

Master's programme in Communication Engineering

Design and Evaluation of a LoRaWAN Smart Agriculture System Based on Microsoft Azure

Pablo del Arco Ortiz

© 2023

This work is licensed under a [Creative Commons](#)
“Attribution-NonCommercial-ShareAlike 4.0 International” license.



Author Pablo del Arco Ortiz

Title Design and Evaluation of a LoRaWAN Smart Agriculture System Based on
Microsoft Azure

Degree programme Communication Engineering

Major MSc. in Communication Engineering

Supervisor Prof. Jyri Hämäläinen

Advisor Dr Alexis Dowhuszko

Collaborative partner Aalto University

Date July 31 2023

Number of pages 78+7

Language English

Abstract

This Master's thesis explores the design and evaluation of a LoRaWAN-based Smart Agriculture system using Microsoft Azure. The system integrates various IoT sensors to monitor essential agricultural parameters, creating a comprehensive network for data collection. Sensor data is simulated through the LoRaWAN (LWN) Simulator and received by the ChirpStack network server. The data is then processed and decoded via Node-RED, transmitted through the Message Queuing Telemetry Transport protocol (MQTT), and presented on the Azure IoT Central dashboard for intuitive visualization. The study also involves a comparative analysis of various Low Power Wide Area Network (LPWAN) technologies and Internet of Things (IoT) communication protocols, aiding in the selection of the most efficient options for Smart Agriculture systems.

Keywords LoRaWAN, Microsoft Azure, Smart Agriculture, IoT Sensors, LPWAN Technologies, IoT Communication Protocols, Cloud Computing

Preface

I would like to express my sincere thanks to my parents, Juan Andrés and Paloma, for their consistent support during my master's degree. I am also deeply grateful to my friends, particularly Ana and Gonzalo, who have been with me every step of the way. Their encouragement and belief in me have been incredibly valuable. I extend my gratitude to my advisor and supervisor for guiding me throughout the completion of this thesis. And finally, to everyone who has contributed and supported me along this journey.

Otaniemi, July 31 2023

Pablo del Arco Ortiz

Contents

Abstract	3
Preface	4
Contents	5
Abbreviations	7
1 Introduction	8
1.1 Motivation	8
1.2 Objectives and scope of the thesis	8
1.3 Outline of the thesis	9
2 Literature review	11
2.1 Overview of Existing Solutions for Agriculture Monitoring	11
2.2 Review of LPWAN Technologies	12
2.2.1 LoRaWAN	12
2.2.2 Sigfox	16
2.2.3 NB-IoT	17
2.3 Alternative IoT Communication Technologies	19
2.3.1 IoT Satellite Communication	19
2.3.2 IoT UAV Swarm System	22
2.4 IoT communication protocols	23
2.4.1 Message Queuing Telemetry Transport (MQTT) protocol	24
2.4.2 Constrained Application Protocol (CoAP)	26
2.4.3 Advanced Message Queuing Protocol (AMQP)	28
2.5 Network Servers	30
2.5.1 ChirpStack	30
2.5.2 The Things Network	32
2.5.3 Comparing ChirpStack and The Things Network	33
2.6 Cloud Platform: Microsoft Azure	34
2.6.1 Leveraging Microsoft Azure in a LoRaWAN Smart Agriculture System	35

3	System Design and Implementation	37
3.1	Description of the Proposed System & Architecture	37
3.1.1	Sensor End Device	37
3.1.2	Gateway	38
3.1.3	Network Server	39
3.1.4	Application Server	39
3.2	Selection of LPWAN technology	39
3.3	IoT Protocol: MQTT	40
3.4	IoT Platform: Azure IoT Central	41
3.5	Setup & configuration of Microsoft Azure cloud platform	42
3.6	Development of Software & Configuration of LoRaWAN network .	45
3.6.1	LWN Simulator	46
3.6.2	ChirpStack	55
3.6.3	MQTT Broker and Node-RED	59
3.7	Configuration and Development of Azure IoT Central	63
4	Results, Visualization & Automations	68
4.1	Visualization	68
4.2	Automations	70
5	Conclusions	72
5.1	Summary of the study	72
5.2	Implications of the study	72
5.3	Recommendations for future research	72
	References	74
A	Installation Guide for LWN Simulator	79
B	ChirpStack Installation	81
C	Installation of MQTT Broker and Node-RED using Docker	83

Abbreviations

LPWAN	Low Power Wide Area Network
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
CoAP	Constrained Application Protocol
AMQP	Advanced Message Queuing Protocol
QoS	Quality of Service
LoRa	Long Range
OTAA	Over-the-Air Activation
ABP	Activation by Personalization
TCP	Transport Control Protocol
UDP	User Datagram Protocol
CSS	Chirp Spread Spectrum
SSL	Secure Sockets Layer
NB-IoT	Narrowband Internet of Things
OFDMA	Orthogonal Frequency-Division Multiple Access
LR-FHSS	Long Range-Frequency Hopping Spread Spectrum
D2D	device-to-device
UAV	Unmanned Aerial Vehicle

1 Introduction

1.1 Motivation

Over recent years, agriculture has undergone dramatic change owing to IoT technology developments. Smart agriculture systems have emerged as key innovations to maximize crop production, resource allocation and overall farm efficiency. Utilizing IoT sensors and communication technologies, these systems enable real-time field data collection for farmers' decision-making processes. Low Power Wide Area Networks (LPWANs) have become essential parts of the contemporary digital ecosystem because they offer long-distance connection solutions while consuming the least amount of power.

Agriculture is one of the oldest and most essential industries, providing healthy natural products to society. Implementation of IoT technologies - particularly LPWANs - has many advantages for this sector, making it possible to update and digitize entire agricultural systems using cost-effective tools.

The global need for food combined with sustainable farming practices has underscored the necessity of IoT technology in agriculture. Utilizing IoT platforms connected to sensors, farmers can monitor essential parameters like soil moisture, temperature, humidity and crop health using precision agriculture techniques that make use of IoT data. Utilizing resources like water, fertilizers and pesticides more effectively leads to maximum impact with efficient resource utilization.

The motivation behind this Master's thesis stems from the need to design and evaluate a LoRaWAN-based Smart Agriculture system using Microsoft Azure as its cloud platform. By exploring integration among IoT sensors, LPWAN communication, cloud services, and related IoT services this research aims to foster efficient and sustainable agricultural practices.

1.2 Objectives and scope of the thesis

This Master's thesis seeks to design and evaluate a LoRaWAN-based Smart Agriculture system using Microsoft Azure as its cloud platform. The proposed system aims to allow real-time monitoring of agricultural parameters while offering actionable insights to farmers for improved decision-making processes. Specific objectives of the thesis include:

- Investigating and comparing various LPWAN technologies and IoT communication protocols suitable for Smart Agriculture systems.
- Design and implementation of LoRaWAN sensor network in agriculture fields.
- Simulation of sensor data using LWN Simulator and integration into ChirpStack network server.

- Developing data processing and decoding mechanisms using Node-RED.
- Implementing data transmission via MQTT (Message Queuing Telemetry Transport).
- Visualizing collected data using Azure IoT Central dashboard.

The scope of this thesis is focused on the simulation, design, and evaluation of a LoRaWAN-based Smart Agriculture system. This includes simulating sensor integration, simulating IoT sensor data generation, developing data processing mechanisms and decoding schemes for processing sensor data, as well as visualizing this simulated data on the Azure IoT Central platform. It is worth mentioning that the scope of this thesis refers to the simulation and evaluation of the system, rather than its actual implementation in a real-world agricultural setting.

By simulating the proposed system, this research seeks to evaluate its feasibility, performance and effectiveness using Microsoft Azure as a testbed. Comparative analysis between various LPWAN technologies and IoT communication protocols will offer valuable insight for selecting optimal options in a simulated Smart Agriculture scenario.

1.3 Outline of the thesis

- **Chapter 1: Introduction.** This chapter serves as an introduction to the research topic, presents its objectives and scope, and details the overall structure of the thesis.
- **Chapter 2: Literature Review.** This chapter surveys existing literature regarding Smart Agriculture systems, LPWAN technologies and IoT communication protocols. It explores their current state-of-the-art status and highlights gaps and challenges within these areas of study.
- **Chapter 3: System Design and Implementation.** This chapter presents the design and implementation details of a LoRaWAN-based Smart Agriculture system. It covers the selection of LPWAN technologies and IoT communication protocols, the design of the sensor network, the simulation of sensor data, the development of data processing mechanisms, and the implementation of data transmission protocols.
- **Chapter 4: Results, Data Analysis & Visualization.** This chapter explores the visualization of collected data using the Azure IoT Central dashboard. Furthermore, it presents an evaluation methodology and evaluates the performance and effectiveness of the proposed system based on accuracy, reliability and power consumption criteria.

- **Chapter 5: Conclusions.** The final chapter presents a summary of the findings presented throughout this thesis, highlighting its contributions and limitations, while suggesting potential avenues of future work in LoRaWAN-based Smart Agriculture systems. By following this structure, this thesis seeks to offer an in-depth examination of designing and assessing a LoRaWAN-based Smart Agriculture system using Microsoft Azure.

2 Literature review

2.1 Overview of Existing Solutions for Agriculture Monitoring

This subsection presents a comprehensive review of the current solutions for rural area monitoring utilizing Internet of Things (IoT) systems. These solutions are primarily focused on several facets of smart agriculture, including yield forecasting, decision-making support systems, greenhouse surveillance, and the determination of water status in vineyards.

Gupta and Nahar [1] introduced a hybrid machine learning model integrated with IoT for yield forecasting in smart agriculture systems. Their methodology involves preprocessing, feature selection, and classification using a variety of sensors. They utilized a two-tier machine learning model that comprises the Adaptive k-Nearest Centroid Neighbour Classifier (aKNCN) and the Extreme Learning Machine algorithm (ELM), with a modified Butterfly Optimization algorithm (mBOA) to enhance performance.

A LoRaWAN-based smart agriculture decision support system for optimum crop yield was proposed by Arshad et al. [2]. This system, which consists of intelligent sensor modules, a controlled fertilizer module and a smart irrigation system, makes use of embedded system automation and the Internet of Things. For real-time monitoring, the sensor data is sent to the cloud via the Long Range (LoRa) communication protocol, and an Android application is created for remote access.

Dwi Putra et al. [3] presented an IoT monitoring system design based on LoRaWAN architecture for smart greenhouses. The proposed system architecture includes a LoRa sender, LG02 Dragino, DHT22 sensor, soil moisture sensor, and MQTT server running on a Linux server. The proposed system has been evaluated on a real farm at a laboratory scale and can support sustainable agriculture through greenhouse technology.

Valente et al. [4] proposed a LoRaWAN IoT system for smart agriculture in vineyards, specifically for water management. The system connects three groups of sensors for soil, plant, and environment through a wireless protocol to monitor water potential using the soil-plant-atmosphere continuum (SPAC) model. The article also covers the possibility of integrating IoT technologies with unmanned aerial systems (UAVs) to sense and automate agricultural lands.

In conclusion, these existing solutions cover different aspects of smart agriculture, such as yield prediction, decision support systems, smart greenhouses, and vineyard water status determination. Each system has its unique features and applications, utilizing IoT technologies and wireless communication protocols like LoRaWAN to address specific challenges in rural area monitoring.

2.2 Review of LPWAN Technologies

Low-Power Wide-Area Network (LPWAN) technologies have emerged as a promising solution for IoT applications that require long-range communication, low power consumption, and cost-effective deployment. These technologies are particularly suited for applications in remote or challenging environments, where traditional communication networks may not be practical or efficient. This section provides an overview of various LPWAN technologies and their potential applications in smart agriculture.

2.2.1 LoRaWAN

Long Range Wide Area Network, or LoRaWAN, is a protocol created for wireless, battery-powered devices in a local, regional, or international network. It focuses on important IoT requirements such geolocation services, mobility, and secure bi-directional communication. [5].

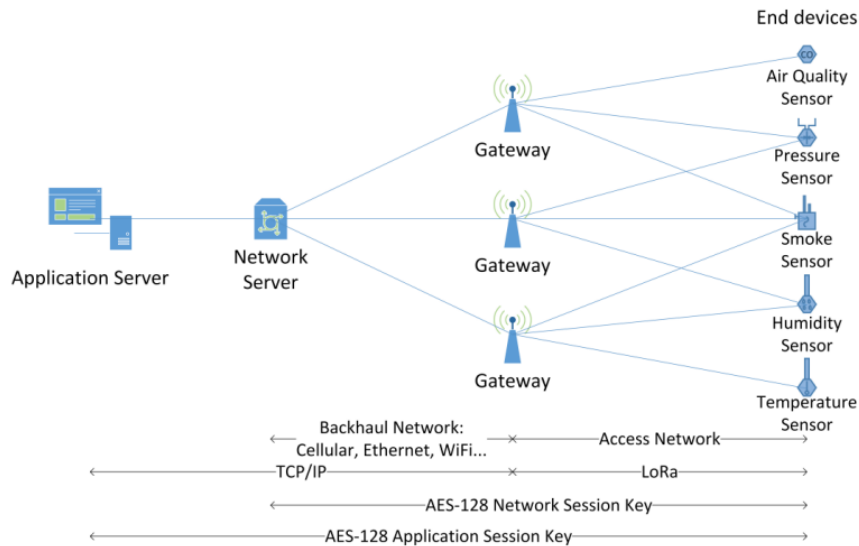


Figure 1: LoRaWAN Architecture [6]

The LoRaWAN architecture's star-of-stars topology enables message relay between end devices and a primary network server through gateways. These can function as bridges, connecting to the network server using conventional IP connections and translating RF packets from the devices into IP packets. ISM (Industrial, Scientific, and Medical) bands are used in this wireless communication. [5].

Three Different Classes

LoRaWAN defines three device classes: Class A, Class B, and Class C [7].

- **Class A:** All LoRaWAN devices must implement Class A. These devices allow for bi-directional communication. After a Class A device sends an uplink transmission, it opens two receive windows at specified times. This is a low-power, asynchronous communication method.
- **Class B:** Class B devices open additional receive windows in addition to Class A devices at predefined periods. As a result, the server can schedule downlink broadcasts from the device by knowing when the device is listening.
- **Class C:** This is the least efficient class since the LoRa nodes are permanently listening and therefore the gateway can transmit downlink messages at any time

Authentication Methods

LoRaWAN offers 2 authentication modes to register devices on the network:

- **OTAA (Over-the-Air Activation):** This is the most secure method as with every new connection, the Application Session Key (AppSKey) and Network Session Key (NwSKey) are erased and new ones are generated. Prior to transmitting information, the device must carry out a joining procedure through its personalization. For this, several parameters must be configured [8]:
 - **DevEUI:** This is a unique identifier that each device possesses. Similar to a MAC address, this identifier uses IEEE EUI64 addresses.
 - **AppEUI:** This is an identifier that is stored in the device's memory before it is activated. It helps identify the application server and resembles a port number. Like DevEUI, it uses IEEE EUI64 addresses.
 - **AppKey:** Unlike the previous parameters, AppKey is a 128-bit key that allows the generation of the NwSKey and AppSKey session keys. Both the server and the device have access to this key.

Once the joining procedure is carried out, the device sends a message to join the network, which contains the AppEUI, AppKey, and DevEUI. If the join request is approved, the server replies with a response containing the following details:

- **DevAddr:** This is a 32-bit number that allows the device to be identified within the network. It resembles an IP address.
 - **NetID:** This is a network identifier.
 - **AppNonce:** The server provides the device with this value to generate the NwSKey and AppSKey keys.
- **ABP (Activation by Personalization):** This method is simpler as it does not carry out the join request process. This is possible because the DevAddr, NwSKey, and AppSKey are all stored in the device, thus it can connect to the network without the need to configure the DevEUI, AppEUI, or AppKey. In

other words, the device already has the necessary parameters registered to join the network [8].

In this work, the OTAA activation method will be used because, although ABP is a simpler process, it does not offer as much security as OTAA does.

Security

While security is not an element as such, it is intrinsically present as it is essential to protect and encrypt all transmitted information. The LoRaWAN network makes use of dynamic AES-128 encryption for data transmission. Two security layers can be distinguished in the LoRaWAN network [9]:

- **TCP/IP SSL LoRaWAN Network Layer:** Makes use of a 128-bit AES key called Network Session Key (NwkSKey) to secure the data transmitted on the network.
- **TCP/IP SSL Secure Payload Application Layer:** Makes use of a 128-bit AES key called Application Session Key (AppSKey) that allows end-to-end encryption of all information so that the network operator does not have access to the data.

LoRa

LoRa (Long Range) is a digital wireless data communication technology that provides long-range, low-power communications for IoT applications. It utilizes a proprietary spread spectrum modulation based on Chirp Spread Spectrum (CSS).

In terms of technical specifications, LoRa offers three configurable bandwidth options: 125 kHz, 250 kHz, and 500 kHz. Its maximum transmission speed is 50 kbps for downlink and 0.3 kbps for uplink. LoRa has high sensitivity, reaching up to -168 dB, which enables reliable data reception. The devices powered by LoRa can have a lifespan of over 10 years in certain scenarios [10].

LoRa operates in different frequency bands depending on the region. In Europe, it operates in the 868 MHz band, in America at 915 MHz, and in Asia at 433 MHz. The coverage range of LoRa varies between 10 and 15 km, depending on the deployment area. This extensive coverage range allows a single gateway to serve a large number of devices.

A spread spectrum technique known as Chirp Spread Spectrum (CSS) modulation encrypts data using wide linear frequency modulation. A chirp is a sinusoidal signal that fluctuates in frequency over time (sometimes with a polynomial expression for the relationship between frequency and time). LoRa employs CSS because it offers resistance to interference and is simple to demodulate in noisy environments [10].

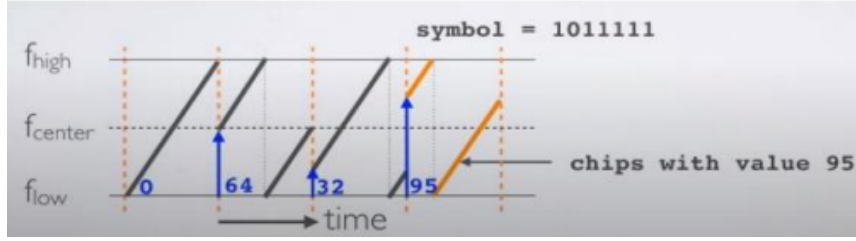


Figure 2: CSS modulation [11]

LoRa's Adaptive Data Rate (ADR) is a mechanism that adjusts the data rate according to the network conditions, like RSSI and SNR of each node. It optimizes power consumption and enhances the network capacity, dynamically optimizing the data rate of the network's nodes.

Spreading Factor (SF) is a critical parameter in LoRa, governing the trade-off between communication range and data rate. It refers to the process where each bit of information is spread across multiple chips. There are a total of 6 spreading factors, from SF7 to SF12, chosen automatically based on various parameters such as distance, Signal to Noise Ratio (SNR) or Received Signal Strength Indicator (RSSI) [10].

In LoRa technology, the spreading factor can be calculated using the formula:

$$SF = \log_2\left(\frac{R}{R_s}\right)$$

where,

- SF is the spreading factor,
- R is the bit rate,
- R_s is the symbol rate.

The symbol rate (R_s) and bit rate (R) are related by the spreading factor according to the formula:

$$R_s = \frac{R}{2^{SF}}$$

The symbol duration (T_s), which is the time taken to send a symbol, is given by the formula:

$$T_s = \frac{1}{R_s}$$

For instance, if we have a Spreading Factor (SF) of 7, on one side we would have symbols formed by 7 bits, and on the other side, 128 possible chips, as 2^7 equals 128.

The SF, along with the bandwidth, is also used to find the symbol rate (R_s) and the symbol time (T_s).

The table below provides a comparison of SF7 to SF10, detailing the variation in range, data rate, and sensitivity:

SF	Range	Data Rate	Sensitivity
SF7	3 km urban, 5 km suburban	5.47 kbps	-123 dBm
SF8	5 km urban, 8 km suburban	3.125 kbps	-126 dBm
SF9	10 km urban, 15 km suburban	1.76 kbps	-129 dBm
SF10	15 km urban, 20 km suburban	0.98 kbps	-132 dBm

Table 1: Comparison of SF7 to SF10 [12]

As we move from SF7 to SF10, we can observe an increase in range and sensitivity at the cost of a reduced data rate.

2.2.2 Sigfox

Sigfox is another LPWAN technology created specifically for IoT applications that require long-range communication with low power consumption. It provides connectivity for a wide range of devices and is known for its simplicity and cost-effectiveness [13].

Sigfox utilizes two Ultra Narrow Band (UNB) modulations for data transmission, namely Differential Binary Phase Shift Keying (DBPSK) in uplink communication and Gaussian Frequency-Shift Keying (GFSK) in downlink. By employing a narrow bandwidth of 100 Hz, the maximum achievable data rates are limited to 600 bps for downlink and 100 bps for uplink transmissions [13].

The characteristics described above enable Sigfox to achieve coverage of approximately 30 to 50 km in rural environments, while in urban areas, the coverage varies between 3 and 10 km. Sigfox operates in different frequency bands depending on the region. In Europe, it operates in the 868 MHz band, in the United States at 902 MHz, and in Asia at 433 MHz [13].

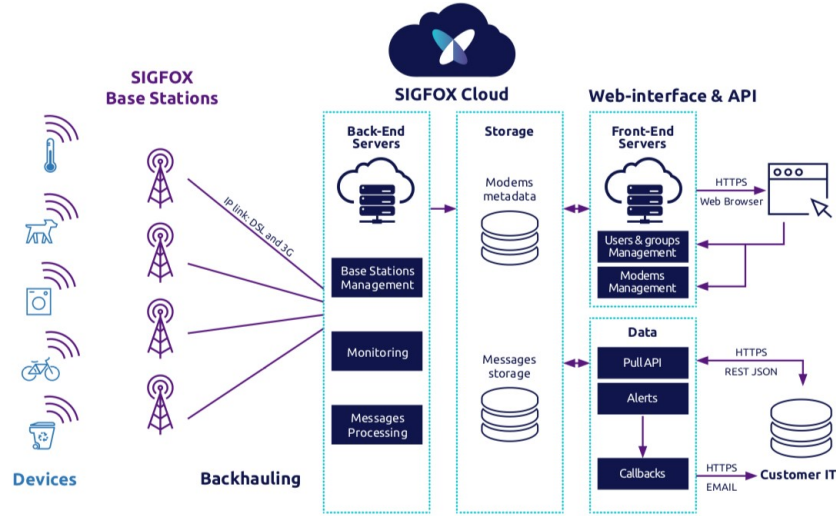


Figure 3: Sigfox Architecture [14]

Sigfox operates on a star network topology, where end devices communicate with a central base station known as the Sigfox Access Station (SAS). This architecture facilitates efficient network management for large-scale IoT deployments, ensuring simplicity, scalability, and centralized data processing and storage in the Sigfox cloud platform [15].

Authentication on the Sigfox network is achieved through the use of unique device identifiers and cryptographic keys. During the registration process, each device is assigned a 32-bit identifier called the Sigfox Device ID (DevID) and a corresponding 128-bit cryptographic key known as the Sigfox Secret Key (Secret Key). These keys play a crucial role in establishing secure communication channels between the devices and the Sigfox cloud, ensuring the integrity and confidentiality of the transmitted data.

Sigfox provides extensive coverage through its global network of base stations, enabling connectivity over long distances, typically spanning tens of kilometers. However, it is important to note that Sigfox operates at a low data rate, allowing only small amounts of data to be transmitted per device. Typically, this translates to a maximum of 140 messages, each containing 12 bytes of data, per day [16].

2.2.3 NB-IoT

NB-IoT (Narrowband Internet of Things), also known as LTE Cat-NB1, is a cellular-based LPWA (Low-Power Wide Area) technology introduced by the 3rd Generation Partnership Project (3GPP) in release 13 in 2016. It emerged as an improvement over LTE-M and operates on 4G/LTE cellular networks. Unlike LTE-M, which operates solely in LTE bands, NB-IoT can be deployed in different parts of the GSM (2G) and LTE (4G) spectrums. The three main spectrum zones for NB-IoT implementation are [16]:

- **Standalone:** NB-IoT can reuse GSM frequency bands.
- **Guard band:** NB-IoT can operate in the guard bands of LTE. These spectrum zones are not used to prevent interference between closely spaced transmissions.
- **In-band:** NB-IoT can share frequency bands with LTE.

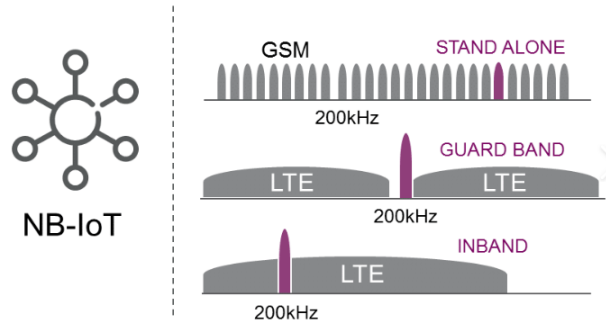


Figure 4: NB-IoT frequency bands [17]

In terms of technical characteristics, NB-IoT utilizes a bandwidth of 200 kHz when operating in GSM bands, while in LTE bands, the bandwidth is 180 kHz. Its transmission speed is lower compared to LTE-M, with a downlink speed of 250 kbps and an uplink speed of 20 kbps. The latency is around 1 second. These characteristics enable devices to extend their battery life up to 10 years and achieve a coverage range exceeding 15 km under certain circumstances. However, it is not suitable for applications requiring mobility or voice transmission. NB-IoT only supports Frequency Division Duplexing (FDD) mode, but similar to LTE-M, it is compatible with power-saving modes such as eDRX (extended Discontinuous Reception) and PSM (Power Saving Mode), which allow devices to conserve energy and prolong battery life up to 10 years. The maximum transmission power is 23 dBm [18].

The NB-IoT downlink utilizes Orthogonal Frequency-Division Multiple Access (OFDMA) and Quadrature Phase Shift Keying (QPSK), while the uplink employs single-carrier Frequency-Division Multiple Access (FDMA) modulation. The maximum payload size for NB-IoT messages is 1600 bytes. The downlink is limited to 200 kbps, while the uplink is restricted to 20 kbps. [18].

Smart metering and asset tracking in urban settings are two examples of applications that can benefit from NB-IoT's wide coverage, low latency, and high dependability. It can support up to 10,000 devices per base station, making it ideal for large-scale deployments [18].

Features	LoRa	SigFox	NB-IoT
Modulation	CSS	DL: GFSK UL: DBPSK	DL: QPSK + OFDMA UL: FDMA
Frequency	868 MHz (Europe) 915 MHz (North America) 433 MHz (Asia)	862 to 928 MHz	LTE frequency bands (700, 800, 900, 1800, 2100, 2600 MHz)
Bidirectional	Full duplex, half duplex	Half duplex	Half duplex
Bandwidth	125 kHz	100 Hz	200 kHz
Max. Data Rate	50 kbps	100 bps	250 kbps
Max. payload (length)	243 bytes	UL: 12 bytes DL: 8 bytes	1600 bytes
Max. messages per day	Unlimited	UL: 140 DL: 4	Unlimited
Range	10 km in rural areas 5 km in urban areas	30 - 50 km in rural areas 3 - 10 km in urban areas	25 km
Interference immunity	Very high	Very high	Low
Adaptive Data Rate	Allowed	Not allowed	Not defined

Table 2: Comparison of LoRa, SigFox, and NB-IoT Technologies [19]

2.3 Alternative IoT Communication Technologies

The exponential growth of IoT in the agricultural sector has prompted the exploration of diverse communication technologies to cater to the specific requirements of different applications. While LoRaWAN has gained considerable popularity in the realm of smart agriculture, it is essential to investigate alternative IoT communication technologies that can offer supplementary advantages or address specific limitations inherent in LoRaWAN. In this section, we delve into two distinct technologies: IoT satellite communication and IoT UAV swarm systems.

2.3.1 IoT Satellite Communication

Connecting organizations in remote locations to IoT solutions has been a challenge due to inadequate coverage and cost-related constraints. A viable solution that has recently emerged is satellite-based IoT connectivity, which offers low-cost, low-power connectivity directly from orbit. In contrast to terrestrial networks, which only cover around 20% of the Earth's surface, satellite networks offer almost global coverage. [20].

The adoption of LEO-based satellite IoT networks has witnessed remarkable growth, with the number of subscribers exceeding 5 million by 2021. Projections indicate that these networks are expected to capture nearly 20% of the global market share by 2026. Several key developments drive this growth [20]:

- **Expansion of LEO-based satellite IoT networks:** LEO constellations enable faster and more cost-effective design and deployment, making them well-suited for low-power communications. The LEO-based satellite IoT connection market is projected to expand at a 25% CAGR between 2022 and 2026.

- **Adoption of hybrid connectivity:** There is a growing trend among IoT network operators to collaborate and offer hybrid connectivity solutions, allowing IoT devices to rely primarily on terrestrial networks but seamlessly switch to satellite connectivity in situations where terrestrial coverage is not available.
- **Entry of tech giants into the LEO-based broadband satellite operator market:** Companies like SpaceX and Amazon are establishing LEO constellations for broadband internet connectivity. Although not exclusively targeting the IoT market, they have the potential to serve customers with diverse bandwidth requirements and emerge as significant competitors in the satellite backhauling for IoT market.

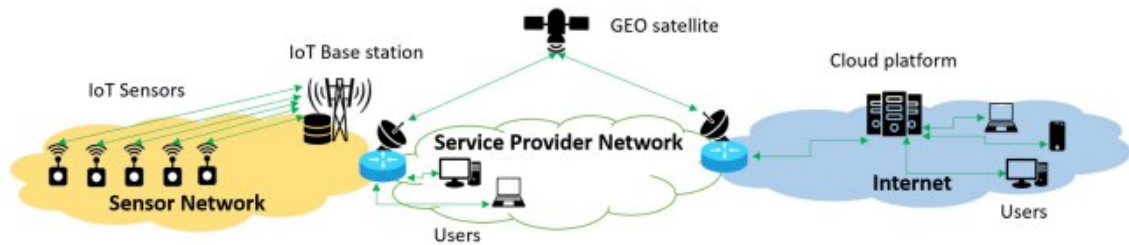


Figure 5: IoT satellite network [21]

Direct-access (fronthaul) and backhaul are the two main functions of satellites in a satellite-terrestrial integrated network for Internet of Things applications. [21, 22]:

- **Direct-access scenarios:** Sensors establish a direct connection with satellites, which serve as the initial link in the communication path between IoT devices and users.
- **Backhaul scenarios:** It involves the utilization of satellites to transmit sensors' data to a dedicated service center or the internet through a satellite data network.

Challenges

The utilization of LEO satellites for IoT communication presents some challenges that must be overcome.

Firstly, the high Doppler shift resulting from the rapid movement of LEO satellites around the Earth introduces a significant carrier frequency offset, making it challenging for the receiver to accurately estimate the channel. To mitigate this issue, advanced signal processing techniques and adaptive algorithms can be employed to track and compensate for the Doppler shift. Additionally, the use of multiple antennas and advanced modulation schemes can help minimize the impact of the Doppler effect on the communication link [23].

Secondly, meeting the data rate requirements is crucial. IoT connectivity scenarios often demand data rates of 10 kbps or higher. However, the longer round trip time (RTT) in satellite networks compared to terrestrial networks needs to be taken into account when calculating the achievable data rate. Efficient error correction codes and optimization techniques can be implemented to ensure reliable data transmission even at lower data rates. Moreover, data compression and prioritization methods can be employed to transmit critical information first, optimizing the utilization of the available data rate [23].

Another significant challenge is the limited power resources of IoT devices. Designing communication protocols for satellite IoT networks must consider the energy constraints of these devices. Energy-efficient communication techniques, such as duty cycling, can be utilized to minimize power consumption and extend the battery life of IoT devices. Additionally, network management strategies that facilitate adaptive power control and resource allocation can optimize the trade-off between communication performance and energy efficiency [23].

LR-FHSS

LR-FHSS is an innovative physical layer specifically designed to overcome the network capacity and robustness limitations of LoRaWAN in densely deployed networks. By implementing fast frequency hopping techniques for uplink communication, LR-FHSS significantly enhances capacity, sensitivity, and interference rejection compared to traditional modulation schemes that rely on a single channel. Its flexibility also makes LR-FHSS ideal for applications that require differentiated service levels, such as satellite communications [24].

Integrating LR-FHSS into IoT satellite communication brings numerous benefits. These include improved network capacity and per-device range, enhanced interference robustness, compliance with 1W devices in the US, and support for larger frequency drifts and Doppler shifts. By leveraging LEO satellites in conjunction with LR-FHSS, the overall communication system achieves low-latency communication and enables global coverage, particularly in remote areas with limited infrastructure [24].

However, IoT satellite communication does face certain challenges that necessitate attention. Mitigating interference, efficiently handling backend traffic, addressing end-node complexity, and developing power-saving algorithms are crucial areas of focus. Instead of propagation-related losses, packet losses might occur owing to interference with the same spreading factor (SF) and the channel. Additionally, limited bandwidth availability and the influence of shadowing, multipath fading, and Doppler shifts on communication performance pose additional hurdles [25].

To address these challenges, researchers have proposed the utilization of network coding and device-to-device (D2D) transmission schemes to enhance LR-FHSS in direct-to-satellite IoT networks. D2D communication enables devices to directly exchange data with each other, bypassing the need for a central gateway. This approach reduces the burden on the network infrastructure and improves overall network capacity.

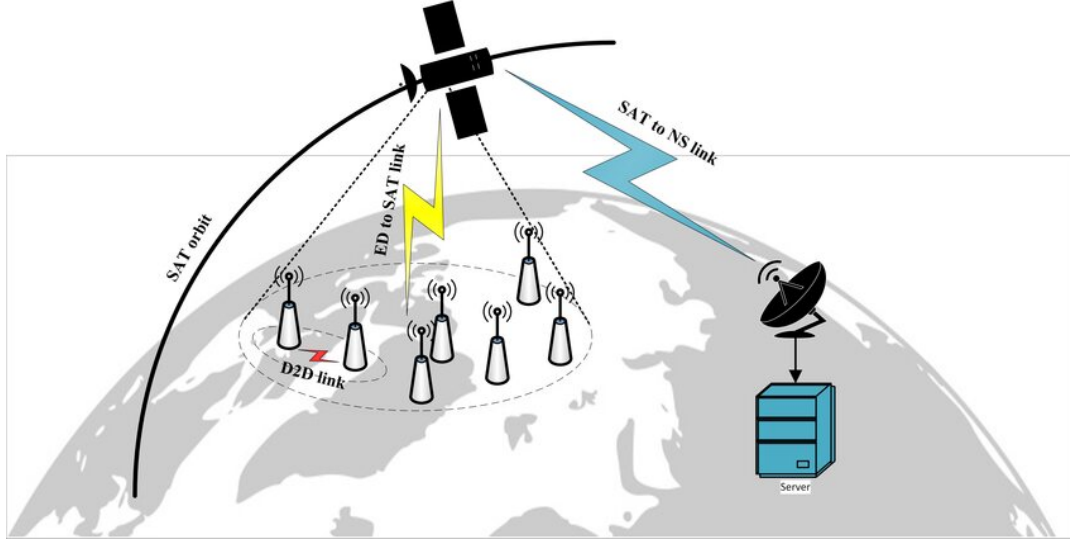


Figure 6: Enhancing LR-FHSS through Device-to-Device (D2D) [25]

Network coding is employed to combine multiple packets into a single transmission, reducing the total number of required transmissions [26].

The integration of D2D and network coding with LR-FHSS has demonstrated significant improvements in network capacity, albeit with an increase in the number of transmissions per IoT end device. In a study by Maleki et al. [26], the proposed D2D-aided LR-FHSS scheme exhibited remarkable enhancements in network capacity. Specifically, at a typical outage of 10^{-2} for DR6 and DR5, the scheme showcased a network capacity increase of up to 249.9% and 150.1%, respectively, compared to the standalone LR-FHSS scheme.

2.3.2 IoT UAV Swarm System

The integration of Unmanned Aerial Vehicle (UAV) technologies and the Internet of Things (IoT) has given rise to innovative intelligent farming systems that enhance agricultural productivity while reducing the environmental impact of traditional farming practices. UAVs, commonly known as drones, are used in various agricultural tasks such as plant monitoring, water and pesticide application, eradication, mapping, seed planting, fertility assessment, and weed detection. Inspired by ants and bees, swarm intelligence has enabled the development of adaptive, scalable and resilient algorithms specifically designed for network routing in wireless sensor networks (WSN). Advancements in swarm technology and mission-oriented control have facilitated the collaboration of groups of drones equipped with 3D cameras and diverse sensors, offering comprehensive land management solutions for farmers [27].

Agricultural UAVs enhance stability, productivity, and measurement precision. They are equipped with sensors that are capable of monitoring crop quantity and quality, field conditions, and climatic variables including temperature, humidity, wind

speed, and wind direction. However, challenges still exist, including identifying the optimal mobility patterns for specific services and developing a standardized methodology for evaluating performance using quality metrics [28].

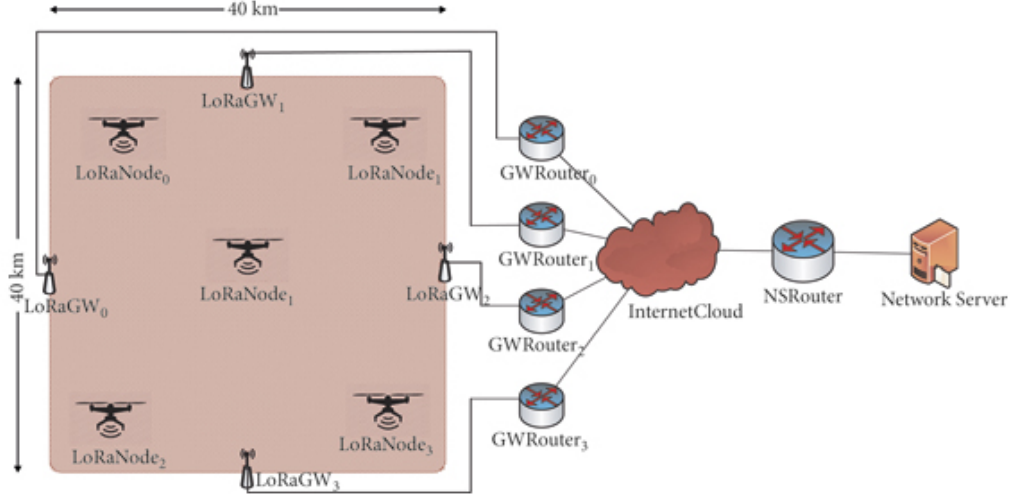


Figure 7: Integration of UAVs with LoRaWAN [28]

The application of LoRa technology in Flying Ad Hoc Networks (FANETs), as well as the challenges involved, have been thoroughly studied. FANETs are UAV-based wireless networks, and LoRa technology is a suitable option for IoT applications because of its scalability, low power requirements, and simplicity of implementation. The authors also looked at different FANET communication topologies, from single UAV architectures to multilayer systems including layers for clustering groups of UAVs. [29].

Despite the advancements made in UAV swarm systems, obstacles such as limited flight time, battery efficiency, payload capacity and communication range demand further research. Connectivity limitations are effectively managed through meshed LoRaWAN gateways and satellite communication systems in remote regions. Future research endeavours should concentrate on overcoming connectivity constraints in remote areas and investigating novel communication technologies that foster sustainable agriculture [30].

2.4 IoT communication protocols

In the current landscape, numerous sensors and IoT devices rely on proprietary communication methods, posing challenges for scalability and standardization in IoT applications. To address this issue, various technology companies have developed open-source protocols aimed at achieving compatibility across a wide range of IoT devices. This section examines the most commonly used IoT communication protocols, their characteristics, and key differences.

2.4.1 Message Queuing Telemetry Transport (MQTT) protocol

MQTT (Message Queuing Telemetry Transport) is a messaging protocol based on the publish-subscribe (PubSub) model, originally developed by IBM in 1999 for machine-to-machine (M2M) communications. While initially proprietary, the source code of MQTT was released in 2010, enabling broader adoption. Under the OASIS standard, MQTT was later upgraded to version 3.1 in 2013, incorporating several enhancements to its functionality. Currently, MQTT stands as one of the most widely used protocols in diverse IoT projects and applications [31].

MQTT relies on the TCP/IP protocol, which offers a connection-oriented approach for robust and reliable data transmission, distinguishing it from CoAP. The architectural design of MQTT follows a star topology, as depicted in the following figure:

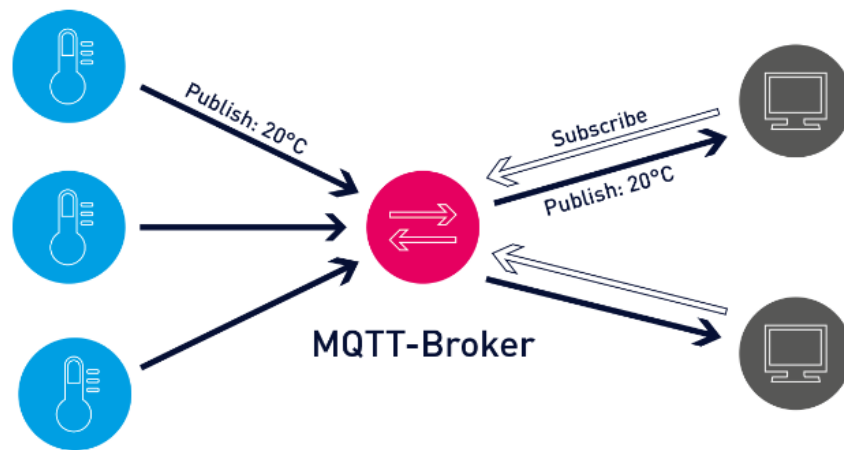


Figure 8: MQTT architecture [32]

In an IoT communication system utilizing the MQTT protocol, various components are involved, as highlighted in the following list [31, 32]:

- **Client:** Nodes that connect to a broker to send or receive messages.
- **Broker:** A server that manages all the messages sent by clients.
- **Topic:** Refers to the subject of the message, acting as a filter to differentiate messages based on the information they contain.
- **Payload:** Represents the content of the message.

MQTT Protocol Functioning

In the initial stage, the sender client initiates a TCP/IP connection with the broker, utilizing the server's associated URL or IP address, along with the designated port. The standard communication ports employed are 1883 for unencrypted connections

and 8883 for connections implementing the SSL/TLS security protocol. The client's first communication entails sending a message referred to as `CONNECT`, containing crucial details such as the client's ID, username, and password [33]. The broker, in response, will either accept or reject the connection by transmitting a message known as `CONNACK`.

Once the connection is successfully established between both parties, the sender client proceeds to use `PUBLISH` packets to convey information to the broker [33]. These packets encapsulate the topic of the message and its corresponding payload.

Topics are structured based on a hierarchy or levels separated by a forward slash ("/").

An example of a topic hierarchy would be:

- Hotel/Room/Wardrobe
- Hotel/Kitchen/Refrigerator
- Hotel/Kitchen/Oven

In order for one or more clients to receive messages from a specific topic, they have to use `SUBSCRIBE` packets to indicate their desired topic of interest. In MQTT, the use of wildcards provides enhanced topic management capabilities. The following types of wildcards can be employed [33]:

- **Single level wildcard:** Represented by the "+" character, this wildcard allows clients to receive messages from a repeating subtopic at a specific level across different topics. For instance, if a client wishes to receive all "Color" topics, they would subscribe to the topic "Garage/+/Color". Similarly, if they want to receive all topics at a specific level, they would subscribe to "Garage/Car/+".
- **Multi-level wildcard:** Represented by the "#" character, this wildcard enables clients to subscribe to all subtopics from a specific level onwards. For example, if a client wants to receive all payloads with topics beginning with "Building/Floor", they would subscribe to "Building/Floor/#". To receive all messages sent to the MQTT broker, the client simply subscribes to the topic "#".

To stop receiving messages from a specific topic, the `UNSUBSCRIBE` packet is utilized. The broker responds to the `SUBSCRIBE` and `UNSUBSCRIBE` actions using `SUBACK` and `UNSUBACK` packets, respectively, indicating whether the client was successfully subscribed to or unsubscribed from the desired topic. Clients periodically send `PINGREQ` messages to check the status of their connection, and the broker responds with a `PINGRESP` packet. Finally, when a client intends to disconnect from the broker, they send a `DISCONNECT` message [33].

Quality of Service (QoS)

In contrast to other IoT protocols, MQTT stands out for its distinctive Quality of Service (QoS) feature. This attribute enables users to select from three distinct service levels, each offering specific characteristics in terms of message transmission latency and robustness. The following describes the features of each level [34, 35]:

- **QoS 0 (At most once):** This level provides the simplest and lightest approach, where the message is sent only once. In case of any errors, the message is lost, and the receiver does not acknowledge receipt. This level is suitable for transmitting non-critical information.
- **QoS 1 (At least once):** Messages at this level are sent at least once, and the receiver acknowledges receipt with a PUBACK packet. Duplicate messages may be received, but they do not pose a problem. This level is suitable for situations where duplicate messages are not a concern.
- **QoS 2 (Exactly once):** This level ensures that the message is delivered only once. It employs two pairs, namely PUBLISH/PUBREC and PUBREL/PUBCOMP, to guarantee exactly-once delivery. This level should be used in critical situations where both duplicate messages and message loss are unacceptable.

All messages sent using QoS 1 or QoS 2 are queued until a client with a persistent session enabled receives them. The MQTT protocol offers several key features, including its lightweight nature designed for resource-constrained devices, efficient and energy-conscious operation, scalability for easy addition of new devices to the broker, secure information transmission using SSL/TLS protocols, and support for asynchronous communication [34, 36].

- Lightweight, designed for devices with limited resources and low-bandwidth message transmission.
- Fast, efficient, and energy-efficient.
- Scalable, allowing for easy addition of new devices to the broker.
- Secure information transmission through SSL/TLS protocols.
- Asynchronous communication.

2.4.2 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is an open-source web-based protocol standardized by the Internet Engineering Task Force (IETF) in 2010, following RFC 7252. It is specifically designed for machine-to-machine (M2M) communication, catering to devices and sensors with limited resources and minimal processing

power. Unlike MQTT, CoAP adopts a decentralized node-to-node (client-server) communication model, eliminating the need for a central node (broker) to manage messages [37].

CoAP follows the RESTful client/server architecture, similar to HTTP, but with the utilization of the User Datagram Protocol (UDP) instead of the Transmission Control Protocol (TCP) for communication. This enables CoAP to achieve faster message transmission, three times quicker than HTTP, due to its connectionless nature. Depending on the message type, acknowledgments may not be necessary [38]. Figure 10 illustrates the functioning of both protocols.

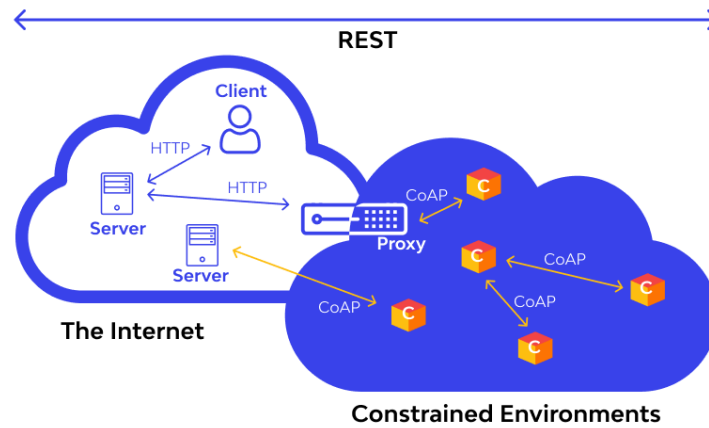


Figure 9: COAP architecture [39]

The previous figure depicts the structures of HTTP and CoAP, consisting of the following elements [38]:

- **Endpoints:** Representing nodes involved in sending or receiving messages.
- **Sender:** The node responsible for transmitting a message.
- **Recipient:** The node designated to receive the message.
- **Server:** The component processing client requests and delivering responses.
- **Client:** The node initiating requests and receiving responses from the server.

CoAP supports four message classifications [39]:

1. Confirmable message (CON).
2. Non-confirmable message (NON).
3. Acknowledgment (ACK).
4. Reset (RST).

For Confirmable messages (CON), the sender ensures message reception by the server, which responds with an acknowledgment (ACK) message. If the server fails to receive the request correctly, it sends a reset (RST) message. Conversely, with Non-confirmable messages (NON), the server does not need to reply with an ACK upon receiving the request.

In terms of security, CoAP employs the Datagram Transport Layer Security (DTLS) protocol to encrypt messages during transmission. DTLS, derived from TLS, is used in conjunction with the UDP transport protocol. Encryption systems compatible with CoAP include ECC, RSA, and AES [39].

The key attributes of the CoAP protocol are [39]:

- Lightweight protocol facilitating message exchange among resource-constrained IoT devices.
- Asynchronous communication facilitated by the UDP transport protocol.
- Based on HTTP.
- Minimized header size to reduce message overhead.
- Suitable for applications leveraging web services.
- Optimal for multicast and broadcast operations.

2.4.3 Advanced Message Queuing Protocol (AMQP)

AMQP (Advanced Message Queuing Protocol) is an open standard application layer protocol developed by John O'Hara at JPMorgan Chase in London, UK, in 2003. It was designed for message-oriented enterprise applications, with a focus on reliability, security, provisioning, and interoperability. This simple M2M protocol includes features like transactions, topic-based publish-subscribe messaging, strong queuing, and dynamic routing. It supports both publish/subscribe and request/response topologies. [40].

An "exchange" with a specific name is created in an AMQP communication system by either the publisher or the consumer and is then used to start a conversation between the two parties. Next, the consumer creates a "queue" and "binds" it to the exchange, enabling messages received by the exchange to be forwarded to the correct queue via the "binding" procedure. [41].

The maximum message payload size for AMQP is set by the broker/server or the programming language, and it typically consists of an 8-byte fixed header. The protocol operates over TCP as its default transport protocol and utilizes TLS/SSL and SASL for security. The communication between client and broker is connection-oriented, and AMQP offers two levels of Quality of Service (QoS) for message delivery: Unsettled (not reliable) and Settled (reliable). AMQP also guarantees at-most-once, at-least-once, and exactly-once message delivery [41].

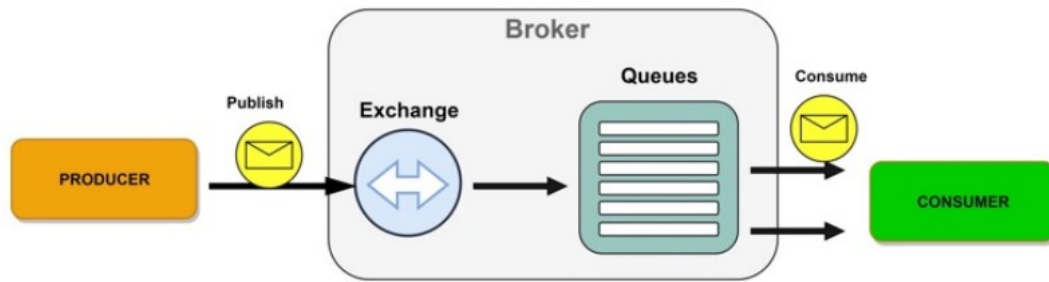


Figure 10: COAP architecture [42]

The previous figure illustrates the three components involved in the AMQP protocol [43]:

- **Broker (Intermediary):** The broker consists of the Exchange and the Queue, responsible for managing the sending and receiving of messages, as well as queuing them.
- **Producer or Publisher:** This component acts as the sender, composing and sending messages.
- **Consumer:** The consumer refers to any application that receives messages from the queue. Each consumer is assigned a specific queue.

When a producer composes messages, they are sent to the exchange. The exchange then routes the messages to the appropriate queues, where they are queued before being forwarded to the consumer. The exchange determines the most suitable queue for each message based on bindings, which are rules defined by the producer. The routing performed by the broker is determined by the type of exchange being used [43].

The model layer and the transport layer are the two layers that form AMQP. The message queue service model as well as other instructions that uniformly control its behaviour are defined at the model layer. The commands are transferred between the application client and the broker through the transport layer. It manages channel multiplexing, framing, data representation, and error processing while returning the results.

AMQP includes four types of exchanges: Direct, Fanout, Topic, and Headers [43]:

1. **Direct Exchange:** This exchange type routes messages based on the routing key sent by the message's producer to the correct message queue.
2. **Fanout Exchange:** Messages received by this exchange type are sent to all message queues bound to it, without any further judgment during message processing.

Criteria	MQTT	CoAP	AMQP
Year	1999	2010	2003
Architecture	Client - Broker	Client - Server Client - Broker	Client - Server Client - Broker
Abstraction	Publish - Subscribe	Request - Response Publish - Subscribe	Request - Response Publish - Subscribe
QoS	QoS 0, QoS 1, QoS 2	Confirmable message Non-confirmable message	Settle Format Unsettle Format
Transport Protocol	TCP	UDP	TCP
Security	TLS / SSL	DTLS	TLS / SSL
Default Port	1883 / 8883	5683 / 5684	5671 / 5672

Table 3: Comparison of IoT Protocols: MQTT, CoAP, and AMQP [42]

3. **Topic Exchange:** Messages are routed to queues based on matching routing keys defined by bindings.
4. **Headers Exchange:** Unlike other exchanges, the headers exchange routes messages based on message header properties, disregarding routing keys. For message-receiving clients, this exchange type offers both unique and shared message distribution techniques.

Although AMQP can be implemented in IoT projects, it is not inherently designed for such environments. Its usage often requires high-performance devices to ensure reliable message delivery, which can be challenging for IoT devices with limited resources.

2.5 Network Servers

In this section, we will discuss the network servers used in our project, specifically focusing on ChirpStack and The Things Network. Both of these platforms play an essential role in enabling the communication between IoT devices and the cloud-based Microsoft Azure platform.

2.5.1 ChirpStack

ChirpStack is a comprehensive LoRaWAN network server solution designed for both private and public network deployments. It provides a modular architecture, allowing users to build customized solutions by integrating its different components as required [44]. In this section, we will delve deeper into the architecture and features of ChirpStack.

ChirpStack Architecture

ChirpStack's architecture is composed of three primary components, each playing a crucial role in the network's operation [45]:

- **ChirpStack Gateway Bridge:** Installed on the LoRaWAN gateway or a nearby server, the Gateway Bridge is responsible for the communication between the gateway and the ChirpStack Network Server. It translates the gateway-specific protocol into a generic format that can be understood by the ChirpStack Network Server. This enables seamless integration of different gateway brands and models.
- **ChirpStack Network Server:** The Network Server is responsible for the core LoRaWAN network management functions. It processes uplink frames from the gateways, manages the downlink frames, and takes care of MAC commands, ADR, and multicast group management. Additionally, the Network Server ensures that the communication between devices and the network is secure and efficient.
- **ChirpStack Application Server:** The Application Server handles the processing and storage of application payloads. It decodes the payload and makes it available to external applications and services via integration mechanisms such as MQTT, HTTP webhooks, or gRPC. This allows users to easily integrate ChirpStack with their existing software infrastructure.

Key Features of ChirpStack

ChirpStack offers a range of features that make it a reliable and flexible solution for managing LoRaWAN networks [46]:

- **Scalability:** ChirpStack's architecture enables horizontal scaling, allowing it to handle growing network demands efficiently.
- **Security:** The platform ensures that all communication between devices, gateways, and servers is encrypted and secure.
- **Interoperability:** ChirpStack adheres to the LoRaWAN specifications, ensuring compatibility with a wide range of LoRaWAN devices and gateways.
- **Customizable:** The open-source nature of ChirpStack allows users to customize and extend the platform to fit their specific requirements.
- **Integration:** ChirpStack offers multiple integration options, making it easy to connect with external applications, databases, and services.

By leveraging ChirpStack's architecture and features in our smart agriculture system, we can build a robust, secure, and scalable LoRaWAN network that can adapt to changing needs and requirements.

2.5.2 The Things Network

The Things Network (TTN) is a global, community-driven LoRaWAN network that empowers users to build and manage their own IoT solutions using the LoRaWAN protocol [47]. This section will provide a comprehensive overview of The Things Network, focusing on its structure and the features it offers.

Structure of The Things Network

TTN's structure is built around several key elements that work together to manage and maintain the network:

- **The Things Gateway:** This is the hardware that receives and transmits data between the IoT devices and the network. It can be a commercially available LoRaWAN gateway or a custom-built one.
- **The Things Network Stack:** The software component manages the LoRaWAN network, including handling data uplinks and downlinks, processing MAC commands, and managing ADR settings. It ensures that the communication between devices and the network is secure and efficient.
- **The Things Network Console:** A web-based user interface that enables users to manage their devices, applications, and gateways. It provides a visual representation of the network, allowing users to monitor their devices' performance and troubleshoot issues.

These key elements provide the backbone for The Things Network and enable its versatile functionality [47].

Features of The Things Network

The Things Network offers a variety of features that make it an ideal choice for deploying and managing LoRaWAN networks:

- **Global Coverage:** TTN provides a worldwide network of gateways, allowing users to deploy their devices anywhere with LoRaWAN coverage.
- **Open-Source:** The platform is open-source, enabling users to customize and extend it according to their needs.
- **Community Support:** TTN boasts a large and active community that contributes to the platform's development and provides support for users.
- **Integration:** The Things Network offers multiple integration options, making it easy to connect with external applications, databases, and services.

- **Security:** The platform ensures that all communication between devices, gateways, and servers is encrypted and secure.

These features are among the key reasons why The Things Network is a preferred choice for IoT solutions [47].

By integrating The Things Network into our smart agriculture system, we can leverage the power of a global, open-source, and community-driven LoRaWAN network to create a scalable, secure, and efficient solution that meets the needs of modern agriculture.

2.5.3 Comparing ChirpStack and The Things Network

Both ChirpStack and The Things Network offer robust solutions for deploying and managing LoRaWAN networks. However, there are some key differences between the two platforms that may influence the choice for a smart agriculture system based on Microsoft Azure. In this section, we will objectively compare ChirpStack and The Things Network based on their characteristics and features.

Deployment and Management

ChirpStack is designed for private network deployments, providing users with full control over their LoRaWAN network infrastructure. This enables users to build customized solutions tailored to their specific needs. On the other hand, The Things Network is a global network that relies on contributions from a multitude of users, which simplifies the process of deploying and monitoring devices in various locations around the globe. However, the global nature of The Things Network may limit the level of customization and control available to users compared to ChirpStack.

Scalability and Flexibility

Both ChirpStack and The Things Network are designed to scale horizontally, allowing them to handle growing network demands efficiently. However, ChirpStack's modular architecture allows users to integrate and configure the platform to fit their specific requirements, offering greater flexibility compared to The Things Network. This can be particularly beneficial for large-scale deployments that require complete control over the network infrastructure.

Security

Both ChirpStack and The Things Network ensure that all communication between devices, gateways, and servers is encrypted and secure. However, ChirpStack's private network deployment model provides an added layer of security, as users have full

control over the network infrastructure and can implement additional security measures as needed.

Integration with Microsoft Azure

ChirpStack and The Things Network both offer multiple integration options for connecting with external applications, databases, and services. Integration with Microsoft Azure is possible with both platforms by leveraging Azure Functions, Azure IoT Hub, or Azure Event Hubs to create seamless data pipelines between the LoRaWAN network and the cloud-based platform.

Upon comparing ChirpStack and The Things Network, we have identified key differences in terms of deployment, management, scalability, flexibility, and security. ChirpStack's private network deployment model, modular architecture, and added security benefits make it a more suitable choice for a smart agriculture system based on Microsoft Azure. By leveraging ChirpStack's architecture and features, we can build a robust, secure, and scalable LoRaWAN network that can adapt to the changing needs and requirements of modern agriculture.

2.6 Cloud Platform: Microsoft Azure

Cloud platforms have become instrumental in modern IoT-based agriculture systems due to the need for efficient, scalable, and data-driven solutions. They provide the infrastructure, resources, and services necessary for managing and analyzing the massive volumes of data produced by IoT devices in agricultural applications [48]. This section overviews why integrating them is advantageous, particularly focusing on the choice of Microsoft Azure for this project and its role in the proposed LoRaWAN smart agriculture system.

The integration of cloud platforms in IoT systems presents several benefits [48, 49, 50]:

- **Scalability:** As the numbers of IoT devices escalate, the volume of data generated in the agricultural field follows suit. Cloud platforms are designed to solve this issue, offering seamless adjustment of resources to cater to growing computational and storage demands. Such scalability ensures the system's performance remains intact, even as demands multiply.
- **Data Management:** Cloud platforms provide a centralized system for storing, managing, and analyzing data. This unified approach aids in transforming raw data into actionable insights, supporting informed decision-making. It also promotes integrated data usage, helping avoid data silos and fostering data-driven agriculture.

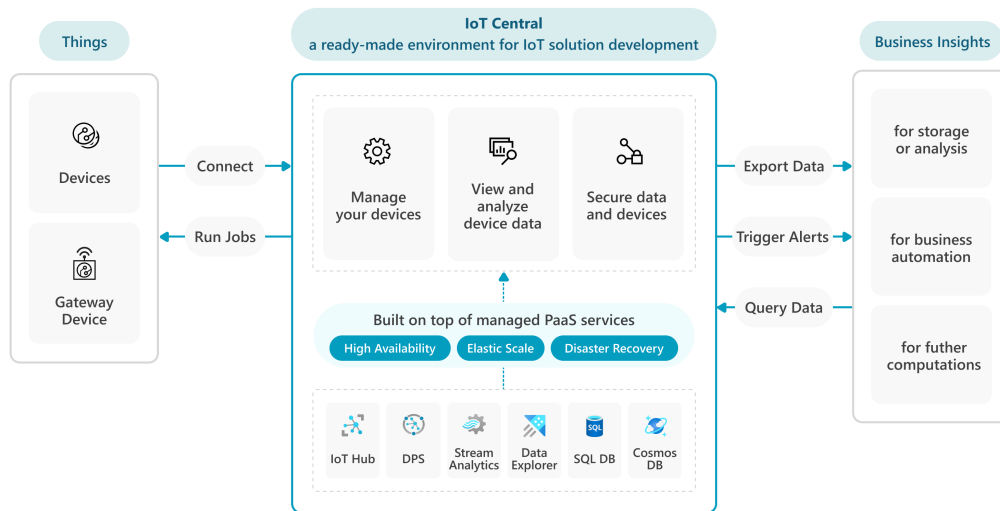


Figure 11: Microsoft Azure Architecture [51]

- **Remote Operations:** Cloud platforms enable remote management of agricultural systems. This capability enhances operational efficiency and can lead to cost reductions. Remote management allows for prompt responses to arising issues, improving agricultural resilience.
- **Data Security:** Amidst mounting cyber threats, cloud platforms present robust security protocols. They employ cryptographic measures, secure access controls, and regular security patches to safeguard sensitive agricultural data, ensuring data integrity and regulatory compliance.
- **Integration Ease:** The incorporation of cloud platforms in IoT systems simplifies the interconnection of devices, services, and APIs. By facilitating seamless interaction amongst different system components, they foster a conducive environment for IoT-driven agriculture. This ease of integration unlocks the potential for advanced, customized solutions that can address specific agricultural challenges.

2.6.1 Leveraging Microsoft Azure in a LoRaWAN Smart Agriculture System

Microsoft Azure has been selected as the cornerstone for this project leveraging its resources to establish the network server, set up the LoRaWAN simulator, and build the IoT dashboard. Azure's broad spectrum of IoT-focused services, tools, and resources drove our selection [51].

Virtual machines afford us the versatility to accommodate various operating systems and software, allowing us to tailor the environment to the needs of the LoRaWAN-centric agriculture system. The scalability of these virtual machines

ensures the project's infrastructure is flexible enough to manage expanding data processing and storage requirements.

Azure IoT Central, a fully-managed IoT application platform, simplifies the construction and deployment of IoT dashboards [52]. Its intuitive interface enables real-time monitoring, management, and analysis of data from IoT devices. The application of Azure IoT Central to our LoRaWAN-based agriculture system allows us to visualize system performance, detect trends or irregularities, and make data-informed decisions to enhance agricultural operations.

3 System Design and Implementation

3.1 Description of the Proposed System & Architecture

The proposed LoRaWAN smart agriculture system aims to enhance agricultural efficiency and productivity by utilizing IoT devices and Microsoft Azure services. The system comprises various components that collaborate to collect, transmit, process, and analyze data from the agricultural environment. The architecture of the proposed system can be divided into five main components:

1. Sensor End Device
2. Gateway
3. Network Server
4. Application Server

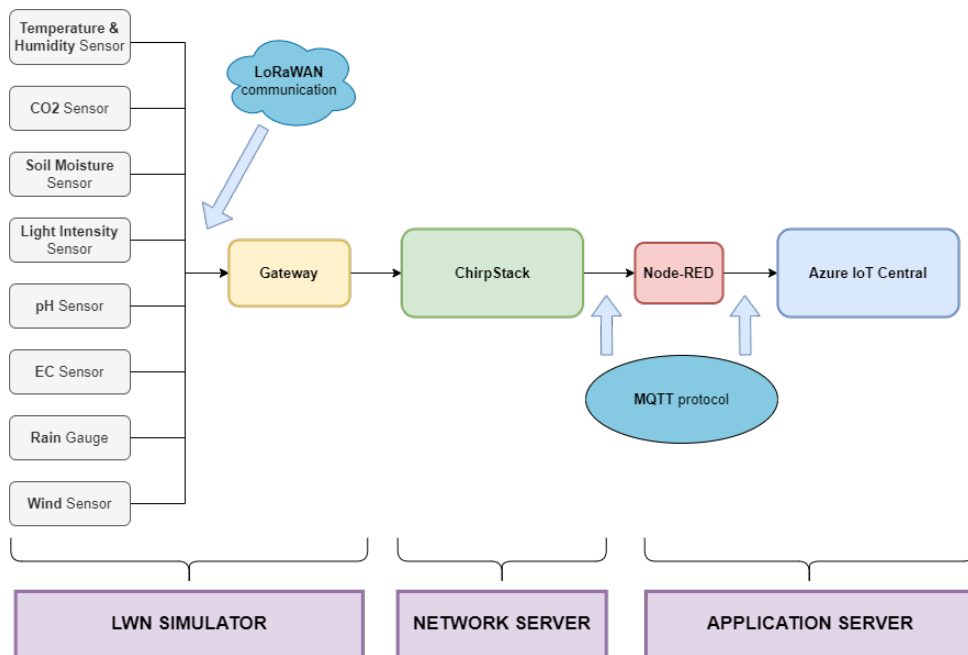


Figure 12: System Architecture

3.1.1 Sensor End Device

The sensor end device comprises intelligent sensor modules strategically placed throughout the field to account for potential microclimates and soil variations. These sensors collect real-time data on numerous parameters crucial to the health and productivity of crops. The sensors and their respective functionalities include [53]:

- **Temperature Sensor:** Monitors the ambient temperature, crucial for predicting plant growth rates and potential pest activity. These sensors measure temperature in degrees Celsius (°C) and typically have a range from -40°C to 125°C.
- **Humidity Sensor:** Measures the humidity level in the air. Humidity directly impacts plant growth and assists in determining the optimal time for irrigation. Humidity is measured in percent relative humidity (% RH), typically ranging from 0 to 100% RH.
- **Soil Moisture Sensor:** Determines the moisture content in the soil, crucial for precise irrigation and efficient water use. Soil moisture is typically measured as volumetric water content, represented as a percentage (%), with typical values ranging from 0% (completely dry) to 50% (saturated).
- **Light Intensity Sensor:** Measures the amount of light reaching the plants. This enables the adjustment of shading systems to optimize plant growth. Light intensity is typically measured in lux (lx), with typical outdoor values ranging from 10,000 to 100,000 lx on a clear day.
- **pH Sensor:** Monitors the pH level in the soil. Soil pH affects nutrient availability and can significantly influence crop yield. pH is a logarithmic scale, typically ranging from 0 (very acidic) to 14 (very alkaline), with 7 being neutral.
- **EC Sensor:** Measures the electrical conductivity of the soil, providing valuable information about nutrient levels and soil salinity. Electrical conductivity is typically measured in deciSiemens per meter (dS/m), with typical values for soils ranging from 0 to 4 dS/m.
- **Rain Gauge:** Records rainfall data, enabling informed decisions regarding irrigation and promoting water conservation. Rainfall is typically measured in millimeters (mm) or inches, with the range depending on local climate conditions.
- **Wind Speed sensor:** Measures wind speed, which can affect plant transpiration rates, pollination, and assist in predicting weather conditions. Wind speed is typically measured in meters per second (m/s) or kilometers per hour (km/h).
- **CO2 Sensor:** Monitors the level of carbon dioxide in the air, essential for photosynthesis. Carbon dioxide concentration is typically measured in parts per million (ppm), with typical outdoor concentrations around 400 ppm.

3.1.2 Gateway

The gateway serves as the bridge between the sensor end devices, actuators, and network servers. It receives data from the end devices and actuators through the LoRaWAN protocol and forwards it to the network servers for processing and analysis.

3.1.3 Network Server

The network servers, hosted on Microsoft Azure, process and analyze the data received from the gateway, providing real-time insights and decision support for farmers. They can also be integrated with other cloud-based services, such as AI algorithms and machine learning models, to provide advanced capabilities for agriculture systems.

3.1.4 Application Server

The Application Server processes the data received from the Network Servers, interpreting it, and executing appropriate actions based on the processed data. It interacts with the Network Servers hosted on Microsoft Azure, receiving data via MQTT protocol, and directs the data to the Azure IoT Central, where it is presented in a user-friendly format.

The proposed system is designed to automatically respond to certain conditions. For example, if the soil moisture sensors detect that the soil is too dry, the system could automatically trigger irrigation. Similarly, if the light intensity sensors detect that the light is too strong, the system could automatically adjust shading systems. This level of automation is made possible by the integration of the sensor end devices, actuators, gateway, network servers, and application server, all working in harmony to optimize agricultural practices.

The key to making this system effective is to ensure that all the sensors are properly calibrated and that the data they collect is accurately interpreted. This will likely involve some trial and error, as well as ongoing adjustments as you learn more about the specific conditions in your field. With the proposed system, farmers can make more informed decisions and optimize their farming practices, leading to increased productivity and sustainability in agriculture.

3.2 Selection of LPWAN technology

Among the various LPWAN technologies available, LoRaWAN has been chosen for the proposed system due to its numerous advantages and unique features compared to other alternatives such as SigFox, NB-IoT, and LTE-M:

- **Long Range:** LoRaWAN is known for its extensive range of up to 15 km in rural areas, which is generally longer than SigFox (with a range of 10 km) and NB-IoT (with a range of up to 10 km). This makes LoRaWAN more suitable for large-scale agricultural applications, which often require long-range communication.
- **Low Power Consumption:** While all LPWAN technologies focus on low power consumption, LoRaWAN devices can achieve lower power consumption than NB-IoT and LTE-M due to its use of spread spectrum modulation, which allows for better signal reception at lower power levels.

- **Scalability and Network Capacity:** LoRaWAN supports a large number of devices within a network and can handle more simultaneous connections than SigFox, which has a daily message limit per device. This makes LoRaWAN a more suitable choice for growing agricultural operations with numerous sensors and actuators.
- **Flexibility and Network Ownership:** Unlike SigFox, which is a proprietary network, and NB-IoT and LTE-M, which rely on cellular carriers, LoRaWAN offers a more flexible network infrastructure that allows users to deploy private or public networks, depending on their requirements. This flexibility provides more control over the network and its operation.
- **Cost-effectiveness:** LoRaWAN has a lower overall cost of ownership compared to NB-IoT and LTE-M. The open-source nature of LoRaWAN and its lower infrastructure requirements lead to reduced costs for deployment and maintenance. SigFox, being a proprietary technology, might also result in higher operational costs.

These unique advantages make LoRaWAN the most suitable choice for the proposed smart agriculture system.

3.3 IoT Protocol: MQTT

Within the sphere of IoT communication protocols, MQTT (Message Queuing Telemetry Transport) has been selected for this thesis, outperforming other alternatives such as CoAP and AMQP. The choice to use MQTT comes from its special features and strengths, which are explained below [54]:

- **Lightweight and Efficient:** MQTT is renowned for its lightweight nature and efficient data transmission, making it ideal for IoT applications where bandwidth and battery life are often limited. In contrast, AMQP, with its robust feature set, can be more resource-intensive, making it less suitable for constrained devices.
- **Quality of Service Levels:** MQTT provides three levels of Quality of Service (QoS), allowing for message delivery customization based on the application's requirements. This flexibility is not as readily available in CoAP, which primarily operates on a best-effort delivery model.
- **Persistent Sessions:** MQTT supports persistent sessions, which allow for the storage of a client's subscription and message information even when the client is offline. This feature is particularly beneficial in unstable network environments and is not offered by CoAP.

- **Publish/Subscribe Model:** MQTT operates on a publish/subscribe model, which is highly scalable and efficient for IoT applications involving a large number of devices. While AMQP or CoAP also supports this model, MQTT's implementation is simpler and more suitable for IoT devices.
- **Ease of Implementation:** MQTT's protocol design is straightforward, making it easier to implement on IoT devices compared to the more complex AMQP. This simplicity can lead to faster development times and lower maintenance costs.

These distinctive benefits make MQTT the most appropriate choice for the proposed IoT system. Its lightweight nature, flexible QoS levels, support for persistent sessions, efficient publish/subscribe model, and ease of implementation collectively contribute to its suitability for IoT applications.

3.4 IoT Platform: Azure IoT Central

For data visualization and management, Azure IoT Central has been chosen as the IoT platform for this project. The decision to use Azure IoT Central is based on the following factors [55]:

- **Compatibility:** Azure IoT Central is compatible with MQTT, allowing seamless communication with the other components of the system hosted on Microsoft Azure.
- **Integration:** Azure IoT Central can be easily integrated with the existing Microsoft Azure infrastructure, enabling a unified platform for managing both virtual machines and the IoT dashboard.
- **Scalability:** Azure IoT Central is a scalable platform that can support a growing number of devices and data, making it suitable for expanding agricultural operations.
- **Security:** Azure IoT Central offers robust security features to protect the data and communication within the system, ensuring the integrity and confidentiality of the information.
- **Ease of Use:** Azure IoT Central provides an intuitive interface and pre-built templates, simplifying the process of creating custom dashboards and visualizations for monitoring and managing agricultural data.
- **Advanced Analytics:** The platform supports integration with various Azure analytics services, allowing users to perform advanced data analysis and gain valuable insights for informed decision-making.

Considering these factors, Azure IoT Central is the optimal choice for the IoT platform in the proposed smart agriculture system. By using Azure IoT Central alongside the other components hosted on Microsoft Azure, the system can benefit from the platform's powerful features and seamless integration.

3.5 Setup & configuration of Microsoft Azure cloud platform

In this section, we will provide a step-by-step guide on how to configure and program a virtual machine in Microsoft Azure. Virtual machines (VMs) are a fundamental component of cloud computing, allowing users to run applications and services in a virtualized environment. Understanding the process of setting up and programming a VM in Microsoft Azure is essential for leveraging the platform's capabilities effectively.

Creating a Virtual Machine in Azure

1. **Step 1:** Log into your Azure account. Navigate to the Azure portal at <https://portal.azure.com> and enter your login credentials. Once logged in, click on the "Create a resource", and then "Create" located in the Virtual Machine section.
2. **Step 2:** Proceed to configure the VM settings. Begin by selecting the "Subscription" and "Resource group" that you've previously created. Subsequently, assign a name to your virtual machine; in this instance, we'll designate it as "Node-RED". The next step is to choose a geographic region for your VM's location, preferably one that is close to your current location to ensure optimal performance. When prompted for the VM's system image, opt for "Ubuntu". The rest of the options can typically be left at their default settings, unless specific adjustments are required.

On the security front, it's essential to set a username and password. These credentials will be used for accessing the virtual machine remotely, providing an added layer of security.

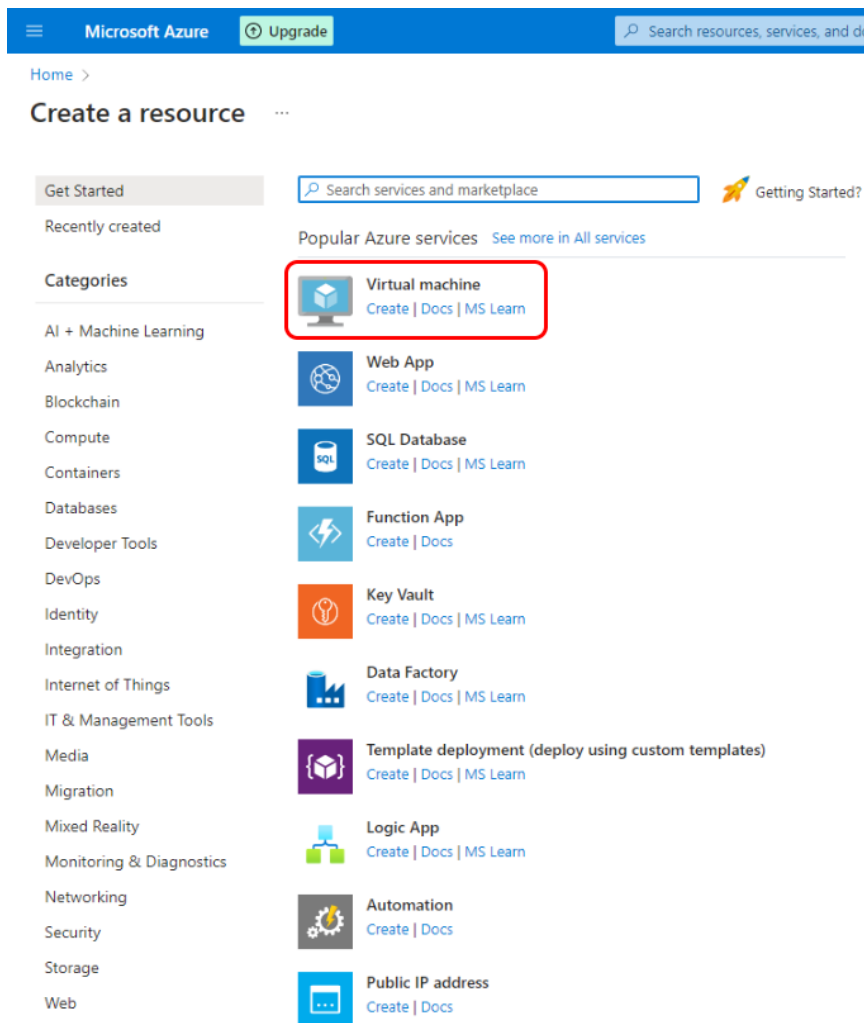


Figure 13: Create a Resource

Step 3: Click on "Review + Create" to review your settings. After validating your settings, click "Create" to start the VM deployment. After this, Azure will start deploying your virtual machine.

Step 4: After your VM is deployed, you need to configure specific ports to ensure access to the services you're planning to use. Navigate to the "Networking" section of the VM settings. Here, we will add several inbound port rules.

For our purposes, we need to add the following ports:

- **Port 1880:** This port is used to access Node-RED.
- **Port 320:** This port is used for MQTT (unsecured).
- **Port 330:** This port is also used for MQTT but with a secure connection.

Create a virtual machine ...

[Basics](#) [Disks](#) [Networking](#) [Management](#) [Monitoring](#) [Advanced](#) [Tags](#) [Review + create](#)

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [Learn more](#)

i This subscription may not be eligible to deploy VMs of certain sizes in certain regions.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	<input type="text" value="Azure subscription 1"/>
Resource group *	<input type="text" value="RG-Thesis"/> Create new

Instance details

Virtual machine name *	<input type="text" value="Node-RED"/>
Region *	<input type="text" value="(Europe) Norway East"/>
Availability options	<input type="text" value="No infrastructure redundancy required"/>
Security type	<input type="text" value="Standard"/>
Image *	<input type="text" value="Ubuntu Server 20.04 LTS - x64 Gen2 (free services eligible)"/> See all images Configure VM generation
VM architecture	<input type="radio"/> Arm64 <input checked="" type="radio"/> x64
Run with Azure Spot discount	<input type="checkbox"/>

Figure 14: Configuring Virtual Machine Details

- **Port 9001:** This port is used for MQTT over WebSocket, which allows you to connect to an MQTT broker from a browser. This is useful for web-based applications.

Size * ⓘ Standard_B1s - 1 vcpu, 1 GiB memory (\$9.64/month) (free services eligible) ▼
[See all sizes](#)

Administrator account

Authentication type ⓘ ☐ SSH public key ☒ Password

Username * ⓘ ✓

Password * ⓘ ✓

Confirm password * ⓘ ✓

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * ⓘ ☐ None ☒ Allow selected ports

Select inbound ports * ▼

⚠ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Figure 15: Review and Create VM

Priority	Name	Port	Protocol
300	⚠ SSH	22	TCP
310	Node-RED	1880	Any
320	MQTT-unsecure	1883	TCP
330	MQTT-secure	8883	TCP
340	MQTT-eclipse	9001	TCP
65000	AllowVnetInBound	Any	Any
65001	AllowAzureLoadBalancerInBound	Any	Any
65500	DenyAllInBound	Any	Any

Figure 16: Adding Inbound Port Rules

3.6 Development of Software & Configuration of LoRaWAN network

The primary focus of this master's thesis is to design and evaluate a LoRaWAN-based smart agriculture system within a simulated environment, rather than deploying physical sensors and devices in an actual agricultural setting. This simulation-focused approach permits comprehensive and flexible testing of the system components, along with their interactions, ensuring the system's robustness and efficiency before any potential real-world implementation.

The simulated environment is designed to imitate the behavior of an actual LoRaWAN network as closely as possible. It incorporates a range of software tools and cloud-based services to emulate the system's various components, all of which are hosted on virtual machines within the Microsoft Azure cloud platform. This cloud-based implementation facilitates scalability, reliability, and efficiency, making it an ideal testing ground for our simulated IoT system.

The following subsections delve into the specific software tools and services that form the backbone of our simulated system, explaining their roles and how they interact to create a complete IoT solution.

3.6.1 LWN Simulator

The LWN Simulator is a software tool used to simulate the transmission of data between the IoT devices (i.e., sensors and actuators) and the gateway in the LoRaWAN network. This simulator is capable of emulating various types of sensors and their respective data readings, replicating the communication patterns and data traffic seen in a real-world IoT system.

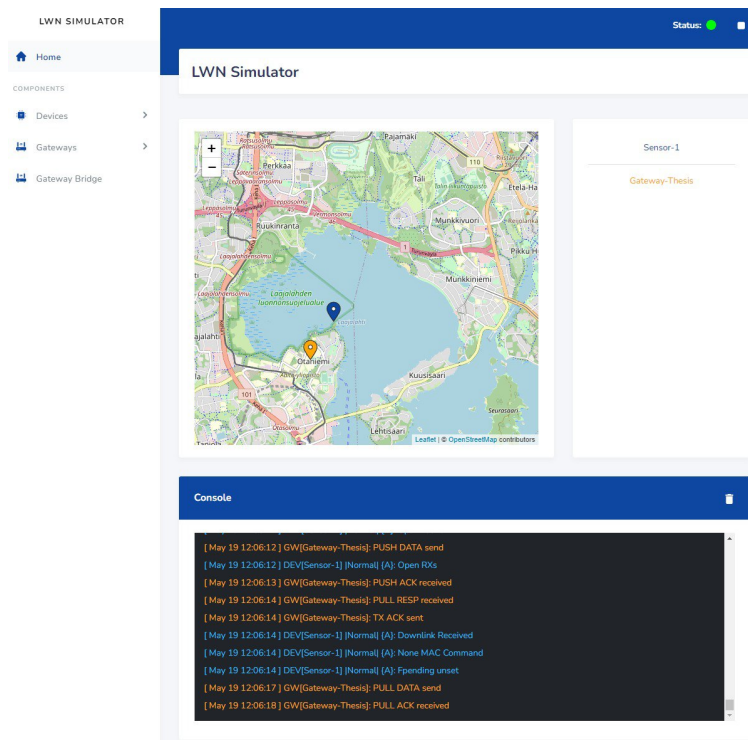


Figure 17: LWN Simulator

The LWN Simulator provides numerous features advantageous for our study:

- It allows the simulation of both LoRaWAN devices and gateways, providing a comprehensive view of the network.

- It is capable of mimicking real-world scenarios, including various environmental conditions and device behaviors.
- It supports the customization of various LoRaWAN parameters, enabling us to test our system under different configurations and conditions.

The installation of the LWN Simulator involves several steps, including setting up the Go programming environment, installing necessary dependencies, and finally, building and running the simulator. A summary of these steps are:

1. Create a directory for the LWN Simulator.
2. Download and extract the Go programming language.
3. Set the GOROOT and GOPATH environment variables.
4. Install the LWN Simulator dependencies.
5. Build and run the simulator.

Please refer to Appendix A for the detailed step-by-step guide, including the exact commands to be executed, for installing the LWN Simulator.

CONFIGURATION OF LWN SIMULATOR

Add new device

1. Open the LWN Simulator.
2. Navigate to the "Devices" section.
3. Click on "Add Device".

General settings

The general settings menu allows you to configure all the parameters related to the devices. For example, we can give our new device a descriptive name like "Temperature-1". We then generate a random DevEUI, which is a globally unique identifier for this device. Additionally, we select the correct region (in our case, EU868) to match the frequency band of our LoRaWAN devices, which in this case is 868 MHz.

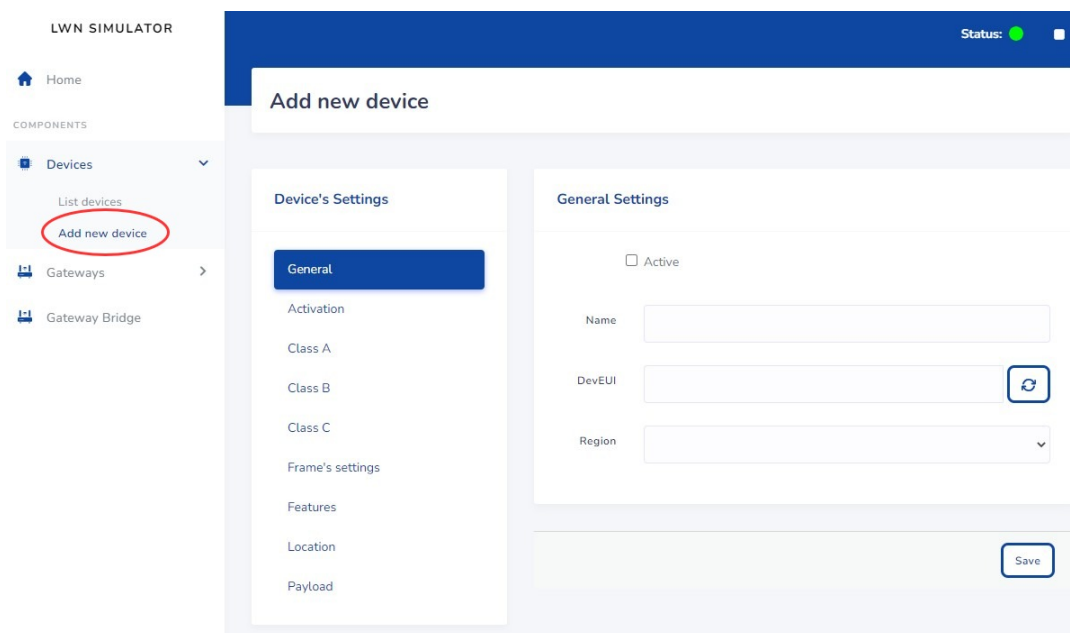


Figure 18: New device creation in the LWN Simulator

Figure 19: General settings configuration

Activation settings

In the activation settings, we first enable the "Otaa" supported option. OTAA stands for Over-the-Air Activation, which is a method for connecting devices in a LoRaWAN network. It provides a more secure connection and reduces the possibility of network impersonation. We then automatically generate the App Key, a secret key used during the OTAA process. The DevAddr, NwkSKey, and AppSKey fields are automatically filled during the OTAA process and cannot be manually modified.

Activation Settings

☒ Otaa supported

App Key
 6a642f662a3cda8c9651ca2fcc8ba8c8 ✓ [Copy] [Paste]

DevAddr
 [Empty] [Refresh]

NwkSKey
 [Empty] [Toggle] [Refresh]

AppSKey
 [Empty] [Toggle] [Refresh]

Figure 20: Activation settings configuration

Class A configuration

In this project, our devices will only use Class A, which is the base class of LoRaWAN. Class A devices allow for bidirectional communications where the server can transmit to the device after each device-initiated transmission. Let's look at the specific parameters:

- **RX1's settings:** This refers to the first receive window after an uplink transmission. The delay is set to 1000ms, which means the device will open a receive window 1 second after sending an uplink transmission. The duration is set to 3000ms, meaning the receive window will stay open for 3 seconds. The data rate offset of 3 is used to calculate the data rate for this window based on the uplink data rate.
- **RX2's settings:** This refers to the second receive window. The delay and duration are configured similarly to RX1's settings. The data rate for RX2 is fixed by the LoRaWAN specification for each region.
- **Channel frequency:** This is the frequency at which the device will listen during the receive windows. The value of 869MHz is one of the default values for the EU868 region.
- **Data Rate:** The data rate (DR0 in this case) corresponds to the modulation parameters for LoRa, such as the spreading factor and bandwidth. DR0 represents the lowest data rate, which means the longest range but also the smallest payload size and the longest transmission time.

- **ACK Timeout:** This is the amount of time the device will wait for an acknowledgment from the server after sending a confirmed uplink. If the acknowledgment is not received within this time, the device may choose to retransmit the uplink.

The configuration of Class A settings is shown in the figure below.

Class A Settings

RX1's settings

Delay

1000

Value in milliseconds. Default value is 1 second.

Duration

3000

Value in milliseconds. Default value is 3 second.

Data Rate offset

0

RX2's settings

Delay

1000

Value in milliseconds. Default value is 1 second.

Duration

3000

Value in milliseconds. Default value is 3 second.

Channel frequency

869525000

Value in Hz. Default value is 869525000

Data Rate

0

Default value is 0. ([Show Table](#))

ACK Timeout

2

Correct value between 1 and 3 seconds.Default value is 2 seconds.

Figure 21: Class A configuration

Frame's settings

In the "Frame's settings" section, we configure parameters for both uplink and downlink frames.

- **Data Rate (uplink):** Similar to the data rate in the Class A configuration, this parameter (set to 0) determines the modulation parameters for LoRa, such as the spreading factor and bandwidth.
- **FPort (uplink):** The FPort value (set to 1) is used to identify the application endpoint on the device that either receives the data (for downlink messages) or sends the data (for uplink messages).
- **Retransmission (uplink):** This value (set to 1) indicates how many times an uplink frame will be retransmitted if an acknowledgment is not received from the server within the ACK timeout.
- **FCnt (uplink):** The Frame Counter (FCnt) value (set to 4) is used to prevent replay attacks. Each time a frame is sent, the counter is incremented.
- **FCntDown (downlink):** The Frame Counter for Downlink (FCntDown, set to 3) serves a similar purpose to the FCnt in the uplink, but for downlink frames. It is also used to prevent replay attacks.

The configuration of Frame's settings is shown in the figure below.

Frame Settings

Uplink

Data Rate

0

(Show Table)

FPort

1

Correct value between 1 and 223.

Retransmission

1

Retransmission for ConfirmedData Uplink

FCnt

1

Downlink

☐ (FCntDown) Disable frame-counter validation.

FCntDown

0

Figure 22: Frame's settings

Features Settings

In the "Features Settings" section, we enable the Adaptive Data Rate (ADR) feature. ADR is a feature where the Network Server can control the data rate of individual nodes, aiming for the best trade-off between energy consumption and communication range. We also set the antenna range, which in this case is left at the default value of 10.000 meters.

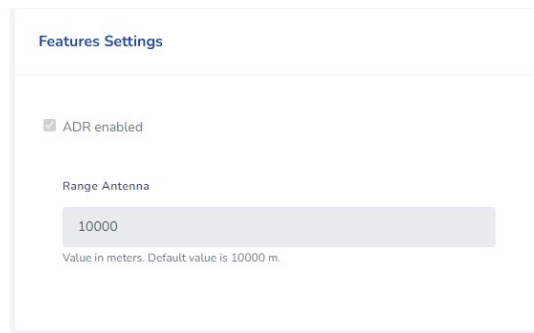
The image shows a web-based configuration interface titled "Features Settings". It contains two main settings: a checkbox labeled "ADR enabled" which is checked, and a numeric input field labeled "Range Antenna" with the value "10000". Below the input field, a small text note states "Value in meters. Default value is 10000 m." The interface is clean and modern, with a light gray background and blue accents for the title and checkbox.

Figure 23: Features Settings configuration

Location & Payload

In the location section, we set the geographical location of the device. This information can be used for location-based services or for network planning and troubleshooting.

In the payload section, we configure the uplink interval, which is the time between successive uplink frames. This parameter may be set based on the specific requirements of your use case. For this example, we will set it to the default value of 240 seconds.

If the payload exceeds the maximum size allowed by regional parameters, we select "Fragments the frame". This allows the payload to be split across multiple frames, ensuring that all data is transmitted even when it exceeds the maximum payload size.

As for MType, we select "ConfirmedDataUp". This means that every uplink frame requires a confirmation from the network server. This is useful when it's important to ensure that data has been received correctly by the network server.

Finally, we write the "Payload" we want to send. As an example, we will write "Hello!".

Payload Settings

Uplink Interval

240

The expected interval in seconds in which the device sends uplink. Default value is 10 seconds.

If the payload exceeds the maximum size allowed by regional parameters:

☒ Fragments the frame ☐ Truncates the frame

MType:

☒ ConfirmedDataUp ☐ UnConfirmedDataUp

Payload

Hello!

Figure 24: Payload

Creating a Gateway

In order to create a gateway in LWN Simulator, we have to initiate the process by clicking on the "Add New Gateway" button. This will present us with several options for the type of gateway we want to add. For our purpose, we select "Virtual Gateway" and proceed by clicking on the "Active" option to ensure that our gateway is active upon creation.

Now we have to set a name to our gateway. For our example, we decide to name our gateway "Gateway-Thesis", a name that helps us easily identify this particular gateway among others in our network.

The next step involves generating a MAC address for our gateway. This can be done by clicking on the right button, next to the MAC address field. This MAC address serves to uniquely identify our gateway in the network.

We also need to set the "KeepAlive" parameter, which defines the time interval between two consecutive PING requests from the gateway to the LWN server. In our case, we will set the default value of 30 seconds.

Furthermore, we are required to configure the location of our gateway. This information is key for network planning and optimization.

Once all the necessary information is accurately filled, we finalize the creation of our gateway by clicking the "Save" button.

Figure 25: Gateway creation in the LWN Simulator

3.6.2 ChirpStack

ChirpStack is an open-source LoRaWAN network server that forms the backbone of the proposed smart agriculture system. As the primary hub for data communication, ChirpStack manages the interactions between the LoRaWAN gateway and the IoT devices, ensuring the secure and efficient transmission of data.

In the context of our system, ChirpStack has been deployed on an Ubuntu server, providing a stable and secure platform for data management. The decision to use ChirpStack was influenced by its robust feature set, which includes support for multiple gateways, flexible device management, and a powerful RESTful API for easy integration with other software components. Furthermore, ChirpStack’s comprehensive security features, including encryption and device authentication, are crucial for maintaining the integrity and confidentiality of the sensor data.

To simulate the operation of the LoRaWAN network, ChirpStack was configured to work with the LWN Simulator. This allowed us to replicate the behavior of real IoT

devices and gateways, providing valuable insights into the performance of the system under different conditions.

A detailed guide on how to install and configure ChirpStack on an Ubuntu server is provided in Appendix B.

CONFIGURATION OF CHIRPSTACK

In the ChirpStack ecosystem, an application serves as a logical grouping for devices. This abstraction is useful because it allows us to manage a set of devices that serve a common purpose or belong to a specific project or deployment. By creating an application, we can group, manage, and handle data for devices more effectively.

Create a new application

On the left-hand side of the ChirpStack dashboard, click on "Applications", and then click on the "+ Create" button to add a new application:

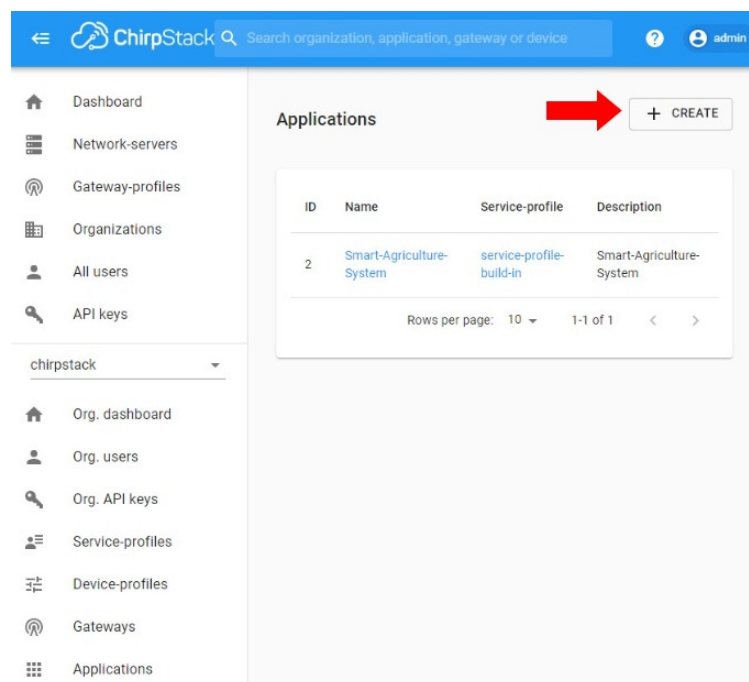


Figure 26: Application Creation Screen

On the "Create application" page, you will need to enter the following details:

- Application name: This can be any name that you choose to identify the application.
- Application description: A brief description of the application.
- Service-profile: The service-profile to which this application will be attached

After entering these details, click on "Create application."

Applications / Create

Application name *
App-1
The name may only contain words, numbers and dashes.

Application description *
App-1

Service-profile *
service-profile-build-in
The service-profile to which this application will be attached. Note that you can't change this value after the application has been created.

CREATE APPLICATION

Figure 27: Create application

Device Configuration

Next, configure the device within the application. So, we have to click on "Devices". and then click on the "+ Create" button to add a new device. On the "Create device" page, you will need to enter the following details:

- Device name: This can be any name that you choose to identify the device.
- Device description: A brief description of the device.
- Device EUI: This is the unique identifier for the device, and can typically be found on the device or in its accompanying documentation.
- Device-profile: We select "device_profile_otaa" since our devices uses the OTAA method for activation.

After entering these details, click on "Create device."

Applications / Smart-Agriculture-System / Devices /

Create

GENERAL

VARIABLES

TAGS

Device name *

device-1

The name may only contain words, numbers and dashes.

Device description *

device-1

Device EUI *

b2 7c 17 db 94 85 70 f8

MSB

Device-profile *

device_profile_otaa

☐ Disable frame-counter validation

Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.

☐ Device is disabled

ChirpStack Network Server will ignore received uplink frames and join-requests from disabled devices.

CREATE DEVICE

Figure 28: Device Configuration Screen

Gateway Configuration

After setting up the device, we have to configure the gateway. On the left-hand side of the ChirpStack dashboard, click on "Gateways", and then, click on the "+ Create" button to add a new gateway.

On the "Create gateway" page, you will need to enter the following details:

- Gateway name: This can be any name that you choose to identify the gateway.
- Gateway description: A brief description of the gateway.
- Gateway ID: This is the unique identifier for the gateway, and is typically the MAC address of the gateway.

In this page there are more parameters to configure but we set the default values. After entering these details, click on "Create gateway."

Gateways / Create

GENERAL

TAGS

METADATA

Gateway name *
Gateway-Thesis
The name may only contain words, numbers and dashes.

Gateway description *
Gateway-Thesis

Gateway ID *
c3 cf e1 8d 61 3b a2 be
MSB

Network-server *
build_in_ns
Select the network-server to which the gateway will connect. When no network-servers are available in the dropdown, make sure a service-profile exists for this organization.

Service-profile
service-profile-build-in
Select the service-profile under which the gateway must be added. The available service-profiles depend on the selected network-server, which must be selected first.

Gateway-profile
Gateway-profile-1
Optional. When assigning a gateway-profile to the gateway, ChirpStack Network Server will attempt to update the gateway according to the gateway-profile. Note that this does require a gateway with ChirpStack Concentratord.

Figure 29: Gateway Configuration Screen

3.6.3 MQTT Broker and Node-RED

In the proposed smart agriculture system, messages are sent from ChirpStack to Node-RED through the MQTT protocol, enabling efficient data management and system integration.

The MQTT broker, specifically Eclipse Mosquitto in our system, serves as the open-source message broker that handles the transmission of data. It ensures secure and reliable delivery of data from the LoRaWAN network server to the application server.

Node-RED, on the other hand, is a powerful programming tool, allows for the seamless integration of hardware devices, APIs, and online services. With its browser-based flow editor, Node-RED enables the creation of data flows, including reading data from the MQTT broker, processing it, and sending it to an IoT dashboard. In our system, Node-RED processes and routes data from the MQTT broker to the Azure IoT Central dashboard.

Both the MQTT broker and Node-RED were installed using Docker, a platform

that automates the deployment, scaling, and management of applications. By utilizing Docker containers, we ensured consistent and uniform execution across different environments, simplifying deployment and reducing potential issues. Docker also provides convenient management commands to start, stop, and monitor the running containers.

Detailed instructions on installing Docker, setting up the MQTT broker and Node-RED, and configuring the data flows can be found in Appendix C. The instructions include creating Docker Compose configuration files, defining necessary network ports and volumes, and initiating the Docker containers.

CONFIGURATION OF NODE-RED

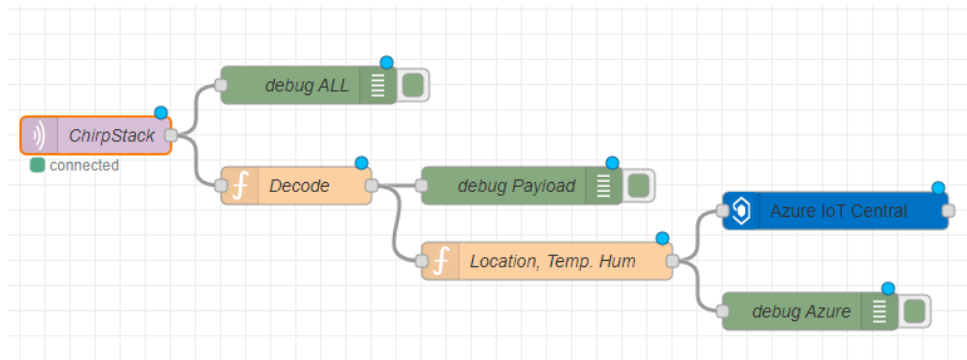


Figure 30: Node-RED configuration

MQTT In Node ("ChirpStack")

This node subscribes to an MQTT broker at 20.251.11.28 on port 1883, subscribing specifically to the topic `application/2/device/a76bd11919dd2e12/rx`. When a message is published to the topic, this node receives it and forwards it to the next nodes in the workflow.

Debug Node ("debug ALL")

This node prints all information from the received message to the Node-RED debug console. This is useful for understanding the raw input to your workflow and for diagnosing any problems that might occur.

Function Node ("Decode")

This node takes as input the output of the MQTT node. It extracts the 'data' field from the received message payload, which is a string encoded in Base64, decodes this

The image shows the configuration window for an MQTT node. It has a title bar with a gear icon and the word "Properties". Below the title bar are several configuration fields:

- Server:** A dropdown menu showing "20.251.11.28:1883" with a pencil icon to its right.
- Action:** A dropdown menu showing "Subscribe to single topic".
- Topic:** A text input field containing "application/2/device/a76bd11919dd2e12/rx".
- QoS:** A dropdown menu showing "0".
- Output:** A dropdown menu showing "auto-detect (parsed JSON object, string or buf)".
- Name:** A text input field containing "ChirpStack".

Figure 31: MQTT node configuration

Base64 string to a UTF-8 string, and then replaces the original message payload with this decoded string.

```

1 // Extract the data property from the input message and decode it from base64 to text
2 const base64Data = msg.payload.data;
3 const decodedData = Buffer.from(base64Data, 'base64').toString('utf8');
4
5 // Set the output message's payload to the decoded data value
6 msg.payload = decodedData;
7
8 // Return the modified message object
9 return msg;

```

Figure 32: Decode node configuration

Function Node ("Temp, Hum & Smoke")

This node takes as input the output of the "Decode" node. It splits the decoded string (which should be a comma-separated list of numbers) into an array of numbers. It then interprets these numbers as temperature, humidity, and smoke readings, adds a random offset to each, clamps the temperature and humidity to a valid range, and repackages these as a new message payload.

```

1  // Parse the input string into an array of numbers
2  let values = msg.payload.split(',').map(Number);
3
4  // Add a random value between -10 and 10 to each input value
5  let temperature = values[0] + Math.floor(Math.random() * 21) - 10;
6  let humidity = values[1] + Math.floor(Math.random() * 21) - 10;
7  let smoke = values[2] + Math.floor(Math.random() * 21) - 10;
8
9  // Clamp the temperature and humidity values to their valid ranges
10 temperature = Math.max(Math.min(temperature, 40), 0);
11 humidity = Math.max(Math.min(humidity, 100), 0);
12
13 // Create the output object with the updated values
14 let output = {
15     temperature: temperature,
16     humidity: humidity,
17     smoke: smoke
18 };
19
20 // Set the output object as the payload of the message
21 msg.payload = output;
22
23 // Return the message
24 return msg;

```

Figure 33: Temp function node configuration

Azure IoT Central Node

This node sends the message from the previous node to an Azure IoT Central application, specifically to a device with ID "temperature_sensor".

Properties	
Transport	MQTT
Authentication	SAS
Scope ID	0ne00A1D0E4
Device ID	temperature_sensor
Primary Key	Xb9NEyNuit9g4EYGTu2n8tcUnZsNXGGMkbltT5z

Figure 34: Azure IoT Central node configuration

- **Transport (MQTT):** The protocol for communication between the node and Azure IoT Central is specified by this setting. Due to its small code footprint and suitability with constrained network capacity, MQTT, or Message Queuing Telemetry Transport, is a lightweight protocol frequently used in IoT contexts.

- **Authentication (SAS):** SAS, or Shared Access Signature, is the selected method for node authentication with Azure IoT Central. A SAS is a security token encapsulated in a string, used for authenticating requests to Azure IoT Central. It's derived from a shared access policy and encompasses information regarding the permissions assigned by the token.
- **Scope ID:** This unique ID identifies the Device Provisioning Service (DPS) instance within an IoT Central application. Each IoT Central application has a unique Scope ID used by devices to connect to their respective IoT Central applications.
- **Device ID:** The Device ID is a unique identifier for a device within the IoT Central application. It's used by IoT Central to manage and track the device, and it must be unique within the application.
- **Primary Key:** Each device in IoT Central is assigned two symmetric keys, the primary and secondary keys. These keys generate SAS tokens used by devices to authenticate themselves with IoT Central. The primary key is typically used as the main key, while the secondary key serves as a backup.

3.7 Configuration and Development of Azure IoT Central

Azure IoT Central, a fully managed global Internet of Things (IoT) Software as a Service (SaaS) solution, simplifies the process of connecting, managing, and scaling IoT devices. It provides reliable and secure communication between devices and the cloud, offering users a comprehensive platform to monitor and manage IoT devices at scale.

The platform removes the complexities of managing the underlying infrastructure, allowing users to focus only on those components they really need. Azure IoT Central allows users to visualise and analyse device data in real time, create custom dashboards, and set alerts and notifications for device monitoring.

In our Smart Agriculture system, Azure IoT Central visualizes data from the devices, which is transmitted via the MQTT protocol from the previous software components, such as the LWN Simulator, ChirpStack, and Node-RED. This presentation of data facilitates user-friendly interpretation and analysis.

Azure IoT Central's compatibility with MQTT, a protocol extensively used in this project for data transmission, is a primary reason for its selection. Additionally, its seamless integration with other Azure services simplifies the architecture since Microsoft Azure is already in use as the project's cloud infrastructure.

CONFIGURATION OF AZURE IOT CENTRAL

Creating an Azure IoT Central Application

The first step involves creating an Azure IoT Central application. Navigate to the Azure IoT Central website and click on "My apps" button followed by the "New Application". Then, select the "Create custom app" option. Next, define the application parameters such as the "Application name", URL for dashboard access, and the "Pricing plan". For this project, the Standard 0 plan was chosen.

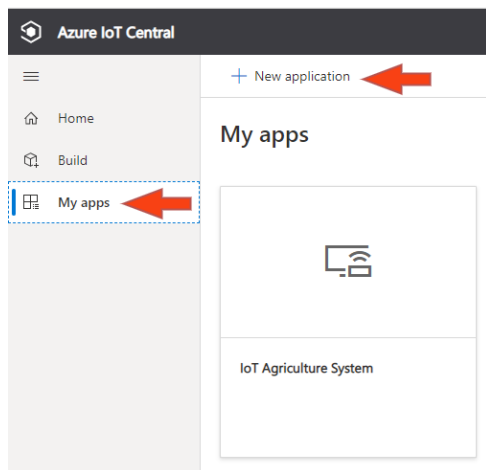


Figure 35: Azure IoT Central Application

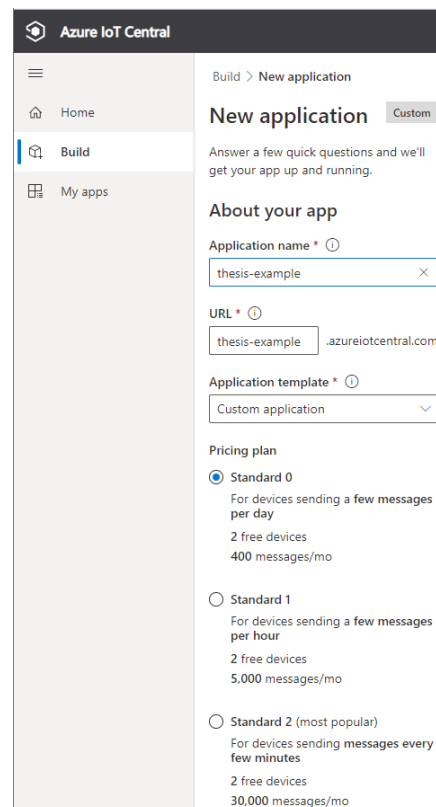


Figure 36: New application configuration

Define Device Templates

The first step in integrating a device with our system involves defining a "device template" in the Azure IoT Central application. This template encapsulates the characteristics and behaviors of the type of device connecting to the application.

To create a device template, we navigate to "Device templates" and click on "+ New". In the subsequent menu, we select the "IoT device" option, provide a name for our template, and then choose the "Custom model" option.

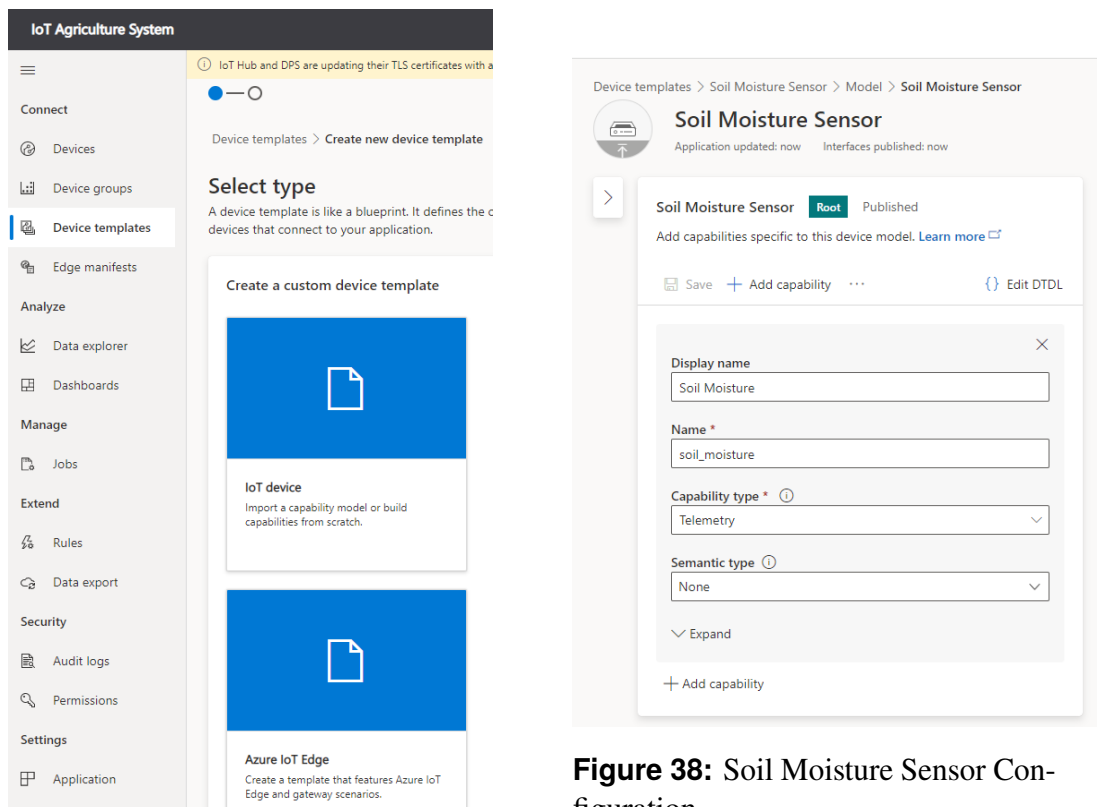


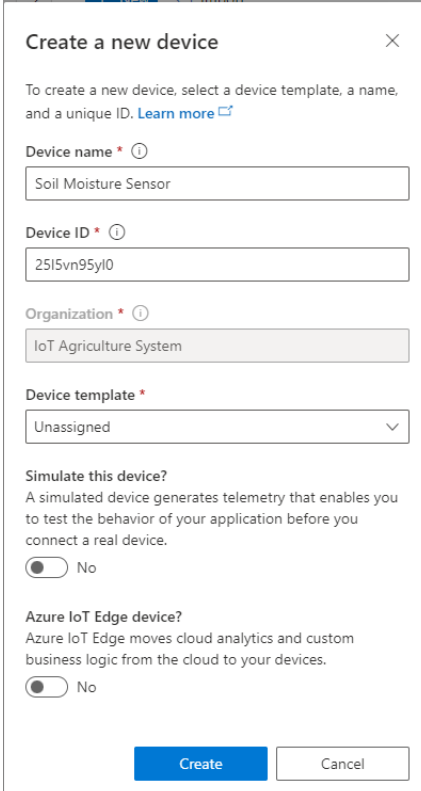
Figure 37: New Template Configuration

Figure 38: Soil Moisture Sensor Configuration

Once the device template is created, we configure the capabilities or the Key Performance Indicators (KPIs) that the device will transmit. For instance, when integrating a soil moisture sensor, we define a capability named "soil moisture", representing moisture levels as a percentage (%). This capability type is classified as Telemetry, indicating that it involves the automatic transmission and measurement of data from remote or inaccessible points. Upon defining these parameters, we finalize the process by clicking on "Save" and "Publish".

Add Devices

To create a device, click on "Devices", followed by the "+ New" button. In this menu, specify the device name, Device ID (the default one can be used), the previously set Organization and the Device template we created previously. After entering the details, click on "Create" to save the configuration.



Create a new device ✕

To create a new device, select a device template, a name, and a unique ID. [Learn more](#)

Device name * ⓘ
Soil Moisture Sensor

Device ID * ⓘ
25I5vn95yl0

Organization * ⓘ
IoT Agriculture System

Device template *
Unassigned ▼

Simulate this device?
A simulated device generates telemetry that enables you to test the behavior of your application before you connect a real device.
☒ No

Azure IoT Edge device?
Azure IoT Edge moves cloud analytics and custom business logic from the cloud to your devices.
☒ No

Create **Cancel**

Figure 39: Device Configuration

Creating a Dashboard

A dashboard to visualize various parameters at the same time can be created by clicking on "Edit dashboard", followed by choosing a "visual". For example, we choose "Last known value (LKV)", which allows us to see the last recorded value of the Soil Moisture. Next, set the parameters of the tool such as the Title, Device group, and the size of the numbers, among others.

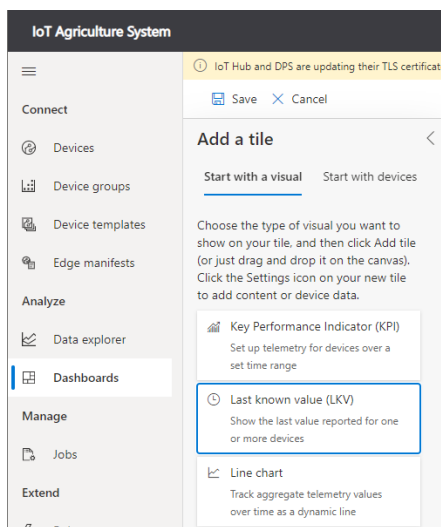


Figure 40: Dashboard Configuration

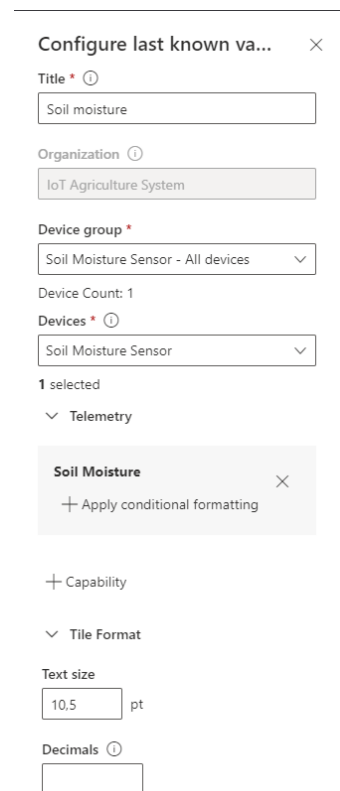


Figure 41: LKV configuration

4 Results, Visualization & Automations

4.1 Visualization

As discussed previously, the core objective of this Master's Thesis is the conceptualization, design, and simulation of a Smart Agriculture System. To make the simulation as close to reality as possible, all generated data are randomized yet remain within the actual operational ranges for each type of sensor.

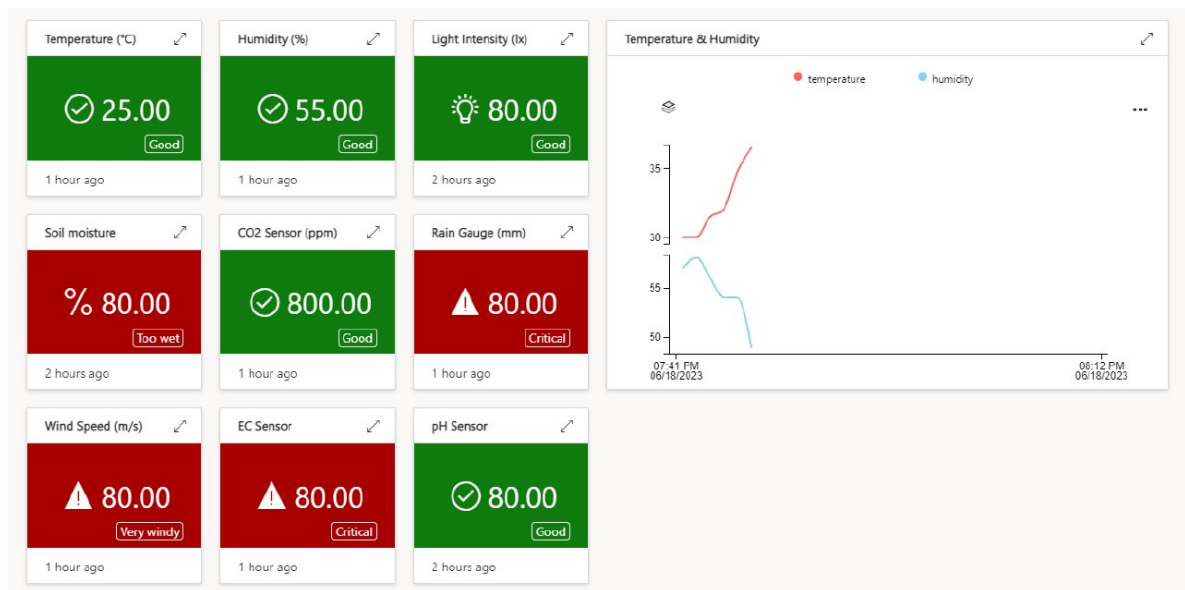


Figure 42: Azure IoT Central dashboard

We have developed a dynamic dashboard on the Azure IoT Central platform. It displays real-time data, as shown in the previous figure. The dashboard is a single point of access for all vital statistics and has been optimized to enhance user interaction. It uses color-coding and displays immediate alert messages when sensor readings cross defined thresholds.

These are the crucial sensor readings that could signal potential problems affecting crop health and yield:

- **Temperature Sensor:** The critical temperature ranges depend on the specific crop. Temperatures below 0°C risk causing frost damage, while those above 35°C might trigger heat stress conditions.
- **Humidity Sensor:** High humidity above 85% may encourage the growth of fungi, and low humidity under 30% could cause dehydration stress in plants.
- **Soil Moisture Sensor:** Low soil moisture under 10% can hinder plant growth, while high moisture over 40% can starve roots of oxygen and make them more susceptible to diseases.

- **Light Intensity Sensor:** High light intensity over 100,000 lx can lead to photoinhibition, whereas low levels below 5,000 lx can restrict photosynthesis, resulting in stunted plant growth.
- **pH Sensor:** Most crops prefer mildly acidic to neutral soil pH levels, thus, levels below 5.5 or above 7.5 may cause nutrient uptake problems or toxicity.
- **EC Sensor:** High soil salinity over 4 dS/m can negatively affect plant water absorption and upset the nutrient balance.
- **Rain Gauge:** Heavy and persistent rainfall can cause soil erosion and waterlogging. The critical range of this sensor varies based on the crop needs and local weather conditions.
- **Wind Speed sensor:** High wind speeds over 15 m/s can damage plants and increase water loss. On the other hand, lack of air movement can result in poor circulation and increased disease risk.
- **CO2 Sensor:** Low CO2 concentrations below 200 ppm may limit photosynthesis. In contrast, levels above 1,500 ppm usually don't add any extra benefits and might suggest poor ventilation in indoor settings.

With respect to the final structure in Node-RED for data management, the following figure illustrates how the nodes are deployed:

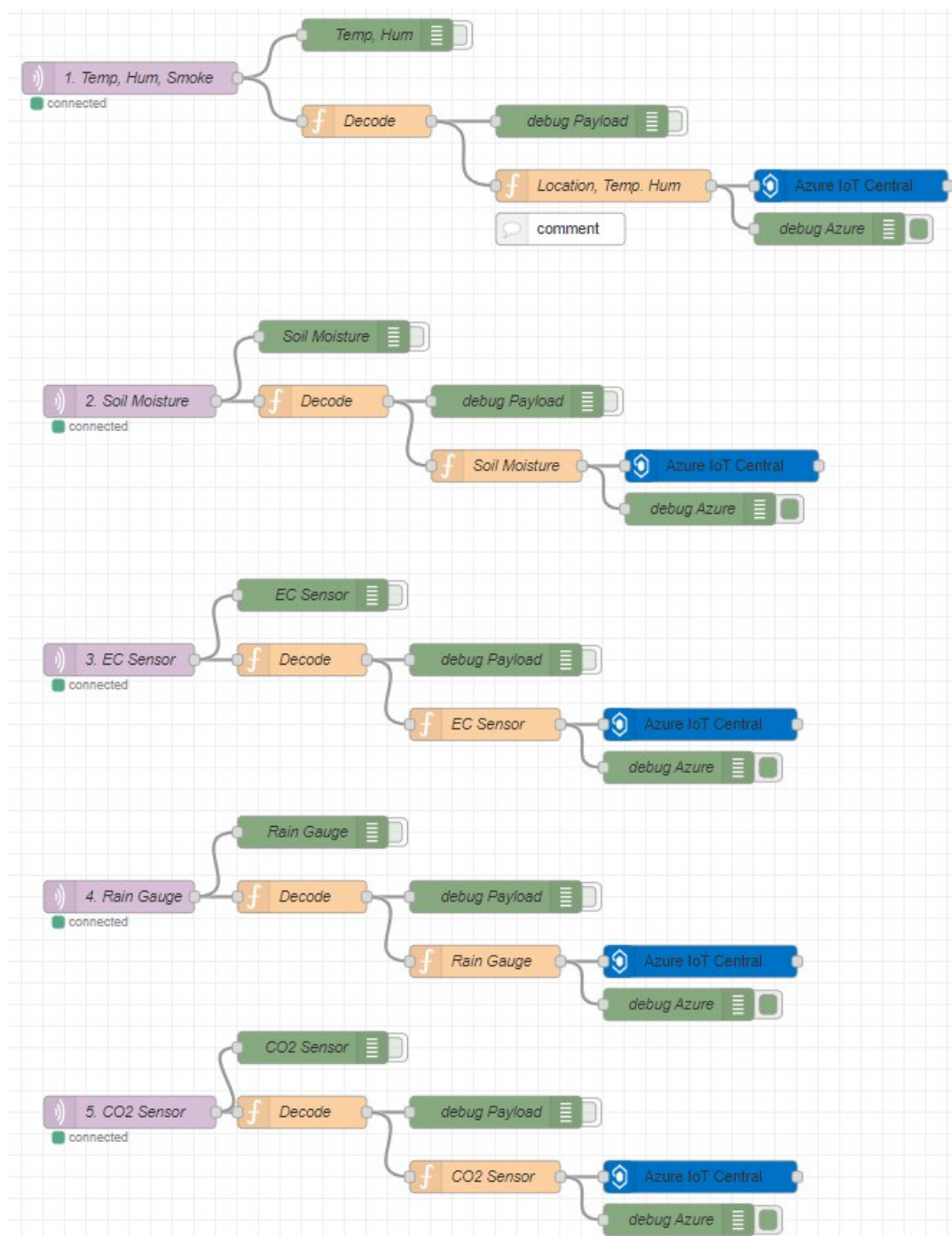


Figure 43: Node-RED Structure

4.2 Automations

The data collected from a suite of sensors in the agricultural field presents an opportunity for robust automation, improving overall farm management and decision-making processes. By leveraging Microsoft Azure's data processing and machine

learning capabilities, some automations can be created, which will respond to real-time changes in conditions detected by these sensors. Each sensor provides vital insights into the on-ground conditions of the agricultural field, forming the backbone of the smart system.

- **Smart Watering System:** By using data from soil moisture, temperature, and humidity sensors, we can design a system that automatically waters crops when needed. This system would trigger watering when the soil is too dry, or when the temperature is too high or humidity too low. This approach saves water and keeps crops well-watered [56].
- **Adjustable Greenhouse Lighting:** Using light intensity data, an automated system can be developed for the dynamic management of light conditions in greenhouse farming. The system would activate supplementary lighting when ambient luminosity falls short of the optimal threshold, offering a balanced light environment for the plants [57].
- **Automated Fertilizer Administration:** By leveraging data from EC sensors (which capture soil nutrient concentration) and pH sensors, we can introduce automation in the fertilizer dispensation process. The system would add fertilizer when the nutrient levels are too low, or adjust the soil acidity when it is too high or low [57].
- **Smart Ventilation System:** Employing data from CO2 sensors and temperature sensors, the proposed system can control the ventilation in greenhouses or indoor farming systems. In response to excessive CO2 levels or a temperature spike, the system could execute the ventilation process, maintaining an ideal growing environment [57].
- **Predictive Pest and Disease Alert System:** By using all the sensor data and machine learning techniques from Azure, we can create a system that predicts when there might be an outbreak of pests or diseases. The system would alert farmers early, based on past data and current conditions [58].
- **Intelligent Alerts:** The system can intelligently analyze all sensor data to generate alerts or suggestions when it foresees potential challenges, or manual intervention is required. Alerts could be related to changing weather, risk of pests, or low nutrient levels in the soil.

Through these automations, the proposed smart agriculture system could maintain optimal farming conditions, potentially leading to enhanced crop yields, operational efficiency, and resource conservation while minimizing manual work.

5 Conclusions

5.1 Summary of the study

This Master's thesis aimed to design and evaluate a LoRaWAN-based Smart Agriculture system using Microsoft Azure as the cloud platform. The study involved investigating and comparing LPWAN technologies and IoT communication protocols suitable for Smart Agriculture systems. A LoRaWAN-based sensor network was designed and implemented for data collection in agricultural fields. The LWN Simulator was utilized to simulate sensor data, which was then integrated with the ChirpStack network server. Data processing and decoding mechanisms were developed using Node-RED, and data transmission was achieved through the MQTT protocol. The collected data was visualized on the Azure IoT Central dashboard.

5.2 Implications of the study

The findings of this study have significant implications for the field of Smart Agriculture. The integration of LoRaWAN technology and Microsoft Azure has demonstrated the feasibility and effectiveness of real-time monitoring of agricultural parameters. The proposed system provides actionable insights to farmers, facilitating informed decision-making for optimized agricultural practices. The comparative analysis of LPWAN technologies and IoT communication protocols offers valuable insights for selecting suitable options in Smart Agriculture systems. The importance of data visualization for intuitive interpretation and analysis of agricultural data is also highlighted.

5.3 Recommendations for future research

Based on the outcomes of this study, several avenues for future research can be explored. Further optimization of the system's performance in terms of data accuracy, reliability, and power consumption is recommended. This can involve refining data processing mechanisms and exploring advanced algorithms for improved data decoding and analysis. Integration of machine learning techniques can be considered to enable predictive analytics and decision support systems for farmers. Field deployment and real-world validation of the proposed system under diverse agricultural conditions and settings would provide valuable insights. Expansion of the system to incorporate additional agricultural parameters and the integration of emerging technologies can further enhance Smart Agriculture systems.

In conclusion, this Master's thesis successfully addressed the research problem of designing and evaluating a LoRaWAN-based Smart Agriculture system using Microsoft Azure. The study contributed insights into LPWAN technologies, IoT communication protocols, and their integration with cloud platforms for agricultural

applications. The designed system showcased the potential of real-time monitoring and data-driven decision-making in agriculture. The findings contribute to the existing knowledge and open up avenues for future research and advancements in the field of Smart Agriculture systems.

References

- [1] A. Gupta and P. Nahar, "Classification and yield prediction in smart agriculture system using IoT," *Journal of Ambient Intelligence and Humanized Computing*, Jan. 2022, doi: 10.1007/s12652-021-03685-w.
- [2] J. Arshad et al., "Implementation of a LORAWAN based Smart Agriculture decision support system for optimum crop yield," *Sustainability*, vol. 14, no. 2, p. 827, Jan. 2022, doi: 10.3390/su14020827.
- [3] S. Dwi Putra, C. Olivia Sereati, and H. Sutrisno, "Design of IoT Monitoring System Based on LoRaWAN Architecture for Smart Green House," *IOP Conf. Series: Earth Environmental Sci.*, vol. 1012, no. 1, pp. 012090, Apr. 2023, doi: 10.1088/1755-1315/1012/1/012090.
- [4] A. Valente, C. Costa, L. S. Pereira, B. Soares, J. Lima, and S. Soares, "A LoRaWAN IoT system for smart agriculture for vine water status determination," *Agriculture*, vol. 12, no. 10, p. 1695, Oct. 2022, doi: 10.3390/agriculture12101695.
- [5] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, "A survey of LORAWAN for IoT: From Technology to Application," *Sensors*, vol. 18, no. 11, p. 3995, Nov. 2018, doi: 10.3390/s18113995.
- [6] W. D. Paredes, H. Kaushal, I. Vakulinia, and Z. G. Prodanoff, "LORA Technology in Flying Ad Hoc Networks: A survey of Challenges and open issues," *Sensors*, vol. 23, no. 5, p. 2403, Feb. 2023, doi: 10.3390/s23052403.
- [7] R. Gupta, S. B. Dhok, and R. B. Deshmukh, "LoRaWAN: A Comprehensive Review on Network Architecture, Specifications and Applications," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, Oct. 2020, pp. 1125-1130.
- [8] S. S. R. Bogale, L. C. B. León, and T. R. Newman, "LoRaWAN: An Adaptive Physical Layer Solution for IoT Applications," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, Sep. 2021, pp. 1-6.
- [9] D. D. Donno, Y. Dong, and M. H. Rehmani, "LoRaWAN Security: A Comprehensive Review," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 463-492, Firstquarter 2021.
- [10] M. R. Palattella et al., "On the Performance of LoRaWAN for Adaptive Data Rate Industrial IoT Applications," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019, pp. 568-573.

- [11] R. Lie, "LoRa/LoRaWAN tutorial 13: Symbol, Spreading Factor and Chip," *Mobilefish.com*, 2018. [Online]. Available: <https://www.youtube.com/watch?v=0FCrN-u-Vpw>
- [12] The Things Network, "Spreading Factors," The Things Network, Nov. 26, 2021. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/>.
- [13] M. Iqbal, A. S. Abdullah, and F. Shabnam, An Application Based Comparative Study of LPWAN Technologies for IoT Environment. 2020. doi: 10.1109/tensymp50017.2020.9230597.
- [14] Student Circuit, "What is SigFox technology," Student Circuit, Oct. 2019, [Online]. Available: <https://www.student-circuit.com/learning/year3/iot/what-is-sigfox-technology/>
- [15] S. C. Gaddam and M. K. Rai, A Comparative Study on Various LPWAN and Cellular Communication Technologies for IoT Based Smart Applications. 2018. doi: 10.1109/icetietr.2018.8529060.
- [16] M. a. M. Almuhaaya, W. A. Jabbar, N. Sulaiman, and S. Abdulmalek, "A Survey on LoRaWAN Technology: Recent Trends, Opportunities, Simulation Tools and Future Directions," *Electronics*, vol. 11, no. 1, p. 164, Jan. 2022, doi: 10.3390/electronics11010164.
- [17] "BLRLABS | IOT Services." <https://www.blrlabs.com//pcs-nb-iot.html>
- [18] S. C. Gaddam and M. K. Rai, A Comparative Study on Various LPWAN and Cellular Communication Technologies for IoT Based Smart Applications. 2018. doi: 10.1109/icetietr.2018.8529060.
- [19] M. Iqbal, A. S. Abdullah, and F. Shabnam, "An Application Based Comparative Study of LPWAN Technologies for IoT Environment," 2020. doi: 10.1109/tensymp50017.2020.9230597.
- [20] E. Pasqua, "Satellite IoT connectivity: Three key developments to drive the market size beyond \$1 billion," *IoT Analytics*, Aug. 2022, [Online]. Available: <https://iot-analytics.com/satellite-iot-connectivity/>
- [21] "TSAT - Satellite backhaul for industrial IoT devices - TSAT," TSAT, Sep. 06, 2022. <https://tsat.net/tsat-satellite-backhaul-for-industrial-iot-devices/>
- [22] M. Luglio, M. Marchese, F. Patrone, C. Roseti, and F. Zampognaro, "Performance Evaluation of a Satellite Communication-Based MEC Architecture for IoT Applications," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 58, no. 5, pp. 3775–3785, Jan. 2022, doi: 10.1109/taes.2022.3199330.
- [23] E. K. Kim and H.-S. Jo, "Problem and solution for NB-IoT uplink in Low Earth Orbit satellite communication," 2022. doi: 10.1109/iceic54506.2022.9748291.

- [24] G. Boquet, P. Tuset-Peiro, F. Adelantado, T. Watteyne, and X. Vilajosana, "LR-FHSS: Overview and performance analysis," *IEEE Communications Magazine*, vol. 59, no. 3, pp. 30–36, 2021.
- [25] M. A. Ullah, K. Mikhaylov, and H. Alves, "Analysis and simulation of Lorawan LR-FHSS for direct-to-satellite scenario," *IEEE Wireless Communications Letters*, vol. 11, no. 3, pp. 548–552, 2022.
- [26] A. Maleki, H. H. Nguyen, and R. Barton, "Outage probability analysis of LR-FHSS in satellite IOT Networks," *IEEE Communications Letters*, vol. 27, no. 3, pp. 946–950, 2023.
- [27] N. Islam, M. Rashid, F. Pasandideh, B. Ray, S. A. Moore, and R. Kadel, "A Review of Applications and Communication Technologies for Internet of Things (IoT) and Unmanned Aerial Vehicle (UAV) Based Sustainable Smart Farming," *Sustainability*, vol. 13, no. 4, p. 1821, Feb. 2021, doi: 10.3390/su13041821
- [28] J. Martinez-Caro and M. Cano, "IoT System Integrating Unmanned Aerial Vehicles and LoRa Technology: A Performance Evaluation Study," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–12, Jan. 2019, doi: 10.1155/2019/4307925
- [29] W. D. Paredes, H. Kaushal, I. Vakilinia, and Z. Prodanoff, "LoRa Technology in Flying Ad Hoc Networks: A Survey of Challenges and Open Issues," *Sensors*, vol. 23, no. 5, p. 2403, Mar. 2023, doi: 10.3390/s23052403.
- [30] Z. Yuan, J. Jin, L. Sun, K.-W. Chin, and G.-M. Muntean, "Ultra-Reliable IoT Communications with UAVs: A Swarm Use Case," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 90–96, Dec. 2018, doi: 10.1109/mcom.2018.1800161.
- [31] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 2017, pp. 1–7, doi: 10.1109/SysEng.2017.8088251.
- [32] "What is MQTT? Definition and Details." <https://www.paessler.com/it-explained/mqtt>
- [33] "What is MQTT? - MQTT Protocol Explained - AWS," Amazon Web Services, Inc. <https://aws.amazon.com/what-is/mqtt/>
- [34] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 2017, pp. 1–7, doi: 10.1109/SysEng.2017.8088251.
- [35] "IBM Documentation." <https://www.ibm.com/docs/en/ibm-mq/8.0?topic=concepts-quality-service>

- [36] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," in *IEEE Access*, vol. 8, pp. 201071-201086, 2020, doi: 10.1109/ACCESS.2020.3035849.
- [37] D. V. Silva, L. S. Carvalho, J. M. S. Júnior, and R. C. Sofia, "A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA," *Appl. Sci.*, vol. 11, no. 11, p. 4879, May 2021, doi: 10.3390/app11114879.
- [38] Wallarm, "What is CoAP Protocol Meaning Architecture," Wallarm, Mar. 27, 2023. <https://www.wallarm.com/what/coap-protocol-definition>
- [39] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, doi: 10.1109/syseng.2017.8088251.
- [40] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017. doi: 10.1109/syseng.2017.8088251.
- [41] C. B. Gemirter, C. Senturca, and S. Baydere, A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data. 2021. doi: 10.1109/ubmk52708.2021.9559032.
- [42] N. Q. Uy and V. H. Nam, A comparison of AMQP and MQTT protocols for Internet of Things. 2019. doi: 10.1109/nics48868.2019.9023812.
- [43] Y. Jiang, K. Liu, T. Huang, C. Kang, and Y. Yang, "Prototype Data System of Highly Reliable Ship Message Queue based on AMQP," 2022. doi: 10.1109/bigcom57025.2022.00053.
- [44] "ChirpStack open-source LoRaWAN Network Server." <https://www.chirpstack.io/>
- [45] "ChirpStack architecture - ChirpStack open-source LoRaWAN® Network Server." <https://www.chirpstack.io/project/>
- [46] "Features - ChirpStack open-source LoRaWAN® Network Server." <https://www.chirpstack.io/>
- [47] The Things Network, "Architecture," 2021. [Online]. Available: <https://www.thethingsnetwork.org/>.
- [48] M. Aazam and K. A. Harras, "Deployment and management of IoT services using edge and cloud computing: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3128-3166, 2019.
- [49] P. Goncalves, M. P. Coutinho, E. Monteiro, and J. A. Afonso, "IoT-based smart agriculture monitoring system using Microsoft Azure," in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2019, pp. 1-6.

- [50] I. Yaqoob, E. Ahmed, M. H. ur Rehman, A. I. A. Ahmed, M. A. Al-Garadi, M. Imran, and S. Guizani, "The rise of big data on cloud computing: Review and open research issues," *IEEE Access*, vol. 8, pp. 619-639, 2020.
- [51] Microsoft, "Azure IoT - Internet of Things (IoT) Solutions," 2021. [Online]. Available: <https://azure.microsoft.com/en-us/overview/iot/>.
- [52] Microsoft, "Azure IoT Central," 2021. [Online]. Available: <https://azure.microsoft.com/en-us/services/iot-central/>.
- [53] J. Arshad et al., "Implementation of a LoRaWAN Based Smart Agriculture Decision Support System for Optimum Crop Yield," *Sustainability*, vol. 14, no. 2, p. 827, Jan. 2022, doi: 10.3390/su14020827.
- [54] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017. doi: 10.1109/syseng.2017.8088251.
- [55] "Azure IoT Central - IoT Solution Development | Microsoft Azure." <https://azure.microsoft.com/en-us/products/iot-central/>
- [56] Len Calderone. "Pervasive Automation in Agriculture." *AgriTechTomorrow*, 12 May 2020. [Online]. Available: <https://www.agritechtomorrow.com/article/2020/05/pervasive-automation-in-agriculture/12146>.
- [57] How Automation Is Transforming Agricultural Farming." *Electricsolenoidvalves.com*, 23 Nov. 2022. [Online]. Available: <https://www.electricsolenoidvalves.com/blog/how-automation-is-transforming-agricultural-farming/>.
- [58] "The State of Food and Agriculture 2022: Leveraging automation to transform agrifood systems." *FAO*, 2 Nov. 2022. [Online]. Available: <https://www.fao.org/newsroom/detail/FAO-state-of-food-and-agriculture-SOFA-2022-automation-agrifood-systems/en>.

A Installation Guide for LWN Simulator

This appendix provides step-by-step instructions to install the LWN Simulator on a Linux system.

1. **Create a directory** for the LWN Simulator. For example, in your home directory:

```
mkdir ~/lwn_simulator
cd ~/lwn_simulator
```

2. **Download and extract the Go programming language** to the same directory:

```
sudo wget https://dl.google.com/go/go1.17.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.17.linux-amd64.tar.gz
```

3. **Set the GOROOT and GOPATH environment variables** to the directory where Go is installed and the directory where you want to store your Go projects. Add the following lines to the end of your `/.bashrc` file:

```
export GOROOT=/usr/local/go
export GOPATH=$HOME/lwn_simulator
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

Then, reload your `/.bashrc` file:

```
source ~/.bashrc
```

4. **Install the LWN Simulator dependencies:**

```
sudo apt update
sudo apt install make git
cd ~/lwn_simulator
git clone https://github.com/UniCT-ARSLab/LWN-Simulator.git
cd LWN-Simulator
go install github.com/rakyll/statik@latest
make install-dep
```

5. **Build and run the simulator:**

```
make build
make run-release
```

6. **Set up the LWN Simulator as a systemd service:** To ensure that the LWN Simulator is always running, even after a system reboot or a crash, you can set it up as a systemd service. Open a new systemd service file for editing:

```
sudo nano /etc/systemd/system/lwn-simulator.service
```

Then, insert the following content into the file:

```
[Unit]
Description=LWN Simulator
After=network.target

[Service]
Type=simple
Restart=always
RestartSec=5s
User=azureuser
WorkingDirectory=/home/azureuser/lwn_simulator/LWN-Simulator
ExecStart=/usr/bin/make run-release

[Install]
WantedBy=multi-user.target
```

Save the file and exit the editor. Then, reload the systemd configuration:

```
sudo systemctl daemon-reload
```

Now, enable and start the LWN Simulator service:

```
sudo systemctl enable lwn-simulator.service
sudo systemctl start lwn-simulator.service
```

You can check the status of the service by using the following command:

```
sudo systemctl status lwn-simulator.service
```

And if needed, you can stop or restart the service with these commands:

```
sudo systemctl stop lwn-simulator \begin{verbatim}
sudo systemctl restart lwn-simulator.service
```


B ChirpStack Installation

This appendix provides step-by-step instructions on how to install ChirpStack on Ubuntu. ChirpStack is an open-source LoRaWAN Network Server stack that can be used to manage a LoRaWAN network, including the network-server, application-server, and gateway-bridge components.

1. Update your Ubuntu packages:

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

2. Install Git:

```
sudo apt install git
```

3. Clone the ChirpStack on Ubuntu repository:

```
git clone https://github.com/RAKWireless/ChirpStack_on_Ubuntu
```

4. Navigate to the cloned directory and install ChirpStack:

```
cd ChirpStack_on_Ubuntu
sudo ./install.sh
```

5. Check the status of the ChirpStack network and application servers:

```
journalctl -u chirpstack-network-server -f -n 50
journalctl -u chirpstack-application-server -f -n 50
```

6. Add the LoRaServer repository to your sources list:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
1CE2AFD36DBCCA00

sudo echo "deb https://artifacts.loraserver.io/packages/3.x/deb
stable main" | sudo tee /etc/apt/sources.list.d/loraserver.list

sudo apt-get update
```

7. Install the LoRa Gateway Bridge and start it:

```
sudo apt-get install lora-gateway-bridge
sudo systemctl start lora-gateway-bridge
journalctl -u lora-gateway-bridge -f -n 50
```

8. To change the configuration of the ChirpStack network and application servers, navigate to the configuration directory and open the corresponding configuration files:

```
cd ChirpStack_on_Ubuntu/chirpstack-network-server_conf/
nano chirpstack-network-server.toml
nano chirpstack-application-server.toml
```

9. Restart the ChirpStack application server to apply the changes:

```
sudo systemctl restart chirpstack-application-server
```

10. To check the status of the ChirpStack network server, use the following command:

```
sudo journalctl -u chirpstack-network-server -f
```

C Installation of MQTT Broker and Node-RED using Docker

This appendix provides a step-by-step guide to installing the MQTT broker and Node-RED using Docker. We used the Eclipse Mosquitto MQTT broker and the Node-RED programming tool in our system. Both applications were packaged into Docker containers for easy deployment and management.

1. Update the package list for the Ubuntu operating system:

```
sudo apt-get update
```

2. Install the Docker package:

```
sudo apt-get install -y docker.io
```

3. Start the Docker service:

```
sudo systemctl start docker
```

4. Enable Docker to start on boot:

```
sudo systemctl enable docker
```

5. Install Docker Compose:

```
sudo apt-get install -y docker-compose
```

6. Create a new directory to store the Docker Compose configuration and move into it:

```
mkdir nodered-mqtt  
cd nodered-mqtt
```

7. Create a new directory to store the Mosquitto configuration file and move into it:

```
mkdir mosquitto-config  
cd mosquitto-config
```

8. Create a new mosquitto.conf file using your preferred text editor:

```
nano mosquitto.conf
```

9. Add the following lines to the mosquitto.conf:

```
listener 1883
allow_anonymous true
```

10. Create a new docker-compose.yml file in the nodered-mqtt directory:

```
nano docker-compose.yml
```

11. Add the following lines to the docker-compose.yml file:

```
version: "3"
services:
  nodered:
    image: nodered/node-red
    restart: unless-stopped
    ports:
      - "1880:1880"
    volumes:
      - ./nodered-data:/data
  mosquitto:
    image: eclipse-mosquitto
    restart: unless-stopped
    ports:
      - "1883:1883"
    volumes:
      - ./mosquitto-config:/mosquitto/config
    command: mosquitto -c /mosquitto/config/mosquitto.conf
```

12. Start the NodeRED and MQTT containers by running the following command:

```
sudo docker-compose up -d
```

13. Verify that the containers are running:

```
sudo docker ps
```

14. Change the ownership of the mounted volume to the current user, so that Node-RED can work:

```
sudo chown -R $USER:$USER ~/nodered-mqtt/nodered-data
```

15. Restart the Docker Compose:

```
sudo docker-compose down  
sudo docker-compose up -d
```

Now, the MQTT broker and Node-RED are installed and running in Docker containers. You can access the Node-RED editor through your browser by typing "localhost:1880" in the address bar. As for the MQTT broker, it is set up to listen on port 1883.