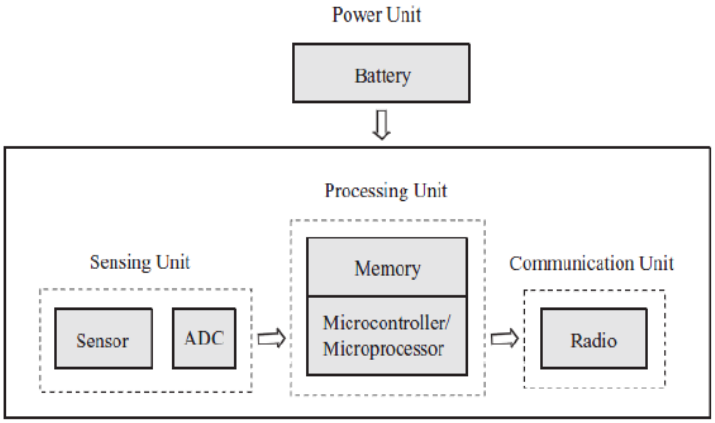


Figure 1.13 WSN Sensor Nodes General Structure

(Joshi, Kanade, & Joshi, 2017)

A sensor node consists of four main components. The sections are: a sensing unit, a processing unit, a power unit and a transmission unit. Depending on the type of application, additional parts such as a location finding method, mobilization and power generator may be available to a sensor node. Generally, the sensing unit takes the burden of sensing and sensor data collection and then transfers the data to the processing unit. The processing unit collects and processes the sensed data in accordance with a specified procedure or system. The sensor is not connected to a network by a transmission mechanism. The power unit provides the power needed to run a node of the sensor.

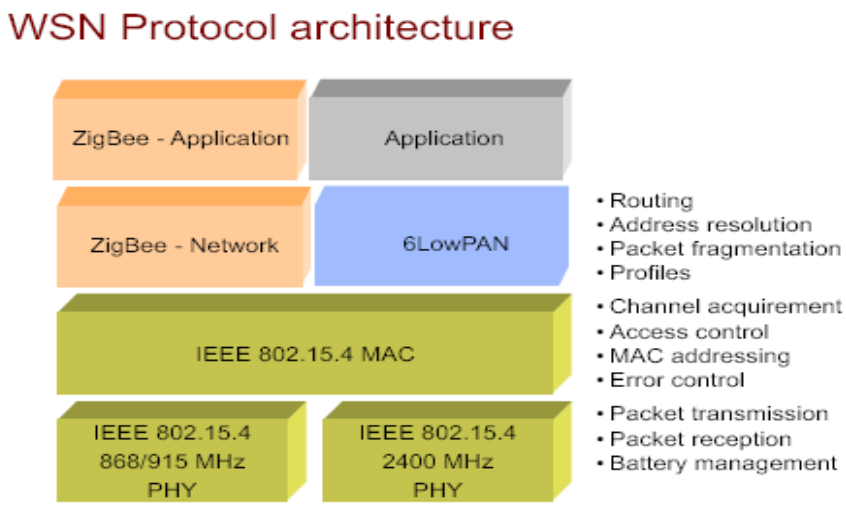


Figure 1.14 Wireless sensor network protocol

(Templeton, Carlson, Leon-Garcia, & Widjaja, 2018)

So, in the IoT section we discussed about different type of protocols. In the following section we are going to discuss about 6LoPAN and IPv6 in depth, which I used in my project.

1.2.1 IPv6

IPv6 is the next generation Internet Protocol (IP) standard intended to replace IPv4, which is still used by many Internet services today. To communicate with other users, each computer, mobile phone, and any other device connected to the Internet requires a numerical IP address. The original system of IP addresses, called IPv4, runs out of addresses, thus the need for a new protocol, IPv6. This is because IPv6 addresses are 128-bit based, whereas IPv4 addresses are 32-bit based.

An IPv6 address is 128 bits long in precise terms and is divided in eight classes, 16 bits each. Each group is represented as four hexadecimal digits and colon separates the groups.

An example of a full IPv6 address could be:

FE80:CD00:0000:0CDE:1257:0000:211E:729C

Other important technological changes in IPv6, apart from the pool of possible addresses, would boost the entire functionality of the IP protocol:

- No more NAT (Network Address Translation)
- Auto-configuration of addresses.
- Unique private addresses, so no private address collisions.
- Better multicast routing.
- Simpler header format.
- Simplified, more efficient routing.

- True quality of service (QoS), also called "flow labelling".
- Built-in authentication and privacy support.
- Flexible options and extensions.
- Easier administration (say good-bye to DHCP).

IPv6 is expected to be the next step in the Internet protocol and is being implemented worldwide at a rapid pace, hitting approximately 30 percent as of December 2019 as shown in (Figure 1.15).

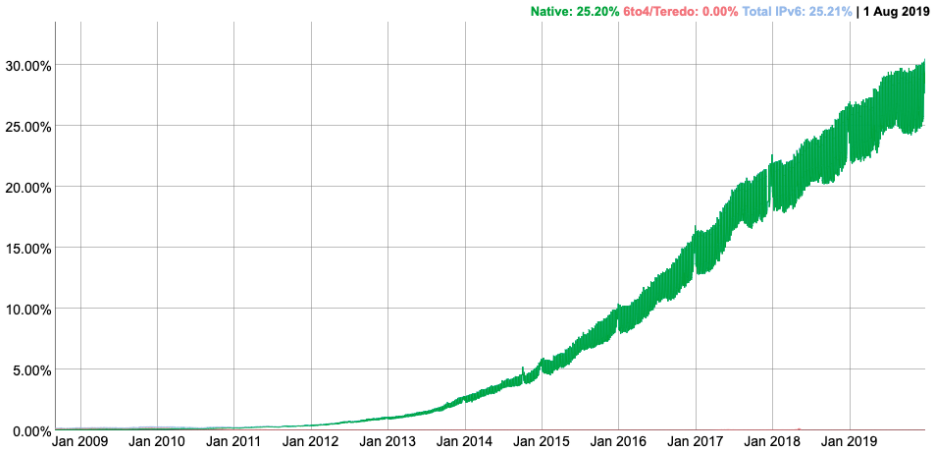


Figure 1.15 Adoption of IPv6 Worldwide **(GoogleIPv6, 2020)**.

1.2.2 6LoWPAN

6LoWPAN is a simple low-cost communication network that enables wireless connectivity in applications with limited power and relaxed throuhput requirements as it provides Ipv6 overIEEE 802.15.4 networking.

It consists of devices compliant with the standard IEEE 802.15.4, characterized by short range, low bit rate, low power, low memory use and low cost.

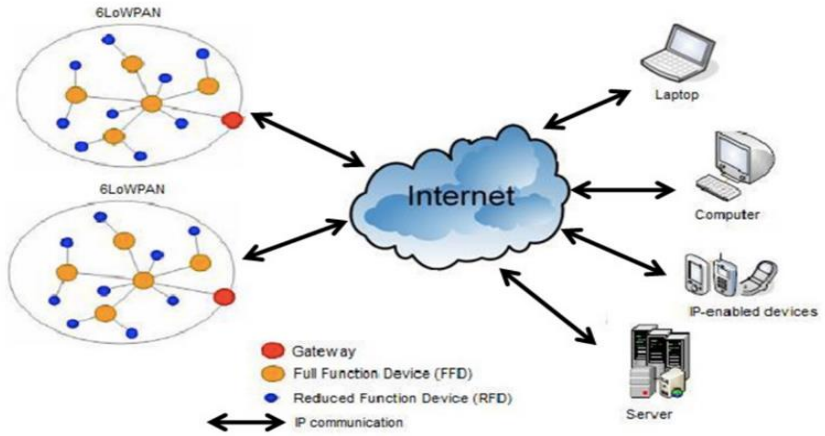


Figure 1.16 6LoWPAN architecture.

(Shelby & Bormann, 2011)

The concept was created because engineers felt they were left out of the Internet of Things as the smallest devices. 6LoWPAN can communicate on an IP network connection with 802.15.4 devices as well as other device types such as WiFi. The two can be linked by a bridge system that functions as 6LBR(6LoWPAN border router). its architecture is shown in Figure 1.16.

When a lower processing capability sensor node in a 6LoWPAN or so-called reduced function device (RFD) wants to send its data packet outside the 6LoWPAN to an IP-enabled device, it sends the packet to the higher processing capability sensor node or so-called full function system (FFD) in the same PAN first. The FFDs that respond in 6LoWPAN as a router will forward the data packet hop by hop to the 6LoWPAN gateway. The 6LoWPAN gateway that connect to the 6LoWPAN with the IPv6 domain will then forward the packet to the destination IP-enabled device by using the IP address.

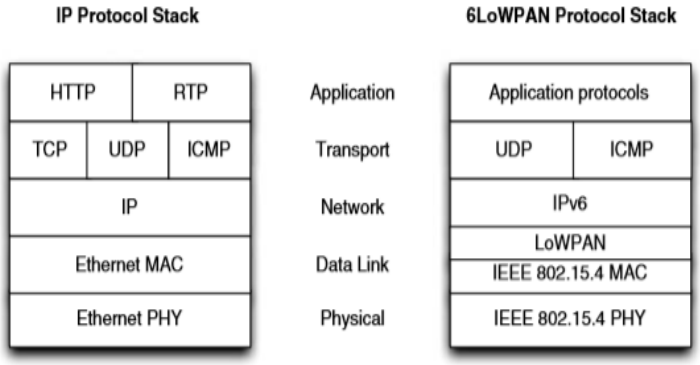


Figure 1.17 Difference between conventional IP and 6LoWPAN protocol stack.

The physical and link layer operates with the standard IEEE802.15.4, which is also known as optimizing IPv6 over IEEE802.15.4. The Figure 1.17 shows the OSI model used in traditional IP protocol stack and the 6LoWPAN.

Chapter 2. System Component

This chapter is divided in the two part: Software part and Hardware Components.

2.1. Software Part

In this section will be introduced and discussed within depth the software and Application used in the project.

2.1.1. Open Source Operating System (ContikiOS)

To facilitate the development and portability of IoT applications, operating systems are useful in low-end devices. Many operating systems are connected to a provider of hardware and target only one architecture of hardware. Contiki is an excellent, but not the only, free and open source operating system. Next IoT's most commonly used options are listed as OS.

- TinyOS
- RIOT
- Contiki

Contiki is an open source operating system for the Internet of Things, it connects tiny low-cost, low-power microcontrollers to the Internet.

Contiki offers efficient low-consumption Internet communication, supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6lowpan, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated. It also uses Rime stack. For sensor networks, it is a lightweight communication stack and has thinner layers than traditional stacks. The layers are simple (only a few bytes) and they have tiny headers. Rime also supports the reuse of code and this protocol's main purpose is to simplify the implementation of sensor networks. (Internet of Things in 5 Days, 2015).

Contiki is a versatile wireless network building toolbox. It was developed by Adam Dunkels et al. at the Swedish Institute of Computer Sciences. Contiki is a highly portable OS and has already been transported to a number of platforms operating on different processor types.

Development is easy and fast with Contiki: Contiki applications are written in standard C, Contiki networks can be emulated with the Cooja simulator before they are burned into hardware, and Instant Contiki offers a complete development environment in one download.

Instant Contiki

Instant Contiki is an entire Contiki development environment in a single download. It is an-Ubuntu Linux virtual machine that runs in VMWare player and

has Contiki and all the development tools, compilers, and simulators used in Contiki development installed.

The latest Instant Contiki release is 3.0, following the Contiki 3.0 source code release. An Instant Contiki 3.0 was used to making of this project. Due to its ease of installation and convenience of availability, distribution was used. The process of installation will be explained and further discussed in section [3.1].

2.1.2. InfluxDB

InfluxDB is an open-source time series database developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. InfluxDB is user-friendly, scalable and highly available. A time series database is used over a period of time to store log, sensor and other data. When the Internet of Things (IoT) arrives, activities need to be tracked through multiple applications.

InfluxDB organizes data on disk into immutable runs of values for a single column of a series. A delete operation needs to undo a lot of that work for a subset of points. Deleting a row from the table in InfluxDB produces a tombstone. A tombstone contains the deleted range's series key and min and max time.

2.1.3. Grafana

We live in a world of big data where large amounts of data are produced even by small-scale IT environments. Once an organization has found out how to tap the data generating data from the different data sources and the method of collecting, processing and storing it, the next step is analysis.

Grafana is an open source visualization tool that can be used in addition to a variety of data stores, but is most widely used in combination with Graphite, InfluxDB, Elasticsearch and Logz.io. It is most commonly used for visualizing time series data for infrastructure and application analytics but many use it in other domains including industrial sensors, home automation, weather, and process control. Once an organization has found out how to tap the data generating data from the different data sources and the method of collecting, processing and storing it, the next step is analysis.

2.1.4. Ubidots

Ubidots is an Internet of Things (IoT) application builder with data analytics and visualization. It turn sensor data into knowledge that matters for business decisions, machine-to-machine interactions, educational research, and improve global resource economization. Ubidots are an easy and affordable way to integrate the IoT's power into your business or research.

Ubidots technology and engineering stack have been built to provide our users with a safe, white-glove experience. Device-friendly APIs (accessed via HTTP /

MQTT / TCP / UDP protocols) provide an easy and secure connection to send and retrieve data in real-time from and to our cloud service. Ubidots application supporting platform supports interactive, real-time data visualization (widgets) and an IoT App Builder that allows developers to broaden the platform with their own HTML / JS code when necessary for private customization. Ubidots are available for businesses and researchers to easily link computers, manage data, and save an area.

2.1.5. Raspbian

Raspbian is a free Debian-based operating system designed for Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. Raspbian provides more than a pure OS, however: it comes with more than 35,000 packages, pre-compiled applications packaged in a nice format for easy installation on your Raspberry Pi.

In June 2012, the initial construction of over 35,000 Raspbian packages, designed for the Raspberry Pi's best performance, was completed. Nevertheless, Raspbian is still under active development with a focus on improving as many Debian packages as possible's reliability and performance.

Raspbian Buster with desktop for this project, is the OS that will run on Raspberry Pi. A well thought-out and comprehensive guide will be given in the next chapter on how to properly install Raspbian and how to set up a proper working environment.

2.2. Hardware Component

Within this section will be introduced and discussed within depth the physical devices used in the project. To fully understand its limitations and implications when implementing the network design in a scenario of real-world testing.

2.2.1. Zolertia Re-Mote

The Remote is a low-power wireless module designed to assist WSN developers in testing and deploying their own applications and prototypes with the best interaction between development time and flexibility in hardware.
Zolertia Re-mote is shown in figure 2.1.

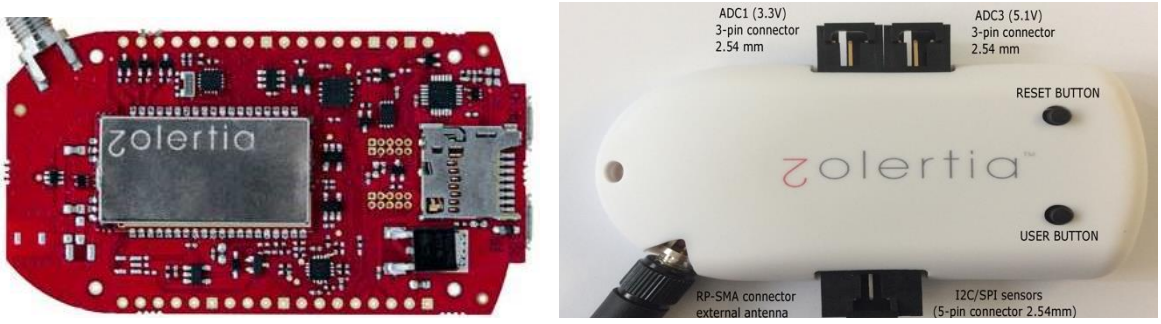


Figure 2.1 Zolertia Re-Mote and some of its features

The platform is based in the Texas Instruments CC2538 ARM Cortex-M3 system on chip (SoC), with an on-board 2.4 GHz IEEE 802.15.4 RF interface, running at up to 32 MHz with 512 KB of programmable flash and 32 KB of RAM, bundled with a Texas Instruments CC1200 868/915 MHz RF transceiver to allow dual band operation.

Zolertia RE-Mote Features:

- ISM 863-950-MHz ISM/SRD band IEEE 802.15.4 compliant radio.
- ARM Cortex-M3 32 MHz clock speed, 512 KB flash and 32 KB RAM (16 KB retention)
- AES-128/256, SHA2 Hardware Encryption Engine
- ECC-128/256, RSA Hardware Acceleration Engine for Secure Key Exchange
- User and reset button
- Consumption down to 150 nA using the shutdown mode.
- Programming over BSL without requiring to press any button to enter bootloader mode.
- Built-in battery charger (500 mA), facilitating Energy Harvesting and direct connection to Solar Panels and to standards LiPo batteries.
- Wide range DC Power input: 3.3-16 V.
- Small form-factor (73 x 40 mm).
- MicroSD (over SPI).
- On board RTCC (programmable real time clock calendar) and external watchdog timer (WDT).
- Programmable RF switch to connect an external antenna either to the 2.4 GHz or to the Sub 1 GHz RF interface through the RP-SMA connector.
- Supported in Open Source Operative Systems as Contiki, RIOT and OpenWSN (in progress).

2.2.2. Temperature and Humidity Sensor (DHT22):

DHT22 Sensor is chosen for temperature and humidity measurement. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). Simply connect the first pin to 3-5V power on the left, the second pin to your data input pin and the pin to the ground on the right. While using a single wire to send data, it is not compatible with Dallas One Wire. DHT22 sensor and Pins shown in figure 2.2.

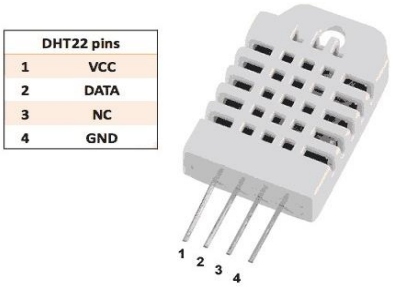


Figure 2.2 DHT22 Sensor and Pins

Technical Specification:

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 0-100% humidity readings with ±2% accuracy
- Good for -40 to 80°C temperature readings ±0.5°C accuracy
- No more than 0.5 Hz sampling rate (once every 2 seconds)
- Body size 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53") 4 pins, 0.1" spacing.

2.2.3. Raspberry Pi

Raspberry Pi (RPI) is a single board computer (SBC) built by the Raspberry Pi Foundation in the United Kingdom. The Raspberry Pi is a very inexpensive computer that runs Linux, but it also includes a series of GPIO (general purpose input / output) pins allowing you to monitor physical computing electronic components and explore the Internet of Things (IoT).

There are some commercial models of Raspberry Pi, the model used in this project is Raspberry Pi 3B (figure2.3.) installed with Raspbian

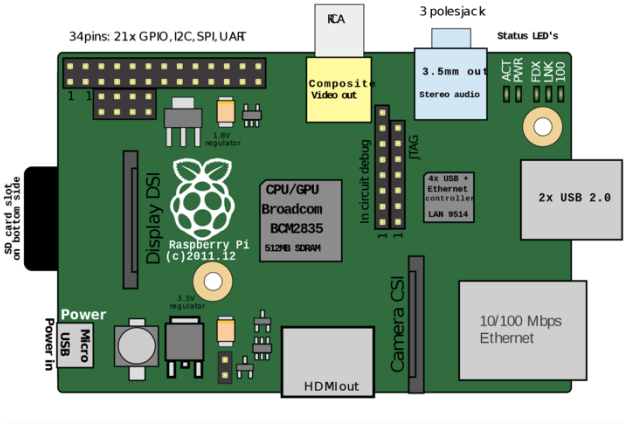


Figure 2.3 Raspberry Pi 3B and some of its features.

Raspberry Pi 3 Model B technical specifications:

The main technical specifications of the Raspberry Pi model to be used in this project are:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI

- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

In the Next chapter we will discuss about the Design and Implementation of the network.

Chapter 3. DESIGN AND IMPLEMENTATION

In this chapter we are going to explain about Design of the solution. Using the network protocol 6LoWPAN the selection of the element in chapter 2 will be used to develop the network over IPv6. The border router will be implemented with the raspberry pi 3 B, this element's main objective on the network is to establish a gateway for exchanging information from the IPv6 to IPv4.

3.1. Instant Contiki installation

Instant Contiki is the OS chosen to run on this project's Zolertia Z1 motes to get Instant Contiki, the following steps are needed.

- Download Instant Contiki from following link and unzip the file, place the unzipped directory in any Drive.
  https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/InstantContiki3.0.zip/download

- Simply open the Instant Contiki Ubuntu 12.04 32-bit.vmx file using VMWare, if asked about the source of the VM, just choose to copy it, then wait for the virtual Ubuntu Linux boot.

- Log in to Contiki Instant. The username and password are **user**. Right now, don't upgrade.

To compile an application, it is necessary to specify the target hardware platform using the TARGET = syntax platform and compile the native platform if it is not specified by default.

```
make TARGET=Zoul savetarget
```

The program can be compiled and loaded on Re-mote node:

```
make TARGET=Zoul BOARD=remote-revb codefile.upload
```

3.2. Raspbian Installation

Raspbian is the selected operating system to run the Raspberry Pi, its main function will be to enable communication between IP networks and the WSN composed of 4 Zolertia Remote motes in this case. A detailed step-by-step guide will be provided in this section on how to get a Raspbian copy running on our Raspberry Pi.

Once you start, remember that you will need an SD card or micro SD card with an SD card adapter, a spare mouse, a keyboard and an HDMI connection screen.

- Raspbian can be downloaded from the original website.
- Burn the image of Raspbian OS on the SD card using BalenaEtcher SD-burning Software.
- Connect the SD card into the Raspberry.
- Use VNC viewer or Putty to see Terminal in the Laptop Screen.

3.3. 6LBR

A border router with 6LoWPAN is a border router which connects 6LoWPAN devices directly to a local network or the Internet. It is developed by CETIC, open source solution that can play a role as a router or bridge. It separates IPv6 and 6LoWPAN networks into different subnetworks when it comes to a router, acting as a normal router. Join two networks that are distinct. The Iot network architecture of LBR is shown in the figure 3.1.

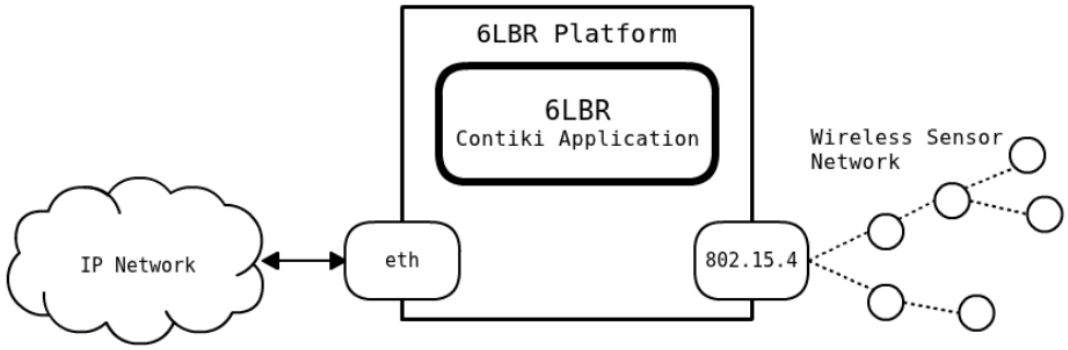


Figure 3.1 LBR in the IoT network architecture

The RPL protocol controls the 802.15.4 network, and NDP handles the Ethernet side. Logically, these two networks are isolated. For example [aaaa::] for ethernet and [bbbb::] for WPAN, every network has a different prefix. The nodes are designed to route via 6LBR and vice versa the traffic intended for WPAN.

In addition to acting as a router, this virtual router, which can be installed in different operating systems, provides many operating modes, as an example we have:

- Smart Bridge
- Transparent bridge
- Router
- NDP-Router
- 6LR
- RPL-Root
- RPL-Replay
- Full Transparent Bridge

For its ease of use and its network-level features, the router mode is configured in this design.

6 LBR Installation:

As previously explained, the CETIC-6lbr is built over Contiki. This system may be catalogued as a middleware, used to handle the IPv6 subnet's data and node configuration.

Before any changes on the system, the packages and repertoires may be updated and upgraded.

6LBR requires the installation of Contiki libncurses, and tap-bridging bridge-utils. The following command has been executed to install them.

```
apt-get install libncurses5-dev bridge-utils
```

To download the latest version of the source code you will need the following commands:

```
git clone https://github.com/cetic/6lbr
cd 6lbr
git submodule update –init --recursive
```

The following commands compile and install 6LBR from source code:

```
Make all
Make plugins
Make tools
Make install
Make plugins-install
Update-rc.d 6lbr defaults
```

When 6LBR has been installed the next stage is configuration to 6LBR. All configuration parameters are in the file /etc/6lbr/6lbr.conf. As previously stated, 6LBR will run as interface configuration in router mode with bridge mode. The setup to 6LBR is as follows:

```
MODE=ROUTER
RAW_ETH=0
BRIDGE=1
CREATE_BRIDGE=0
DEV_BRIDGE=br0
DEV_TAP=tap0
DEV_ETH=eth0
RAW_ETH_FCS=0
DEV_RADIO=/dev/ttyUSB0
BAUDRATE=115200
```

NODE_CONFIG=/etc/6lbr/node_config.conf

The Bridge interface needs to be created, too. The contents of the interface configuration file (/etc / network / interfaces) when DHCP is enabled are as follows:

```
iface eth0 inet static
address 0.0.0.0
auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_stp off
up echo 0 > /sys/devices/virtual/net/br0/bridge/multicast_snooping
post-up ip link set br0 address ip link show eth0 | grep ether | awk {print $2}
```

Whenever the configuration is changed it is necessary to restart the service

```
service 6lbr restart
```

3.4 Design of the Solution

In this section we will explain about the design of the system and working. The gateway will give an order to all sensor nodes to execute data on the network. - Sensor node publishes its data in the MQTT broker for a specific topic and the gateway subscribes to all sensor node topics to receive all data from sensor nodes. A data base server (InfluxdbDB) is running inside the Raspberry pi, which allows it to store everything the sensors can send in a non-relational database. The data stored in infulxdb is then processed to send in real time to the Grafana and cloud and visualize as a graph.

3.4.1. Functional Design

In almost every area the Internet of Things may have applications. Such structures are responsible for gathering information from natural ecosystems to houses and factories in different environments. Embedded devices with limited CPU, memory and power capacities can be connected to an IoT network. Many manufacturers, however, are still waiting to see what to do and when to begin. This is a benefit for small businesses as they are able to move forward and create new concepts that are suited to the Internet of Things.

In our architecture, we consider a WSN network, a border router, a server, a database and finally a cloud-based system Ubidots for the presentation of information. Zolertia sensors are extending the WSN network. The remote Zolertia acquires the information about the environment and sent to the bouter using MQTT as the transport protocol.

Figure 3.2, demonstrates the various stages of the scenario suggested. Devices that build the WSN network connected to sensors, a border router, various services that allow higher programming environment storage and processing of other applications.

As stated in chapter [2.7.1], Zolertia Re-mote nodes have built up sensors that can be used to monitor physical signals from the world. But in this project we used external sensor. Temperature, Humidity and Soil have been tracked in this project.

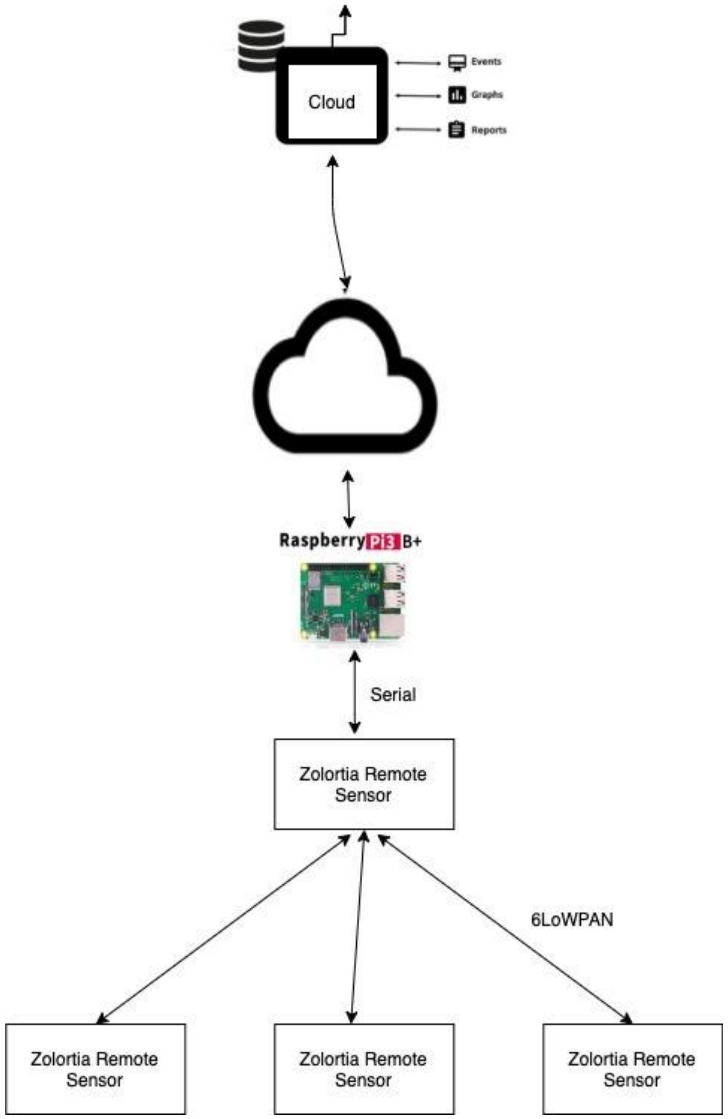


Figure 3.2 Schematic of the whole solution including devices and communications

As seen in [Figure 3.3.] the data sensed by the WSN is added to the database independently of the web server, this increases the solution's modularity and, if desired, allows multiple web servers to feed from that same database. It is also worth noting that both a zolertia Remote and a Raspberry Pi act as a single border routing device make up the Border router interface. The web server will also have a user interface that allows users to read data about relevant events collected by the WSN in real time and in history.

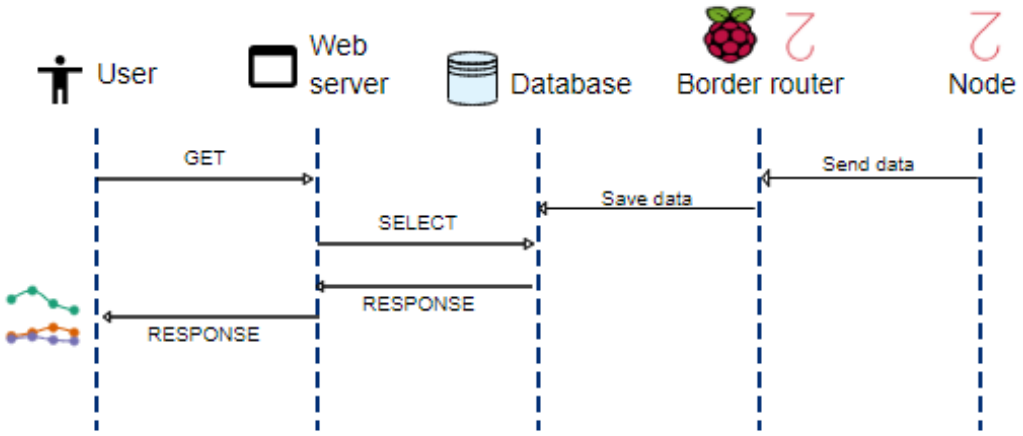


Figure 3.3 Communications flow within the system

3.4.2 Node Design and Programme architecture

Beginning from the diagram's edge and at the heart of the Design programs, there's the WSN node. The nodes run over the Contiki, explained in chapter [3.1] in more detail. Zolertia nodes are equipped with two digital sensors: temperature and 3-axis accelerometer, and also a built-in voltage sensor provided as an input channel for ADC (Analog to Digital Converter) but in this project we used the external sensor to measure the temperature and humidity.

Each node reads the sensor values and the data is transmitted via 6loWPAN to the Border Router after receiving sufficient data to send a packet.

Sensor connection with Re-mote

The DHT22 sensor will be installed in the pins 13, 12 & 11 to keep the sensor in operation mode and reduce power consumption. The ADC port mapping must provide the above-mentioned requirements of the sensor. Figure 3.4 shows the configuration mapping of the pin out of the sensor node's ADC channels:

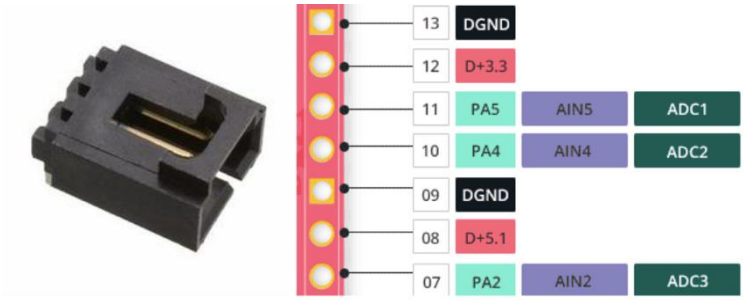


Figure 3.4 RE-Mote terminal pin out mapping.

source: https://github.com/Zolertia/Resources/wiki/RE-Mote

The analog-digital converters ADC0, ADC3 have 5V1 and 5V2 voltage output, while the MSP430 microcontroller operates with 3V1 and 3V2 output voltage ADC1 and ADC7.

The ADCs in the RE-MOTE are ADC1,2 and 3, but on the socket the ADC2 is not physically allowed. With respect to this case, ADC1 suits the minimum voltage supply requirement in the DHT22. As shown in Figure 3.5, the signal conditioning requires a pull-up resistor. The manufacturer suggests this pull-up to ensure the signal status. The resistance value will be chosen for the length of the cable from the MCU to the board.

Operating with 3.3V is recommended from the user manual for this module. To avoid a possible voltage drop and measurement error, it is important that the length of the cable is not greater than 100 cm. The recommendation for the installation of the module is therefore to connect it as a pull-up resistor with a cable measuring less than 20 cm and 10K ohm.

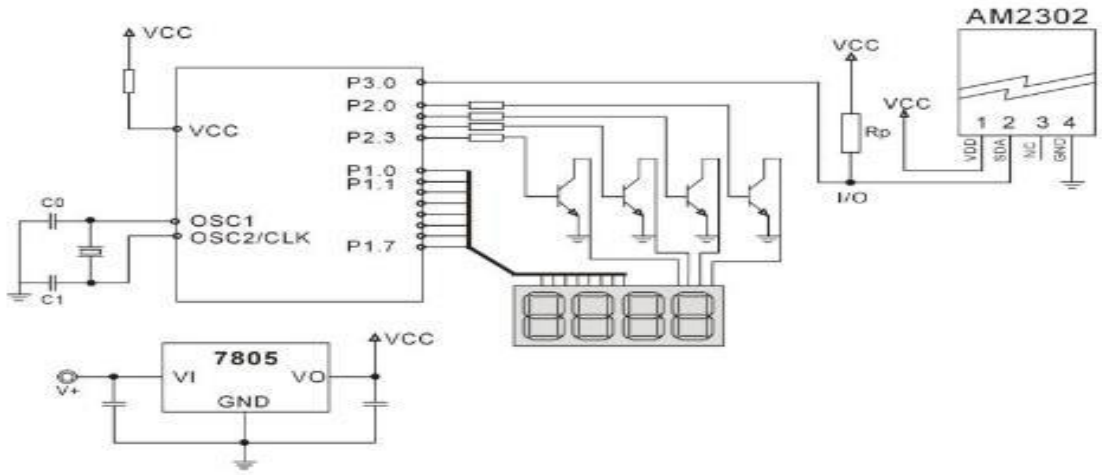


Figure 3.5 Typical schematic for single bus

The output of these modules depends on the synchronization of the MSP430 with the DHT22, otherwise an error will be produced by the output. The structure of the data consists of 40 bits divided into five bytes. The first 16 bits are for humidity, the next 16 bits are for temperature, and the last 8 bits are for parity.

Program architecture

A ‘C’ program was built with the purpose of fulfilling the requirements of this network. When looking at a flowchart [Fig 3.6] the actions of this main thread can be easily understood.

It should also be noted that each of the sensor nodes runs a slightly modified version of the same code in which the package is labelled with an ID, this ID is later used to determine to which data is being transmitted by one of the sensor node devices.

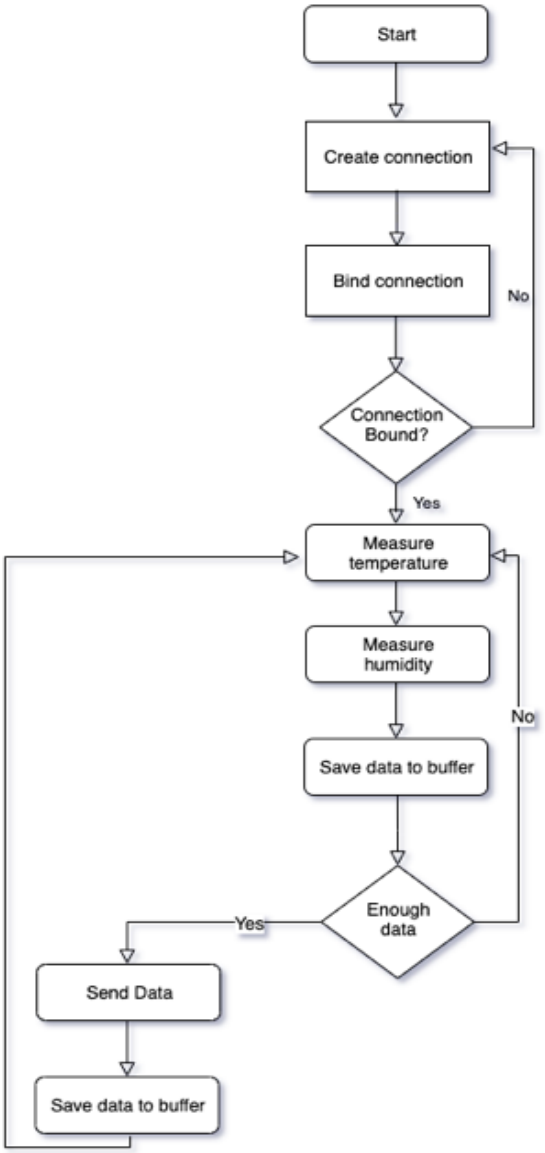


Figure 3.6 Node program flow chart

In the project, we use raspberry Pi for designing an IOT gateway that stores, analyses and transmits sensor data to Grafana and cloud.

3.4.3 Database

A data base server (influxDB) is running inside the Raspberry pi, which allows it to store everything the sensors can send in a non-relational database. This is to ensure that the information collected in our WSN is registered, and that this information is processed for any higher level application. It could be either a front-end app or a desktop app.

Installing InfluxDB

To install the InfluxDB in the raspberry Pi we will use the use the following commands.

```
sudo apt-get update && sudo apt install apt-transport-https curl
curl -sL https://repos.influxdata.com/influxdb.key |
sudo apt-key add -echo "deb https://repos.influxdata.com/debian jessie stable" |
sudo tee /etc/apt/sources.list.d/influxdb.list
sudo apt-get update && sudo apt-get install influxdb
```

Start the InfluxDB service

```
sudo service influxdb start
```

It is possible to verify that everything is working correctly with

```
sudo service influxdb status
```

To test the influxdb installation open the raspberry pi web browser and type:

```
Localhost:8086
```

You may enter the real IP address of your raspberry Pi, instead of writing "localhost" in the address field: Example: 192.168.1.101:8086

3.4.4 Front End for IoT

There are several IoT platforms with some free apps, designed to simplify the system integrators life with respect to user application programming, incorporating pre-designed dashboards to view incoming data, store it, etc. Moreover, these systems are equipped with a high bandwidth for discrete times and signals where there are no continuous signals. There are several on the market but we used Grafana and Ubidots. In the figure 3.7 and 3.8 we can see the dashboard of Grafana and Ubidots.

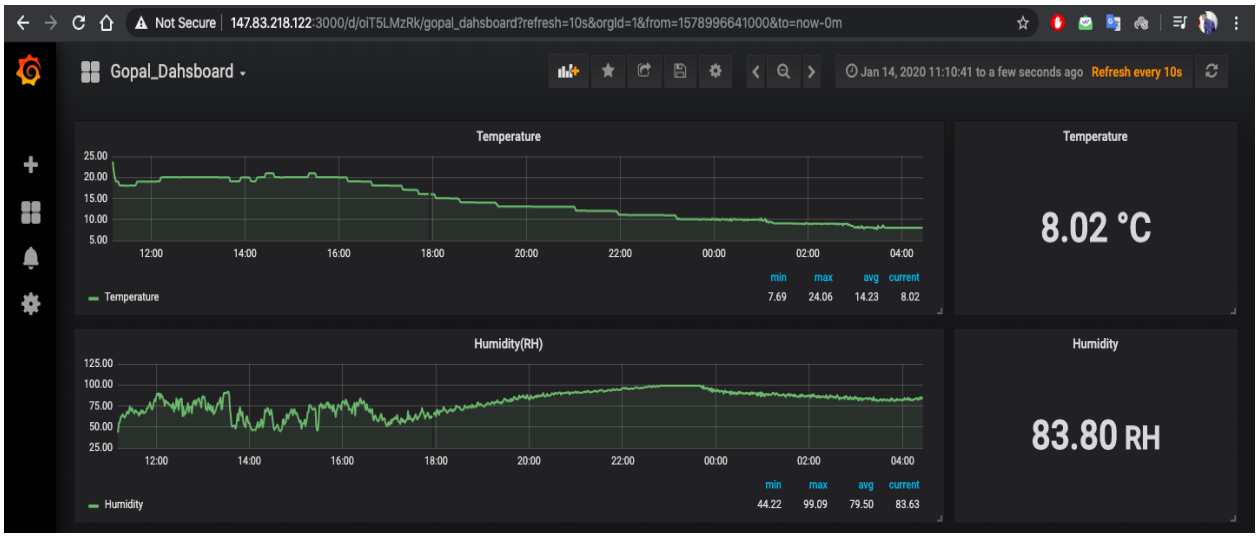


Figure 3.7 Dashboard of the results from the sensor node to the Local server

Be patient, the service needed time on the localhost to initialize, the first login can be made with the following credentials:

URL: http://147.83.218.122:3000/
Username: admin
Password: admin

We select Ubidots IoT platform for the cloud platform which allows users to manage and track sensor node data.

Figure 3.8 Dashboard of the results from the sensor node to the Cloud server

CHAPTER 4. EVALUATION AND RESULTS

This chapter tests on-field device and discusses the functionality of the end. The results obtained in this thesis are from UPC greenhouse where the network is built. The principles used in this section are contrasted with the Agropolis environmental behaviour. In the real scenario, the final node implemented limited the number of sensor nodes to one considering the availability of the sensors. In the proposed system, we use three sensor node and a gateway for testing system that shown in figure 4.5. The Nodes connected to a DHT22 sensor for measurement of temperature and humidity. The nodes are power by a lipo battery.

First our system was tested in laboratory then it was implemented in green house. The laboratory scenario is as follows:

4.1. Laboratory scenario

The data frame must first of all be transported from the node to the border router through one terminal port to the server. The Script for packet transmission follows the next structure:

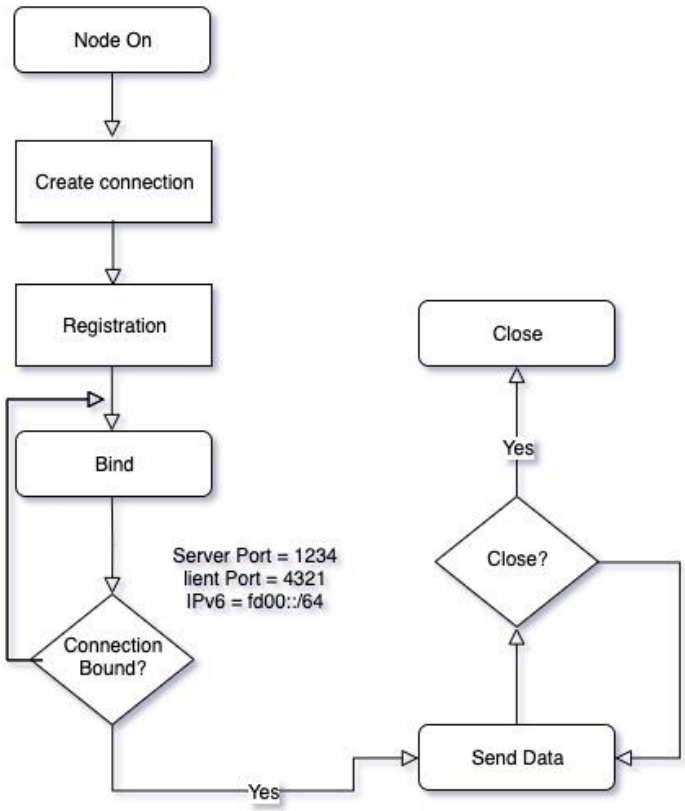


Figure 4.1 Flow Chart for UDP API

The UDP-client followed figure xx and kept to send loop on client port. While, the border router still listens to the channel while the application script running over Python. In client port 8765, where UDP-server interface is configured, the

throughput goes into buffer (1 Kb). For forward transmission on IPv4 using MQTT, the buffer contained an object structure that had to be split and reassembled emulating a JSON structure.

The DHT22 Sensor values are observed in Figure 37, using the sudo make login PORT=/dev / ttyUSB1 function. The loop generated on the script shows the output from the port in use, otherwise the possibility exists of avoiding system specification in case no other node is associated

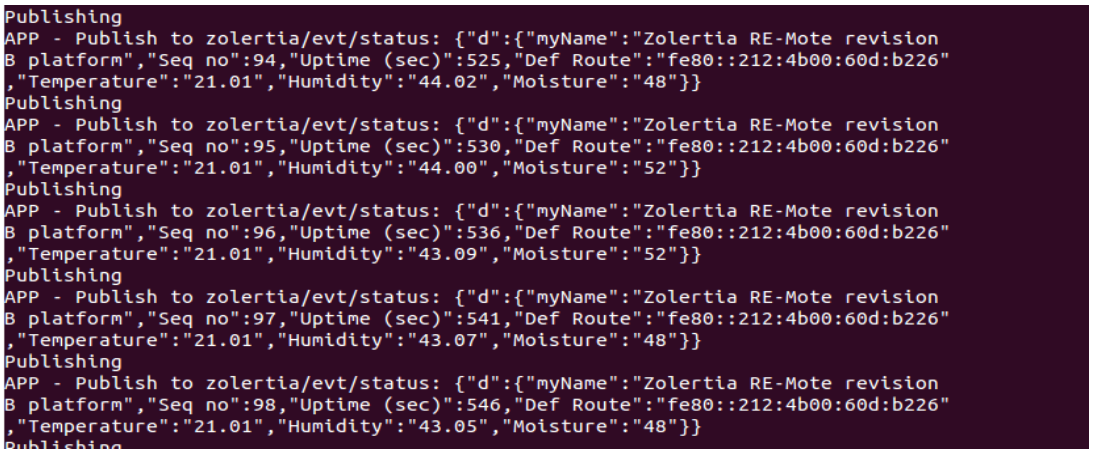


Figure 4.2 Sensor reading from the node

The udp-client reports to the border router using tunneling from the localhost (127.0.0.1) to the bridge. Figure 38 displays the tunnel layout and the packets transported from the 6LoWPAN network to check the bit traffic. See Annex 10 for more information on when building the border router.

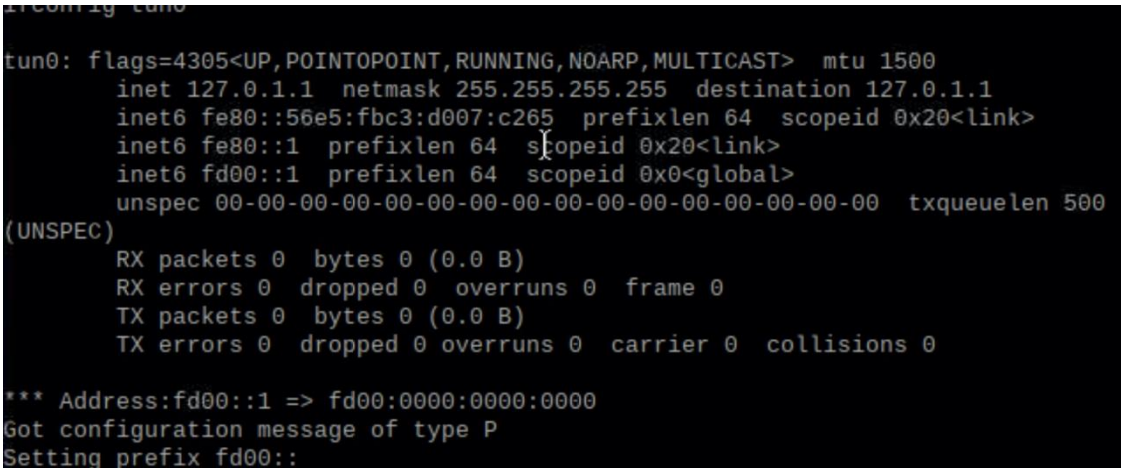


Figure 4.3 Tunnel created using border-router.c

The captured packets are taken from server port where the python script listens, and the value is printed on the command prompt. Using the URL assigned in the script, the MQTT created and sent through client Port.

The border-router.c had been executed on the border router before running the script in annex . The values sent to the MQTT broker via IPv4 are shown in Figure

34 using a static IP address to subscribe to the topic created on the dashboard that it is identified as an entity in order to publish the data.

4.2 Errors and solutions

There are often errors in compiling and debugging this project and this section shows the more common, including the solutions. The errors are scenario based. For example: the node Makefile route, the border router waits for the slip radio connection when the webserver service is started and the node configuration does not allow successful flashing of the memory.

The folder in /var / log/6lbr.log provided a first overview of any warnings or errors that might exist. The big error has texted on the file 6lbr.err.

The waiting for ttyUSB0 connection is a standard log that occurs due to the absence of a node in the port (node unplugged), the USB serial interface connectivity on the border router can be checked using dmesg | grep tty.

The log fetching to a MAC address happens when the slip-radio script does not control the RE-Mote. The bin file did not set the ttyUSB0 as configured serial interface in /6lbr/6lbr.conf.

Open the port in the nodes needed to use the sudo make login /dev / ttyUSBx command depending on the USB node interface. Any serialdump-linux conflict should follow the following commands:

```
cd 6lbr/tool/sky
sudo rm serialdump-linux
sudo make serialdump
sudo mv serialdump serialdump-linux
sudo chmod -x serialdump-linux
```

The motes needed updating and working properly the python magic and intelhex libraries otherwise node programming will not be compiled. A common error concerning the missing previous modules could be fixed to ensure the proper installation of java-sdk and python-pip.

```
sudo update-alternatives --config java
```

The synchronization problems are log as follows: ERROR: timeout waiting for ACK / NACK after ' Synch (0x55 0x55); ' this problem can be solved by manually setting the flashing mode on the RE-MOTE. You can enable programmable mode by pressing and holding the debug button while pressing the user button for a while and releasing the user button followed by the debug button first. The ' system registered redness to read but returned no data ' error occurs when the serial port AMA0 is activated, while USB0 attempts to access a port dispute at the same time.

The compilation of the sensor uses module in the motion transducer script which is not available in the 6lbr downloaded packet. For this purpose it is necessary to create a C file with the algorithm in the next url in case of missing this script motion-sensor.c.

https://github.com/alignan/contiki/blob/iot-workshop/platform/zoul/dev/motion-sensor.c
The DHT22 implementation was achieved by following the example in [51] and the package structure in:

https://github.com/contiki-os/contiki/blob/master/examples/zolertia/zoul/test-dht22.c

Possible problem may appear regarding the time of the sample and the existence of the files, or may appear in the MakeFile.

To interrupt the 6lbr service, both scripts run over /dev / ttyUSB0 and create a link conflict before starting the border router.

4.3 Results

The location and disposition of the material had been chosen the accessibility and availability of the infrastructure. As far as the distance of the packet beamed via radio frequency shown in is concerned (Internet of Things in 5 Days, 2015), there is a maximum range of approximately 200 meters on direct view and power consumption in comparison with other modulesimproves in low power consumption.

We place the gateway near the Wi-Fi Router with wireless internet connection. The Sensors nodes are placed inside the greenhouse for sensing environment information as shown in figure 4.5.

Figure 4.5(a) Green House. Figure 4.5(b) Gateway. Figure 4.5(c) Sensor Node.

In the script, we set the sampling time for 10 minutes, so that the gateway sends a command every 10 minutes to request sensor node data. Sensor node takes the sensor measurement after receiving the gateway command, and then transmits it back to the gateway. Inside the gateway, the data of the sensor node will be saved in influxdB and visualized as a graph and transmitted continuously to the cloud where it is visualized as shown in the figure 4.6. This figure illustrates environmental information gathering data as a greenhouse area from 12 January to 16 June 2020. During the time period, data from sensor nodes are successfully

sent to the gateway with a data loss of 17 percent. Users could get a better understanding of the greenhouse area environment according to the graphs.

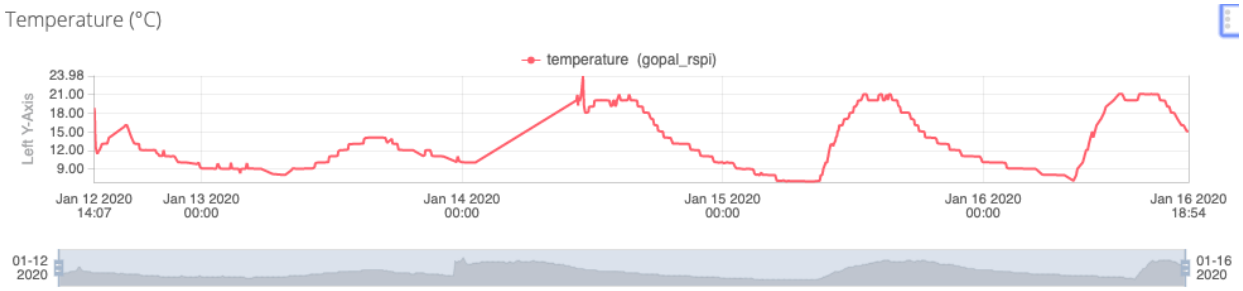


Figure 3.6(a) Agropolis results from the 12 to 16 of January 2020

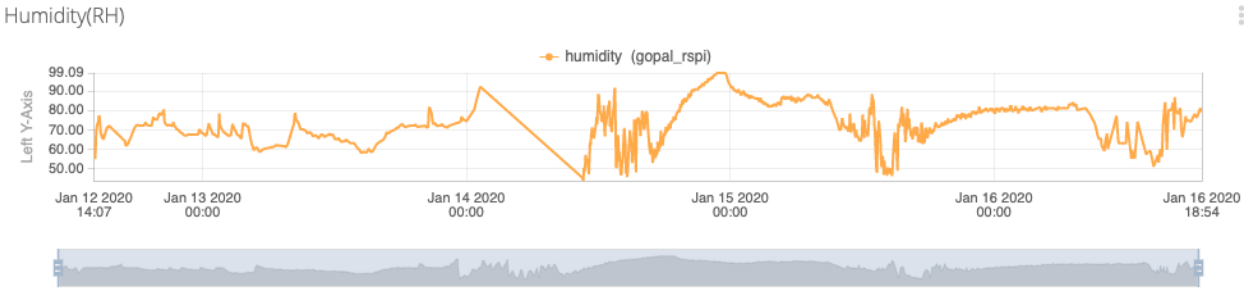


Figure 4.6(b) Agropolis results from the 12 to 16 of January 2020

The correlation between variables after processing on Python is shown in Figure 4.7. The low correlation between temperature and humidity can be confirmed.

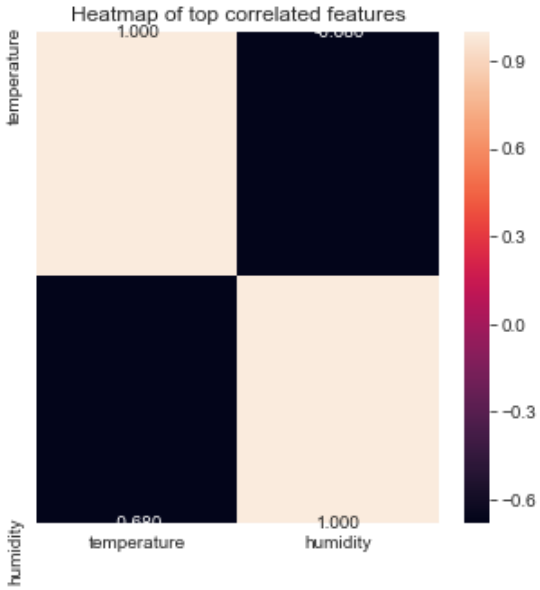


Figure 4.7 Correlation matrix

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1. Conclusions

The general aim of this thesis has been development a Wireless Sensor Network, continuous monitoring using IoT. The study is using wireless network sensor technology to implement a monitoring system in the agricultural sector. Gateway and sensor nodes contribute to the proposed system. it is designed for supporting farmer. The system of environmental monitoring for agriculture has been used to increase crop yields and to support farmers in decision-making on the basis of the environmental information provided and is the basic monitoring system for agriculture and greenhouse activities.

The evolution of the technology creates big changes in the object's industry and lifetime. Implementations of new devices are required to optimize and improve the process in all the departments. The agricultural sector is aware of those developments that involve updating of the techniques about the increasing demand for better quality products on the market. Monitoring of greenhouse crops moderates the potential impacts that the plant dealt with during certain periods of the year. Presence of plagues and parasites is related to weather changes and crop water retention.

The data acquisition of variables correlated with possible environmental changes could give the farmer the possibility of creating a contention plan to improve the production of the crop.

In my opinion, due to the benefits of free band (ISM) space and low power consumption, the network presented in this memory using the 6LoWPAN protocol has to be spread among the users. Implementing the IPv6 protocol offers numerous benefits, in this case many applications with multiple nodes used on this network protocol.

To greenhouse and small-scale farms, the system can be regarded as no more than 2 hectares. Hence, the system's efficiency may be influenced by the fading and shadowing of the plants and staff in the field. I observed in practice that node orientation and cumulative obstacle can impact the results.

In the case of medium and large territorial extensions, a more robust device with a direct connection to the IoT cloud or server is suggested. New technology trends such as NB-IoT and LoRa have been implemented for open spaces in many companies, given the robustness of the protocol against shadowing and fading that has affected the network. Because of the high market demand, and as shown in the literature review of this protocol, the agricultural sector tends toward LoRa integration. The NB-IoT, meanwhile, is still in the process of development, but that the solutions using this method suggests a possible future application.

5.2 Future Work

This work opens up new doors for future studies. An example could be the implementation of this network in large field and the calculation of the packet losses about the plant's shadowing. Furthermore, the machine's external condition effect, how salinity and water can influence the REMote, and the autonomy can be explored after applying the low power consumption mode.

Further field installation with strong base station will be deployed in future work. Security and power management of network data with efficient algorithms will also be given higher priority in our future work. Also we will integrate a warning system that sends to farmer a warning SMS message due to the data.

Considering other end-customer requirements, integrating other sensors, programming and executing differ ent scripts about their alternatives on the market. In addition, to reduce the power consumption, the modification and evaluation of other MAC scripts can be investigated using topologies for parent-son regarding the spot.

Finally, the research of the possibilities of using a LoRa module and the network implementation with this technology will produce meaningful results in the agricultural sector.

Bibliography

1. *IoT architecture*. (2019, July 16). Retrieved from https://www.avsystem.com/blog/what-is-iot-architecture/
2. *"The evolution of WIFI"*. (2017, October 16). Retrieved from EE Publishers: https://www.ee.co.za/article/the-evolution-of-wifi.html
3. *“Topology Options | Bluetooth Technology Website.”*. (2019, June 27). Retrieved from https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/
4. (2020, January 10). Retrieved from Home: https://www.threadgroup.org/
5. Beinschob, P., & Reinke, C. (2015). “Graph SLAM based mapping for AVG localization in large-scale warehouses,”. *2015 IEEE 11th International Conference on Intelligent Computer Communication and Processing*, 245-248.
6. Brenman, L. (2018, June 13). *API Builder and MQTT for IoT – Part 2*. Retrieved from https://devblog.axway.com/apis/api-builder-mqtt-iot-part-2/
7. *GoogleIPv6*. (2020, 01 26). Retrieved from https://www.google.com/intl/en/ipv6/statistics.html
8. Hao, Y., & Foster, R. N. (2008). “Wireless body sensor networks for health-monitoring applications”. *Physiological Measurement, vol. 29, no. 11,* .
9. *Internet of Thing*. (2020). Retrieved from GSMA: https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/
10. Internet of Things in 5 Days. (2015). In A. Colina, A. Vives, A. Bagula, & M. Zennaro.
11. *IoT Agenda*. (2020). Retrieved from https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT
12. Joshi, P. P., Kanade, S., & Joshi, S. P. (2017). Wireless Sensor Network and Monitoring of Crop Field. *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE*, 23-28.
13. kim, Y., Evans, R., & Iversen, W. (July, 2008). "Remote Sensing and Control of an Irrigation System Using a Distributed Wireless Sensor Network". *IEEE*, 1379–1387.
14. Matilla, A. S. (2018). "IoT Connectivity". Spain.
15. Monazzah, A. H., Safaei, B., Bafroei, M. B., & Ejlali, A. (2017). Reliability Side-Effects in Internet of Things Application Layer Protocols. *International Conference on System Reliability and Safety.* Milan, Italy.
16. Paredes-Parra, J. m., Garcia-Sanchez, A. J., Mateo-Aroca, A., & Molina-Garcia, A. (2019, March 6). An Alternative Internet-of-Things Solution Based on LoRa for PV Power Plants: Data Monitoring and Management.
17. *RabbitMQ for beginners*. (2015, 05 18). Retrieved from CloudAMQP: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html
18. Reinbacher, T., Leon, M. B.-d., & Wee, D. (2018). *The IoT as a growth driver.* McKinsey & Company.
19. Shelby, Z., & Bormann, C. (2011, 06 13). 6LoWPAN: The wireless embedded Internet – Part 3: 6LoWPAN architecture, protocol stack & link layers.

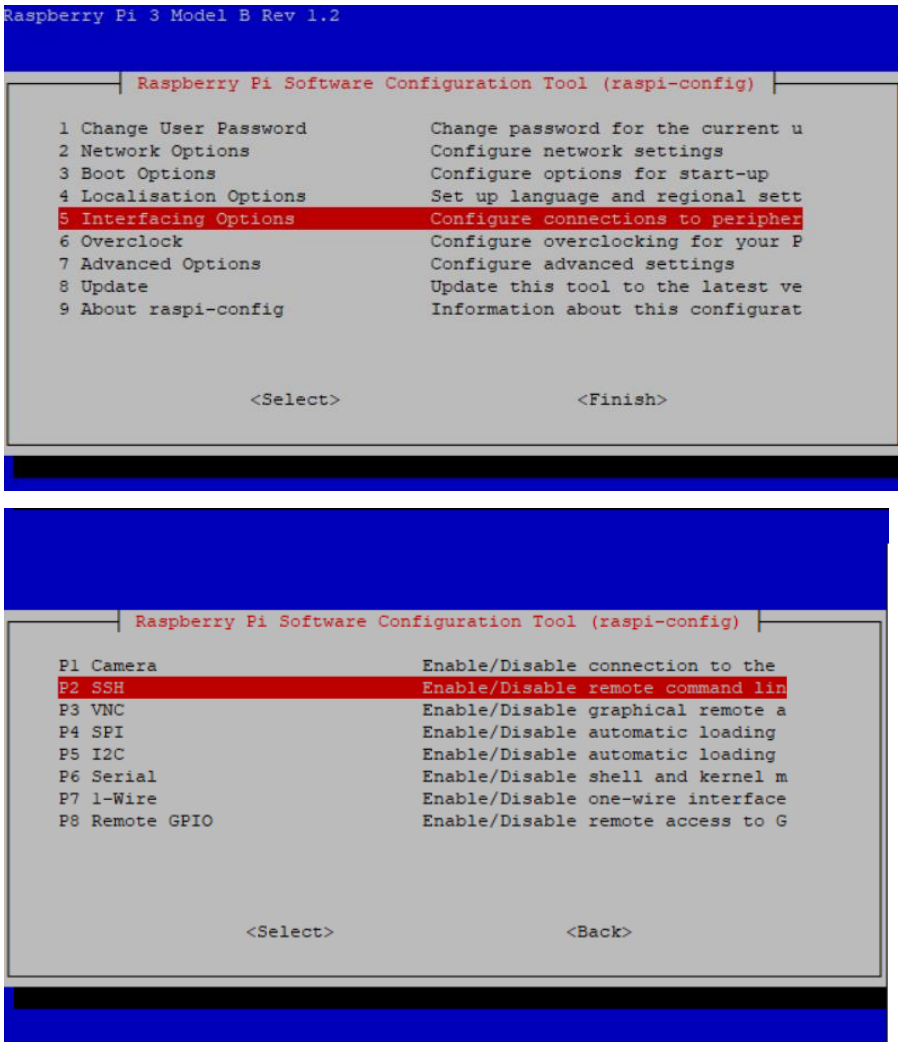
20. Sikder, A. K., Petracca, G., Aksu, H., Jaeger, T., & Uluagac, A. (2018). “A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications”.
21. Templeton, C., Carlson, N., Leon-Garcia, A., & Widjaja, I. (2018). "Communication Networks". *Ref. Modul. Fife Sci*, 1-24.
22. *What is CoAP protocol IoT | CoAP Architecture, message format*. (n.d.). Retrieved from Home of RF and Wireless Vendors and Resources: https://www.rfwireless-world.com/IoT/CoAP-protocol.html

ANNEXES

Annex 1 Node configuration code, click to open the object.

```
/*---------------------------------------------------------------------------*/
#include "contiki.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-udp-packet.h"
#include "sys/ctimer.h"
#include <stdio.h>
#include <string.h>
#include "dev/adc-zoul.h"
#include "dev/zoul-sensors.h"
#include "dev/dht11.h"
#include "dev/button-sensor.h"
/*---------------------------------------------------------------------------*/
/* Enables printing debug output from the IP/IPv6 libraries */
#define DEBUG DEBUG_PRINT
#include "net/ip/uip-debug.h"
/*---------------------------------------------------------------------------*/
/* Default is to send a packet every 10 minutes */
#define SEND_INTERVAL (600 * CLOCK_SECOND)
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* This is the UDP port used to send and receive data */
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
/*---------------------------------------------------------------------------*/
```

Annex 2 SSH Enable in Raspbian OS



Annex 3 Python script UDP forward data to Grafana and Ubidots

```
import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient
from typing import NamedTuple
import re
import json
import requests
import time


# Change accordingly to the MQTT Broker and topic you want to subscribe
# In the example it would be either "test.mosquitto.org" or "fd00::1" if
# running a mosquitto broker locally
MQTT_URL   = "fd00::1"
MQTT_TOPIC_EVENT = "zolertia/evt/status"
MQTT_TOPIC_CMD   = "zolertia/cmd/leds"

INFLUXDB_ADDRESS = "localhost"
```

```
INFLUXDB_USER = "root"
INFLUXDB_PASSWORD = "root"
INFLUXDB_DATABASE = "zolertia_db"

TOKEN = "BBFF-d43FgxWMgUs6WWR3BKOxAQtFdu9rEl"  # Put your TOKEN
here
DEVICE_LABEL = "Gopal_Rspi"  # Put your device label here
VARIABLE_LABEL_1 = "temperature"  # Put your first variable label here
VARIABLE_LABEL_2 = "humidity"  # Put your second variable label here
VARIABLE_LABEL_3 = "moisture"  # Put your second variable label here

influxdb_client = InfluxDBClient(INFLUXDB_ADDRESS, 8086,
INFLUXDB_USER, INFLUXDB_PASSWORD, None)

class SensorData(NamedTuple):
    location:str
    measurement:str
    temperature:float
    humidity:float
    moisture:float


def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe(MQTT_TOPIC_EVENT)
    print("Subscribed to " + MQTT_TOPIC_EVENT)
    client.subscribe(MQTT_TOPIC_CMD)
    print("Subscribed to " + MQTT_TOPIC_CMD)

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    #print(msg.topic + " " + str(msg.payload))
    sensor_data  =  _parse_mqtt_message(msg.topic,  msg.payload.decode("utf-
8"))
    if sensor_data is not None:
        _send_sensor_data_to_influxdb(sensor_data)
        ubidots_payload = build_ubidots_payload(VARIABLE_LABEL_1,
VARIABLE_LABEL_2, VARIABLE_LABEL_3, sensor_data)
        print("Attemping to send data to the ubidots")
        post_request(ubidots_payload)
        print("Finished to send data to the ubidots")


def _parse_mqtt_message(topic, payload):
   # match = re.match(MQTT_TOPIC_EVENT, topic)
   # if match:
    location = "evt"
    measurement = "status"
    data_src = json.loads(payload)["d"]
    temperature = data_src["Temperature"]
```

```
    humidity = data_src["Humidity"]
    moisture = data_src["Moisture"]
    return SensorData(location, measurement, float(temperature), float(humidity),
float(moisture))
 #   else:
 #       return None

def _send_sensor_data_to_influxdb(sensor_data):
    json_body = [
        {
            "measurement": sensor_data.measurement,
            "tags": {
                "location": sensor_data.location
            },
            "fields": {
                "temperature": sensor_data.temperature,
                "humidity" : sensor_data.humidity,
                "moisture" : sensor_data.moisture
            }
        }
    ]

    influxdb_client.write_points(json_body)
    print("Inserted to the influxdb.")

def build_ubidots_payload(variable_1, variable_2, variable_3, sensor_data):
    ubidots_payload = {variable_1:sensor_data.temperature,
                       variable_2:sensor_data.humidity,
                       variable_3:sensor_data.moisture
                       }
    return ubidots_payload

def post_request(payload):
    # Creates the headers for the HTTP requests
    url = "http://industrial.api.ubidots.com"
    url = "{}/api/v1.6/devices/{}".format(url, DEVICE_LABEL)
    headers = {"X-Auth-Token": TOKEN, "Content-Type": "application/json"}

    # Makes the HTTP requests
    status = 400
    attempts = 0
    while status >= 400 and attempts <= 5:
        req = requests.post(url=url, headers=headers, json=payload)
        status = req.status_code
        attempts += 1
        time.sleep(1)

    # Processes results
    if status >= 400:
        print("[ERROR] Could not send data after 5 attempts, please check \
```

```
            your token credentials and internet connection")
        return False

    print("[INFO] request made properly, your device is updated")
    return True


def _init_influxdb_database():
    databases = influxdb_client.get_list_database()
    if len(list(filter(lambda x: x["name"] == INFLUXDB_DATABASE, databases)))
== 0:
        influxdb_client.create_database(INFLUXDB_DATABASE)
    influxdb_client.switch_database(INFLUXDB_DATABASE)

_init_influxdb_database()

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

print("connecting to " + MQTT_URL)
client.connect(MQTT_URL, 1883, 60)
client.loop_forever()
```

Annex 4 Border router script

```
* \file

* border-router

#include "contiki.h"

#include "contiki-lib.h"

#include "contiki-net.h"

#include "net/ip/uip.h"

#include "net/ipv6/uip-ds6.h"

#include "net/rpl/rpl.h"

#include "net/rpl/rpl-private.h"

#if RPL_WITH_NON_STORING

#include "net/rpl/rpl-ns.h"

#endif /* RPL_WITH_NON_STORING */

#include "net/netstack.h"

#include "dev/button-sensor.h"

#include "dev/slip.h"

#include <stdio.h>

#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>
#define DEBUG DEBUG_NONE
#include "net/ip/uip-debug.h"
static uip_ipaddr_t prefix;
static uint8_t prefix_set;
PROCESS(border_router_process, "Border router process");
```

Annex 5 Setting up Grafana on the Raspberry Pi

1. Before we start installing Grafana to the Raspberry Pi, we should first make sure that all the packages currently installed are up to date.

We can do this by executing the two commands that follow. Such commands update the package list, and then upgrade all the programs installed to their latest versions.

```
sudo apt update
sudo apt upgrade
```

2. The first thing we need to do to install Grafana into the Raspberry Pi is download the Grafana package from their website. Run the two commands you use to download the Grafana kit, and install it.

```
wget https://dl.grafana.com/oss/release/grafana_6.4.4_armhf.deb
sudo dpkg -i grafana_6.4.4_armhf.deb
```

The first command downloads to your Raspberry Pi the corresponding Grafana Debian package file.

The second makes use of the “dpkg” package manager to install the package that we just downloaded to the operating system.

3. Our next step is getting Grafana started at startup. Thankfully for us, Grafana comes with a service file that is systemd.

All we need to do is run the following command to enable Grafana to start at boot time.

```
sudo systemctl enable grafana-server
```

This command will tell the systems service manager to enable the service file called “grafana-server.service”.

4. Finally, let’s start up the Grafana server software by running the command below in the Pi’s terminal.

5. Now that we have Grafana built on your Raspberry Pi, we can now use their web interface.

6. With your IP address handy, go to the following web address. Grafana’s web interface sits on port 3000 of your Raspberry Pi’s IP address.

Make sure that you replace “<IPADDRESS>” with the IP that you retrieved in the previous step.

```
http://<IPADDRESS>:3000
```

7. The first thing you will see when loading up Grafana is a login screen.

On this screen, you will be able to login using the default admin user that was created when you first installed Grafana to your Raspberry Pi.

The username for this user is “admin” and the password is “admin“.