Implementation and Evaluation of a Relay System for LoRaWAN-based IoT Networks

LEONOR SOFIA RODRIGUES LOBO
(Licenciada)

Dissertação para obtenção do grau de mestre em Engenharia de Eletrónica e Telecomunicações, no Perfil de Telecomunicações

Orientadores:
Especialista Nuno António Fraga Juliano Cota
Doutor Nuno Miguel Machado Cruz

Júri:
Presidente: Doutor Rui António Policarpo Duarte
Vogais:
Doutor António João Nunes Serrador
Especialista Nuno António Fraga Juliano Cota

Outubro de 2025

Implementation and Evaluation of a Relay System for LoRaWAN-based IoT Networks

LEONOR SOFIA RODRIGUES LOBO
(Licenciada)

Dissertação para obtenção do grau de mestre em Engenharia de Eletrónica e Telecomunicações, no Perfil de Telecomunicações

Orientadores:
Especialista Nuno António Fraga Juliano Cota, ISEL
Doutor Nuno Miguel Machado Cruz, ISEL

Júri:
Presidente: Doutor Rui António Policarpo Duarte, ISEL

Vogais:
Doutor António João Nunes Serrador, ISEL
Especialista Nuno António Fraga Juliano Cota, ISEL

Outubro de 2025

Acknowledgments

I would like to express my sincere gratitude to my girlfriend, Margarida, for her constant presence and unconditional support throughout this journey. To my parents, Luís and Olga, I am deeply thankful for their patience and understanding during the development of this thesis, as well as for their help with the tests and their encouragement at key moments. To my sister, Inês, a special appreciation for her psychological support and her always comforting presence.

To my supervisors, Professors Nuno Cruz and Nuno Cota, I am grateful for believing in me, for their dedicated guidance, and for all the support provided over the past year. I would also like to thank Solvit for providing the hardware and facilities necessary to carry out this work. Finally, I thank ISEL, the institution where I completed my entire academic path, from the bachelor's degree to the master's.

To all, thank you!

Statement of integrity

I declare that this project work is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author

Assinado por: **Leonor Sofia Rodrigues Lobo**
Num. de Identificação: 14949130
Data: 2025.10.22 18:00:36+01'00'

Lisbon, 22 October, 2025

Abstract

The rapid expansion of IoT has revolutionized various industries, yet its growth remains limited in remote areas where traditional network infrastructures, such as mobile networks and satellites, are either unreliable or too costly to implement. This thesis explores the use of LoRaWAN as a promising solution for low-power, long-range communication in remote locations. Despite its advantages, LoRaWAN faces challenges in areas with weak backhaul infrastructure or long-distance communication needs. To address these issues, this work proposes and implements a relay system composed of two devices: the C-Mesh, which acts as a gateway and listens continuously to end-devices messages and the C-Point, which encapsulates and forwards those uplinks. A dedicated communication protocol between C-Mesh and C-Point was developed, including a mechanism to update OTAA session keys over BLE. The potential of LoRa Mesh and satellite networks is also explored, along with the study of existing relay systems, such as the TS011-1.0.0 specification, that provide useful design insights. Field tests showed that the proposed relay increased delivery from 12.0% to 47.3% and raised the message receptions with SNR > 0 dB from 16.2% to 42.1%, confirming its role in extending LoRaWAN coverage and improving signal quality.

Keywords

IoT, LoRaWAN, LPWAN, Network Coverage Extension, Relay System, Edge Computing

Resumo

A rápida expansão da IoT tem vindo a revolucionar vários sectores, mas o seu crescimento permanece limitado em áreas remotas, onde as infraestruturas de rede tradicionais, como redes móveis ou satélites, são pouco fiáveis ou demasiado dispendiosas de implementar. Esta tese explora o uso de LoRaWAN como uma solução promissora para comunicações de baixo consumo e longo alcance em locais remotos. Apesar das suas vantagens, o LoRaWAN enfrenta desafios em zonas com fraca infraestrutura de backhaul ou com necessidades de comunicação a longas distâncias. Para responder a estes desafios, este trabalho propõe e implementa um sistema de repetição composto por dois dispositivos: o C-Mesh, que actua como gateway e escuta continuamente as mensagens dos dispositivos finais, e o C-Point, que encapsula e reencaminha essas mensagens. Foi ainda desenvolvido um protocolo de comunicação dedicado entre o C-Mesh e o C-Point, incluindo um mecanismo de actualização das chaves OTAA através de BLE. É também explorado o potencial das redes LoRa Mesh e das redes por satélite, bem como o estudo de sistemas de repetição existentes, como a especificação TS011-1.0.0, que fornecem contributos relevantes para o desenho da solução. Os testes de campo demonstraram que o repetidor proposto aumentou a taxa de entrega de 12,0% para 47,3% e elevou as receções com SNR > 0 dB de 16,2% para 42,1%, confirmando o seu papel na extensão da cobertura LoRaWAN e na melhoria da qualidade do sinal.

Palavras-chave

IoT, LoRaWAN, LPWAN, Extensão de Cobertura de Rede, Sistema de Repetição, Computação na Periferia

Contents

List of Figures

List of Tables

Acronyms

3GPP	3rd Generation Partnership Project
ABP	Activation By Personalization
AS	Application Server
BLE	Bluetooth Low Energy
BPSK	Binary Phase Shift Keying
Bw	Bandwidth
CAD	Channel Activity Detection
CR	Coding Rate
CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
CUPS	Configuration and Update Server
DtS-IoT	Direct-to-Satellite Internet of Things
GEO	Geostationary Orbit
gRPC	Remote Procedure Call
HAL	Hardware Abstraction Layer
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
ISM	Industrial, Scientific, and Medical Band
ItS-IoT	Indirect-to-Satellite Internet of Things
IUU	Illegal, unreported, and unregulated fishing
LEO	Low Earth Orbit
LNS	LoRaWAN Network Server
LoRa	Long Range

LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Networks
LR-FHSS	Long Range Frequency Hopping Spread Spectrum
LTE	Long-Term Evolution
MAC	Medium Access Control
MIC	Message Integrity Code
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrowband Internet of Things
NS	Network Server
OFDM	Orthogonal Frequency Division Multiplexing
OTAA	Over-the-Air Activation
PDR	Packet Delivery Ratio
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
ToA	Time On Air
TTN	The Things Network
TTS	The Things Stack
UNB	Ultra Narrowband
VM	Virtual Machine
VMS	Vessel Monitoring Systems
WOR	Wake On Radio

Chapter 1

Introduction

The Internet of Things (IoT) has seen significant growth in recent years, transforming industries like agriculture, healthcare, smart cities, and manufacturing. While many IoT applications thrive in well-connected urban environments, there is a growing need to extend IoT networks to remote locations where traditional network infrastructure is either unreliable or unavailable. These areas such as offshore locations, mountainous regions, or isolated rural settings often lack mobile network coverage, which presents a major challenge for IoT deployment.

In some cases, satellite communication can be used to connect IoT devices in remote locations. However, satellites are often impractical for small-scale IoT operations due to their high operational expenses and significant energy requirements [65], making them unsuitable for the low-power devices that are typical in IoT ecosystems. Traditional mobile networks are effective in urban and suburban environments but struggle to provide consistent coverage in geographically isolated areas. Expanding mobile or satellite infrastructure to cover these regions is expensive and energy-intensive, which makes these technologies less appealing for IoT applications that prioritize cost efficiency and energy savings. As a result, the dependence on mobile networks or satellite connections limits the potential of IoT in areas without reliable coverage.

In this context, LoRaWAN (Long Range Wide Area Network) has emerged as a promising solution for enabling low-power, long-distance communication. LoRaWAN allows IoT devices to transmit data over large distances using minimal energy, making it well-suited for remote areas where energy-efficient solutions are crucial. In maritime scenarios, links beyond 130 km have been demonstrated when a high-altitude gateway is available [66] but with typical coastal gateway heights, coverage drops off at roughly 40 km offshore, which is insufficient for fishing operations that routinely reach 60–80 km from shore.

In a typical LoRaWAN setup, the gateway collects data from IoT devices and forwards it to a central network server. This communication between the gateway and the network server generally happens through cellular mobile networks, Wi-Fi or Ethernet. In areas where these backhaul infrastructures are weak or nonexistent, the gateway's ability to relay data to the network server can be severely interrupted. Additionally, when IoT devices are located beyond the

effective range of the gateway or face physical obstructions, communication becomes unreliable. The work in [71] demonstrates that in dense urban environments, increasing the spreading factor alone may not be sufficient to overcome coverage issues, indicating the need for more system-level solutions. Another common approach to mitigate these limitations is to deploy additional gateways to extend coverage and improve overall network reach, but this increases infrastructure costs and may not be practical in remote or energy-constrained areas.

A relay system becomes critical in these scenarios to extend the effective communication range of LoRaWAN networks. By acting as an intermediary between the IoT end devices and the gateway, the relay ensures that devices in remote or obstructed locations can still transmit their data. Furthermore, integrating edge computing capabilities into the relay allows for localized data processing, reducing the need for constant back-and-forth communication with the network server. This not only improves network efficiency by offloading processing tasks to the edge but also improves reliability and reduces latency, especially in managing tasks such as secure key synchronization and repeated message handling.

This work investigates the design and implementation of a LoRaWAN relay system with edge computing, specifically targeting remote environments where existing LoRaWAN coverage is insufficient.

1.1 Motivation

The motivation for this work goes beyond the technical limitations of existing IoT connectivity solutions. Extending LoRaWAN coverage is not only a technological challenge but also a necessity for addressing pressing society and environmental problems. Reliable IoT communication in remote areas can enable new classes of applications with direct impact on sustainability, resource management, and safety.

In the maritime sector, limited connectivity has severe consequences. Current Vessel Monitoring Systems (VMS) are mandatory only for large vessels, leaving most smaller fishing boats untracked and outside regulatory oversight. This gap undermines sustainable fisheries management and creates difficulties in fighting illegal, unreported, and unregulated (IUU) fishing [70]. Furthermore, the widespread loss of fishing gear at sea contributes to marine pollution and ghost fishing, with long-lasting damage to marine ecosystems [69]. Affordable and energy-efficient IoT solutions that extend connectivity offshore are therefore essential to enable continuous monitoring of vessels and equipment, supporting both environmental protection and compliance enforcement.

Similarly, in forestry and environmental monitoring, faces similar challenges. Forest ecosystems, vital for biodiversity and climate regulation, are increasingly threatened by wildfires intensified by climate change. Studies such as [67] and [68] demonstrate how IoT-based monitoring systems can improve early detection and response to these situations. Extending LoRaWAN coverage in these contexts is vital to ensure that sensor networks can provide timely and reliable data to support decision-making in disaster prevention and environmental management.

Despite its advantages, LoRaWAN still presents several limitations in remote environments. Its coverage, although long-range compared to traditional wireless systems, depends strongly on gateway placement, antenna height, and line-of-sight conditions, which can be affected by terrain, vegetation, or sea surface curvature [66]. The architecture also relies on a central network server, creating a dependency on the gateway's backhaul connection and if this link is unavailable (the lack of cellular or Ethernet connectivity), the transmitted data cannot reach the application layer. Together, these factors make it difficult to maintain reliable communication over extended areas such as open sea or dense forests.

A practical demonstration of these limitations was provided by the Custodian Project, where monitoring fishing activities and equipment offshore required connectivity beyond traditional coastal LoRaWAN coverage. The project revealed the high cost of satellite services for small-scale operators and the incompatibility of the new LoRaWAN relay standard with already deployed devices, highlighting the need for an intermediate solution that is cost-effective, interoperable, and capable of extending LoRaWAN without protocol modifications.

These insights motivated the development of a custom relay system with integrated edge computing. Unlike standard relays, the proposed system not only forwards messages but also performs local processing, secure key management, and duplicate message filtering. This enables a scalable and flexible architecture that extends LoRaWAN into environments that were previously unreachable. By doing so, the system helps mitigate climate change, reduces environmental impact and promotes sustainable resource management. Ultimately, the approach ensures that connectivity is no longer a barrier to innovation and sustainability in remote regions.

1.2 Research Question and Hypothesis

To guide the work presented in this thesis, a central research question and a corresponding hypothesis were formulated. The research question establishes the problem to be investigated, while the hypothesis defines the expected outcome of the proposed approach.

Research Question (RQ): Can the integration of a relay system with edge computing capabilities effectively extend the coverage of LoRaWAN networks in remote environments, while ensuring compatibility with existing devices and preserving security and performance? Additionally, can this approach improve signal quality in challenging communication scenarios?

Hypothesis (H): The proposed relay system with integrated edge computing will extend LoRaWAN coverage by at least 30% compared to a deployment without relays, while improving the packet delivery ratio by at least 20% under weak-signal conditions (RSSI < –115 dBm or SNR < -5 dB). Furthermore, the system is expected to provide measurable improvements in signal quality, reflected in higher average SNR and RSSI values, while maintaining compatibility with existing devices and ensuring secure key synchronization and efficient duplicate message management.

1.3 Objectives

The main focus of this thesis is the development of a relay system for LoRaWAN networks. Unlike a simple transparent repeater that simply forwards messages, this relay is designed to perform additional processing and coordination tasks. To achieve this, the system requires a LoRa concentrator chip (Semtech SX1302/SX1303) capable of multi-channel, multi-data rate LoRa packet reception to collect LoRaWAN traffic, and a separate LoRa transceiver chip (SX1276 /SX1278) dedicated to forwarding messages. A central processing unit with sufficient RAM and storage is also required to coordinate the LoRa components, manage communication protocols and the software stack, handle processing tasks, and execute the relay logic. The relay must be compatible with existing devices already on the market such as sensors, gateways, and other LoRaWAN-enabled devices and support both ABP and OTAA activation methods. The following specific goals were defined to guide the development and assessment of the proposed system:

- Study the state of the art in relay-based systems for LoRaWAN and analyze the trade-offs of the technologies used when building an IoT system.
- Implement and evaluate a relay-based system for IoT networks using the LoRaWAN protocol.
- Ensure compatibility with devices already on the market and with both activation methods (ABP and OTAA) in LoRaWAN.
- Specify and implement a dedicated message protocol to support relay operation, ensuring 100% compatibility with existing LoRaWAN payload structure.
- Develop and validate a key synchronization mechanism between the relay node and the central platform.
- Evaluate the system in both controlled laboratory and real-world field conditions by measuring:
  - Coverage improvement: ≥30% increase in communication range compared to direct device-to-gateway links.
  - Packet delivery ratio (PDR): ≥20% improvement in weak-signal conditions (RSSI < –115 dBm or SNR < -5 dB).
- Demonstrate scalability by supporting over 16 end-devices per relay node.
- Quantify the overall network improvement, providing statistical analysis of coverage, PDR and signal quality before and after relay deployment.

1.4 Report Layout

The report is organized into seven main chapters, followed by several appendices.

- **Chapter 2 – Theoretical Background:** Provides the necessary background for the study, starting with LPWAN technologies (LoRa, NB-IoT, and Sigfox) and their comparison. It then introduces LoRaWAN network architecture, including its network topology, device classes and activation methods.
- **Chapter 3 – Related Work:** Reviews existing work and state-of-the-art solutions relevant to this study. It addresses relay systems, the integration of LoRaWAN with mesh networking, and the role of satellite networks (including LR-FHSS). The chapter concludes by identifying current research and technological gaps.
- **Chapter 4 – System Architecture:** Describes the proposed system, starting with an architectural overview. It details the hardware selected for the implementation, and explains the communication flows, including message formats and processing sequences.
- **Chapter 5 – System Development:** Focuses on the implementation. It covers the software setup of the relay node and the supporting virtual machine, followed by the implementation details of the relay software and VM software components.
- **Chapter 6 – Evaluation and Results:** Defines the evaluation objectives and performance metrics. It presents results from controlled tests as well as field trials, analyzing the performance of the relay system in real-world conditions.
- **Chapter 7 – Conclusions:** Summarizes the main findings of the work, highlights its contributions, and suggests directions for future improvements and research.

1.5 Contributions

This thesis contributed to the submission and publication of the following paper:

1. Cruz, N.; Mendes, C.; Cota, N.; Esteves, G.; Pinelo, J.; Casaleiro, J.; Teixeira, R.; Lobo, L. ”Extending LoRaWAN: Mesh Architecture and Performance Analysis for Long-Range IoT Connectivity in Maritime Environments. Systems 2025, 13, 381. https://doi.org/10.3390/systems13050381

A second scientific article is currently under development to present the extended results and advancements achieved throughout this thesis work.

Chapter 2

Theoretical Background

This chapter provides a review of the current technologies and methodologies relevant to extending communication coverage in LoRaWAN-based IoT networks. It begins with an overview of LPWAN technologies, including LoRa, NB-IoT, and Sigfox, followed by a comparative analysis of their characteristics. The chapter then examines LoRaWAN in detail, followed by a description of its network architecture, highlighting the functions of gateways, packet forwarders, network servers, and the application layer.

2.1 LPWAN Networks

Commonly used short-range radio technologies, such as ZigBee and Bluetooth, are not suited for scenarios that demand long-range transmission. While cellular communication technologies, including 2G, 3G, and 4G, can provide broader coverage, they come with the drawback of high energy consumption, making them unsuitable for battery-powered IoT devices. As a result, the growing demands of IoT applications have driven the development of LPWAN technologies. These networks are designed to offer long-range connectivity with minimal power consumption, making them ideal for IoT devices deployed in remote or large-scale environments where traditional communication methods fall short [1].

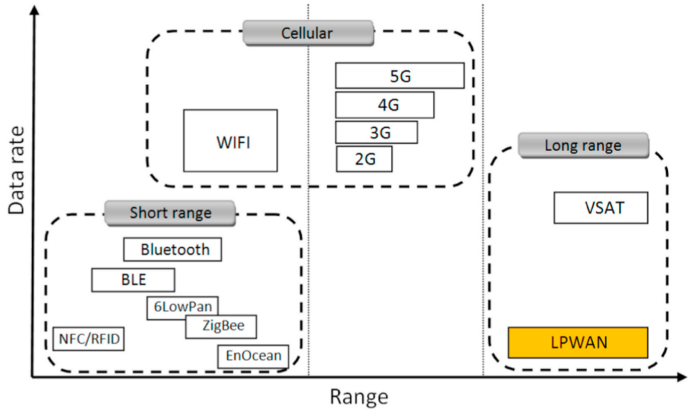


Figure 2.1 Data Rate and Range Capabilities of Radio Communication Technologies [1].

Low Power Wide Area Networks (LPWAN) refer to wireless communication technologies optimized for long-range, low-power, and low-bandwidth connectivity as shown in figure 2.1. LPWANs have been developed to support Internet of Things (IoT) applications where devices need to operate in remote or challenging environments and maintain extended battery life, often lasting years on a single charge.

2.1.1 LoRa

The term LoRa refers to a proprietary radio frequency technology patented by *Semtech*, designed for low-power, long-range network applications (LPWAN). This technology is suitable for solutions requiring extensive coverage areas (in the order of kilometers) and, simultaneously, significant battery life for end devices (lasting several years).

LoRa Physical Layer

In LoRa, the modulation technique used involves a spread spectrum method where the signal frequency is linearly altered over the signal duration. This signal is referred to as a Chirp, and the modulation as CSS. The technique was initially developed for radar and sonar applications and provides robust performance in noisy radio channels and immunity to Doppler effects [3, 4].

Semtech adapted this technique by applying such signals to a relatively large bandwidth (125, 250, or 500 kHz) compared to the achievable data rates (250 bps to 50 kbps). At the same time, the efficiency of the modulation and error correction schemes supports low SNR values. The SNR represents the ratio of signal power to noise power within the channel. Lower SNR values indicate that the signal power is weaker relative to the noise level [2, 3].

The channel bandwidth Bw corresponds to the range of frequencies used for transmitting data. In LoRa, channels can have bandwidths of 125, 250, or 500 kHz. A central frequency, f_c, is associated with the channel's center frequency. In Europe, the operating frequency band is 868 MHz, while in the United States, it is 915 MHz. Different frequencies are used in other regions of the world.

A symbol is a segment of the physical signal where the data is encoded and corresponds to a Chirp that sweeps between two frequencies: a minimum (f_{min}) and a maximum (f_{max}). These frequencies are determined by the Bw and f_c, as follows:

$$f_{\text{min}} = f_c - \frac{Bw}{2} \qquad [\text{H}_z] \tag{2.1}$$

$$f_{\text{max}} = f_c + \frac{Bw}{2} \qquad [\text{H}_z] \tag{2.2}$$

A Chirp will vary linearly and cyclically between the minimum and maximum frequencies over *Ts* seconds. This duration, called the symbol period (*Ts*), is defined as a function of the bandwidth (Bw) and the spreading factor (SF) by the following formula [3]:

$$Ts = \frac{2^{SF}}{Bw} \qquad [\mathrm{s}] \tag{2.3}$$

Finally, the spreading factor (SF) determines how many bits of information can be encoded per symbol. In LoRa, each symbol can encode between 6 and 12 bits. For instance, an SF of 8 indicates that a LoRa symbol encodes 8 bits. The SF also determines the symbol duration, as per the previous formula, where different SFs result in different values for *Ts* [3].

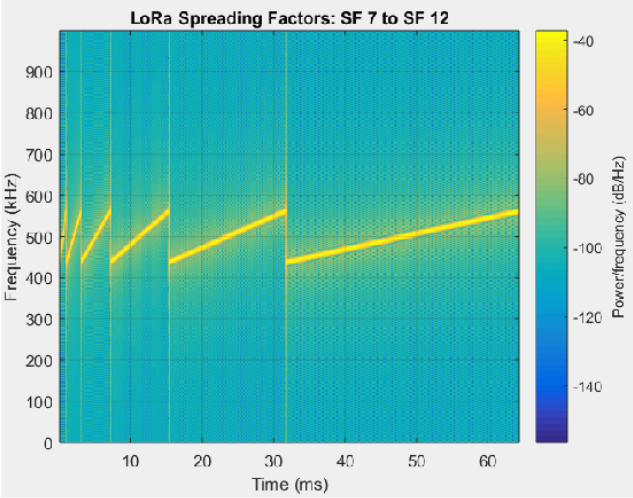


Figure 2.2 Simulation of LoRa modulation with different SF values [5].

Figure 2.2 illustrates the relationship between the spreading factor (SF) and the time required to transmit a single symbol, assuming a bandwidth of 125 kHz. As the spreading factor increases from SF7 to SF12, each symbol is encoded with more chirps, which improves receiver sensitivity and link robustness but also increases the symbol duration—and therefore the total time on air of each transmission. In other words, higher SF values improve communication range and reliability at the cost of longer airtime and reduced data rate.

LoRa implements forward error correction (FEC) by adding redundant bits to the transmitted data, allowing receivers to detect and correct errors introduced by interference (e.g., flipping a 0 to a 1 or vice versa). The strength of this error correction is defined by the coding rate (CR), which expresses the proportion of useful information bits relative to the total transmitted bits. The available CR values in LoRa are 4/5, 4/6, 4/7, and 4/8. For example, with a CR of 4/7, for every 4 bits of useful data, 7 bits are transmitted, including 3 extra bits for error correction. This redundancy enables the recovery of information even when part of the transmission is corrupted.

LoRa technology can be integrated with various protocols, each offering distinct architectures. For instance, LoRaWAN employs a star topology, while LoRa Mesh uses a decentralized, mesh-based architecture. A detailed explanation of LoRaWAN and LoRa Mesh is presented in Sections 2.2 and 3.2, respectively.

2.1.2 NB-IoT

Introduced by the 3GPP in 2016, NB-IoT is a cellular standard based on LTE that operates in licensed spectrum and relies on existing mobile infrastructure. It was designed to provide extended coverage (≈20 dB more than LTE), massive device connectivity, and low power consumption, with devices lasting up to 10 years on a single battery [7]. NB-IoT can be deployed in standalone (refarmed GSM spectrum), in-band (within the LTE carrier), or guard-band (adjacent to LTE carriers) modes. Its architecture builds on the LTE Enhanced Packet Core (EPC) and introduces the Service Capability Exposure Function (SCEF), enabling lightweight non-IP data transfer directly to application servers and thereby reducing device complexity and energy use [9]. Simplified modulation schemes (BPSK, QPSK), single-antenna operation, and the lack of support for some LTE features such as handovers, roaming, and SMS further reduce complexity [8]. While widely supported by operators worldwide, NB-IoT depends on licensed spectrum and subscriptions, which makes it less attractive for individual or low-cost applications.

2.1.3 Sigfox

Sigfox, founded in 2009 by a French company, is a LPWAN technology designed for Internet of Things (IoT) and Machine-to-Machine (M2M) applications. It operates in unlicensed ISM frequency bands, such as 868-869 MHz in Europe, 902-905 MHz in North America, and 922-923 MHz in Japan and South Korea [9]. Sigfox is a proprietary LPWAN technology that operates on a centralized model, where local operators deploy and manage base stations connected to the Sigfox Cloud. Its architecture consists of end devices, gateways, and the cloud platform, which handles device management and integration through callbacks or a REST API. Sigfox employs ultra-narrowband (UNB) modulation (100–600 Hz channels) with ALOHA random access protocol, enabling long-range, low-power transmissions but limiting payload sizes (typically up to 12 bytes) and data rates (≈100 bps in Europe). Subscription-based pricing models exist for both small and large-scale deployments, but the combination of low throughput, limited downlink, dependence on local operators, and recurring subscription fees makes Sigfox costly and less attractive for applications requiring frequent bi-directional communication or high data volumes [12, 13, 15].

2.1.4 Comparison

Each one of the three IoT technologies has its strengths and weaknesses outlined in table 2.1, making them suitable for different applications. LoRa technology, used for example within the LoRaWAN protocol, is well-suited for public or private, low-power, long-range applications with moderate data rate requirements . NB-IoT is ideal for applications that need higher data rates and large payloads but comes with the downside of operator dependency, which may result in higher costs and limited availability depending on the LTE infrastructure. Sigfox offers very low data rates and limited payload sizes. As a proprietary technology owned and operated by

Sigfox, it also restricts flexibility compared to open standards such as LoRa.

Table 2.1 Comparison of LoRa, NB-IoT, and Sigfox for IoT Applications [16, 17].

Parameter	LoRa	NB-IoT	Sigfox
Radio Technology	CSS	OFDM	UNB
Spectrum [MHz]	868(EU); 915(US)	Licensed LTE bands	868(EU); 915(US)
License	LoRa-Alliance	3GPP	Sigfox
Range	<15km	<15km	<50km
Maximum peak data rate	50kbps	250kbps	100bps (EU)
Maximum payload length	243 B*	1600 B	12 B (UL); 8 B (DL)
Maximum Messages/day	Limited duty cycle*	Unlimited	140 (UL); 4 (DL)
Power Consumption	Very low	Very low	Very low
Adaptive Rate	Yes*	No	No
Network type	Private/public*	LTE	Sigfox
Bidirectional	Yes*	Yes	Limited
Security	AES-128*	SNOW3G	Optional AES-CTR
Re-use existing cellular networks	No	Yes	No

* *Values refer to the LoRaWAN protocol.*

LoRa and Sigfox both operate in unlicensed frequency bands, giving both flexibility in terms of deployment and availability. In contrast, NB-IoT operates within licensed LTE bands, which ensures more reliable communication and better coverage, but also requires access to the licensed spectrum, typically controlled by telecommunication operators. Their data rates differ significantly: LoRa reaches up to 50 kbps, NB-IoT up to 250 kbps, and Sigfox only 100 bps in the EU. Payload sizes follow the same trend, with LoRa modulation when used with LoRaWAN supporting up to 243 bytes, NB-IoT up to 1600 bytes, and Sigfox limited to 12 uplink and 8 downlink bytes restricting its use to simple message exchanges. LoRa, via LoRaWAN, supports both private and public network deployments and NB-IoT operates on operator-based networks via LTE, ensuring reliable coverage but requiring access to licensed cellular infrastructure. Sigfox also operates on operator-based networks, but it uses its own proprietary network infrastructure. Both LoRa (using LoRaWAN) and NB-IoT support bidirectional communication with strong security (AES-128 and SNOW 3G, respectively), whereas Sigfox offers only limited downlink and optional, less robust AES-CTR encryption [16, 17].

For this project, LoRa was chosen for its suitability to meet the project's needs such as long-range communication, low power consumption, scalability, lower operational costs and use of open-standards. Multiple projects use LoRa for their IoT applications. In [53], the au-

thors conducted a study under the Lisbon Smart City initiative to evaluate LPWAN technology for waste management, selecting LoRa and LoRaWAN over Sigfox and NB-IoT. The objective was to replace GSM-based sensors with a more cost-effective and long-range alternative. Experimental results confirmed LoRa's reliability, even in challenging environments such as underground containers, demonstrating resilience to high signal attenuation. The study's success led to the deployment of a city-wide LoRa network, improving waste management and enabling broader smart city applications. This network is now accessible to municipal departments and citizens for various IoT-based initiatives.

2.2 LoRaWAN

LoRa and LoRaWAN are distinct concepts, yet they are often used together due to their complementary roles. LoRa refers to the physical layer, utilizing CSS modulation to enable long-range communication as mentioned in section 2.1.1. In contrast, LoRaWAN is a MAC layer protocol built on top of LoRa created by the LoRaWAN Alliance to support the LoRa physical layer, designed to manage communication and data transmission within LoRa-based networks. LoRaWAN has become a widely adopted protocol for IoT networks, leveraging the capabilities of LoRa to provide low-power, long-distance communication solutions [2].

2.2.1 Network Topology

The network topology used is known as star-of-stars, and it defines several mandatory device types in the architecture, as illustrated in Figure 2.3 [3]:

- *End-device:* Low-power sensors that communicate with gateways using LoRa;
- *Gateways:* Intermediate devices that forward packets between terminals and the network server, also referred to as packet forwarders. The communication between gateways and the network server can use any IP-based high-speed interface, such as WiFi, LTE, or Ethernet;
- *Network Server (NS):* Responsible for the overall network management. Handles tasks such as allowing nodes to join the network, handling duplicate messages received from multiple gateways,performing message decoding and security verification, forwarding data to the relevant application servers, routing application responses or acknowledgments to end nodes via the best gateway, managing the data rates of nodes, and responding to MAC requests from nodes;
- *Application Server (AS):* Process the data received from the network server, being responsible for decoding the messages and may also perform further processing, such as data aggregation, filtering, or transformation, to prepare it for use by applications. Applications are typically hosted in the cloud on a server separate from the NS.

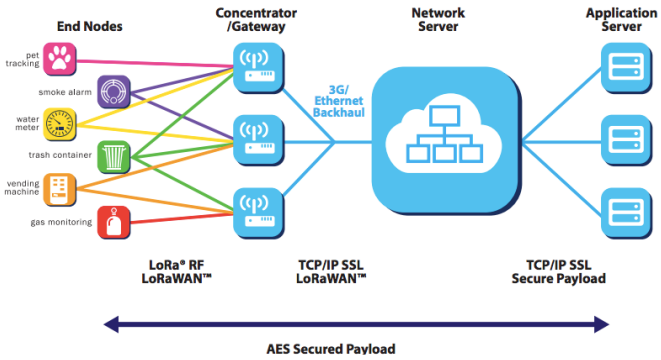


Figure 2.3 LoRaWAN network architecture [5].

Gateways and Packet Forwarders

A packet forwarder is a software component that runs on a LoRaWAN gateway and is responsible for relaying messages between the LoRa radio hardware and the network server. It interfaces with the concentrator chip (e.g., Semtech SX1301, SX1302, or SX1303) through the HAL provided by Semtech, which handles the low-level radio operations. On top of this hardware interface, different packet forwarders implement its own upstream protocol.

The Semtech UDP Packet Forwarder is the original reference implementation developed by Semtech to connect gateways with network servers. It relies on a simple UDP-based protocol, transmitting JSON-formatted packets over UDP sockets. Uplink messages received by the gateway are encapsulated into JSON objects and sent to a predefined server IP and port, while downlink messages follow the reverse path. Despite its simplicity and widespread adoption, this protocol lacks encryption and has no built-in authentication. However, due to its lightweight nature and compatibility with early LoRa hardware, it remains one of the most commonly supported packet forwarders across commercial and open-source gateways [44].

LoRa Basics Station is a newer and more secure packet forwarder developed by Semtech as a replacement for the legacy UDP forwarder. Unlike its predecessor, Basics Station employs the LNS sub protocol, which uses WebSockets for communication and supports TLS encryption and certificate-based authentication, ensuring secure connectivity between gateways and network servers. It also integrates configuration and firmware update mechanisms through the CUPS sub protocol, enabling gateways to be managed remotely without manual intervention. However, LoRa Basics Station is more complex to configure and operate than the legacy UDP forwarder and is still in transition [45].

The ChirpStack Concentratord is an open source daemon designed to manage LoRaWAN gateway hardware by interacting with its concentrators. It provides a ZeroMQ-based API that allows multiple applications to communicate with the gateway hardware simultaneously. By leveraging Semtech's HAL, the Concentratord abstracts hardware details and ensures compatibility with concentrator modules like the SX1301, SX1302, and SX1303, offering scalability and flexibility for LoRaWAN deployments [24].

Network Servers

The Things Network (TTN) is a highly popular community-driven LoRaWAN network that allows anyone to connect gateways and devices free of charge up to a limit of 10 devices and 10 gateways. It provides the core LoRaWAN components: Identity Server (manages users, organizations, and devices), Join Server (authentication and key management), Network Server (message routing and network control), Gateway Server (gateway connectivity), and Application Server (integration with external applications). TTN is a good entry point for prototyping and learning but is not designed for large-scale deployments. While convenient for beginners, it also limits control and customization compared to self-hosted solutions. and While convenient for beginners, it limits control and customization compared to a self-hosted solution. For these scenarios, The Things Industries offers The Things Stack (TTS), a commercial version of the same network server available as managed cloud or enterprise installations with SLA guarantees [18, 19, 21]. The Things Industries also created the Packet Broker, which enables interaction between different LoRaWAN networks, including TTS instances and TTN. By allowing public and private operators to exchange traffic, it facilitates gateway resource sharing and extends coverage across otherwise separate networks [20].

ChirpStack, in contrast, is a fully open-source LoRaWAN network server with a modular architecture, where components such as the Network Server, Application Server, Gateway Bridge, Concentratord, and MQTT Forwarder can be deployed independently, as shown in Figure 2.4. This design provides flexibility and complete control over the infrastructure but requires technical expertise to configure and manage effectively [22].

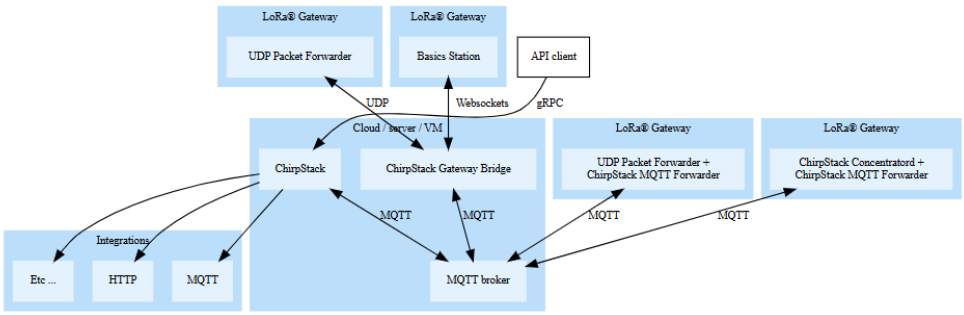


Figure 2.4 Overview of ChirpStack Architecture [22].

The ChirpStack Gateway Bridge is a key component, converting data from various LoRa Packet Forwarder protocols into an MQTT format (JSON or Protobuf) to be processed by the network server. The Gateway Bridge supports multiple packet-forwarder backends, including Semtech UDP Packet Forwarder, Basic Station, and ChirpStack Concentratord (described in section 2.2.1), ensuring compatibility with various gateways. It also offers integration with MQTT brokers and cloud platforms like Google Cloud IoT Core and Azure IoT Hub, enabling data forwarding to cloud services for processing and storage [25, 26]. The ChirpStack MQTT Forwarder is designed for LoRa gateways, enabling packet transmission over the MQTT protocol

(described in section 2.2.1). Unlike the Gateway Bridge, the MQTT Forwarder must always be installed on the gateway. It can be used with either the Semtech UDP Packet Forwarder or the ChirpStack Concentratord, which abstracts hardware details.

Helium is a decentralized wireless network that provides long-range, low-power LoRaWAN connectivity through a blockchain-based model. Users deploy and manage gateways, called Hotspots, which supply coverage and earn Helium tokens (HNT) as rewards. The architecture relies on the Helium Packet Router (HPR), which manages data flow and tracks Hotspot contributions, and the IoT Configuration Service, which allows routing rules to be defined between devices and Network Servers. Helium supports both participation However, while the model initially attracted attention due to its financial incentives, many users have expressed dissatisfaction with the reliance on cryptocurrency. [27].

Application Layer

In the application layer of LoRaWAN, different protocols can be used to deliver data from the network server to external applications. Among the most common are HTTP, gRPC, and MQTT, each offering different trade-offs in terms of simplicity, efficiency, and scalability.

HTTP, being the most widespread web protocol, is widely supported, easy to implement making it a good option when simple integration matters more than efficiency. On the other hand, gRPC provides a more modern and efficient approach that focuses on performance. It uses Protocol Buffers (Protobuf) to encode messages in a compact format and runs over HTTP/2, which enables faster communication and bidirectional streaming. These characteristics make gRPC particularly advantageous in IoT backends where low latency, reduced payload size, and high performance are required [46].

Message Queuing Telemetry Transport, MQTT, is a lightweight messaging protocol widely used in IoT applications especially in environments with limited bandwidth, high latency, or constrained devices. The protocol follows a publish/subscribe (pub/sub) communication model, where devices or applications, known as clients, connect to a central component called the broker as in Figure 2.5. The broker is responsible for receiving messages from publishers and forwarding them to subscribers that have expressed interest in specific topics. A topic is simply a string message that is published and subscribed to (for example, sensors/temperature), allowing publishers and subscribers to exchange information without needing direct knowledge of each other [47].

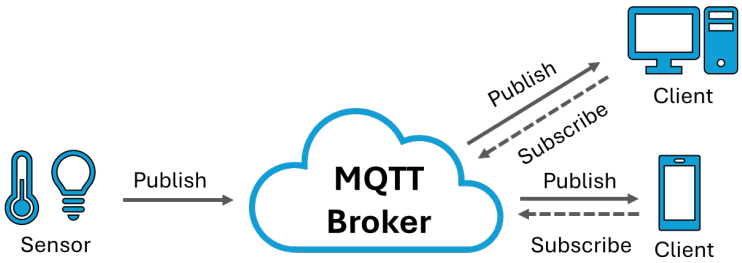


Figure 2.5 Overview of MQTT Architecture.

Several MQTT brokers are widely available. One of the most popular open-source brokers is Eclipse Mosquitto [48], known for its simplicity, small footprint, and reliability, making it an excellent choice for testing, prototyping, and lightweight production environments. On Linux distributions such as Ubuntu, Mosquitto can be installed easily through the package manager. For enterprise-level deployments, HiveMQ [49] provides a highly scalable broker with advanced features such as clustering, monitoring, and integration with cloud platforms. It is designed to handle millions of simultaneous connections, which makes it suitable for large-scale industrial IoT solutions. An important feature of MQTT is its Quality of Service (QoS) levels, which define how reliably messages are delivered. With QoS 0 (at most once), the message is delivered on a best-effort basis, meaning it is sent once and may be lost if the connection fails. This level is the fastest and consumes the fewest resources, but it does not guarantee delivery. With QoS 1 (at least once), the broker ensures that the message reaches the subscriber, but in some cases the message may be delivered more than once, requiring the receiver to handle possible duplicates. Finally, QoS 2 (exactly once) is the most reliable option, guaranteeing that each message is delivered only once, using a more complex handshake process between client and broker [47].

In [46], the results showed that MQTT outperforms HTTP and gRPC in throughput and scalability, making them better suited for IoT telemetry. HTTP is preferable for large file transfers, while gRPC offers efficiency in structured microservice communication.

2.2.2 Device Classes

The LoRaWAN specification classifies devices into three distinct classes (A, B, or C), depending on their energy consumption, with different data transmission and reception profiles.

Class A devices are designed for minimal power consumption, making them ideal for battery-powered applications. Communication is initiated by the end-device with an uplink transmission. Following this uplink, the device opens two short receive windows to listen for potential downlink messages from the network. The first window is mandatory, but the second depends on whether a downlink was received in the first. If so, the device does not need to open the second window. If no downlink is received within these windows, the device enters a sleep state until the next scheduled uplink transmission [2, 3]. This is the most energy-efficient device class

since no medium listening mechanisms are used, and devices remain in sleep mode if no data is to be transmitted. The specification requires all devices to implement this class.

Class B devices offer a compromise between power consumption and latency. Similar to Class A, communication is initiated via uplink transmissions. However, Class B devices also incorporate periodic receive windows, called ”ping slots”, synchronized with the network through beacon transmissions. These ping slots enable the device to listen for downlinks at regular intervals, reducing latency compared to Class A and facilitating more frequent downlink communication [3, 21].

Class C devices prioritize minimal latency, enabling near real-time downlink communication. These devices maintain their receive windows open almost continuously, closing them only briefly during uplink transmissions. However, this continuous listening mode results in higher power consumption, making Class C devices unsuitable for battery-operated deployments [3, 21].

2.2.3 Node Activation

To participate in a LoRaWAN network, an end device must be activated. LoRaWAN offers two methods of activation: OTAA and ABP [3]. During activation, various identifiers and keys are required. These keys are used to encrypt data or create message integrity codes (MIC).

Activation by Personalization is the least secure activation method and has the disadvantage that devices cannot easily switch network providers without manually updating the keys on the device. A join server is not involved in the ABP activation process [3]. The following parameters must be pre-configured on the device:

- **DevAddr**, device address: A 32-bit value that identifies the node in the network. It consists 25 bits that uniquely identify the device within the network and 7 bits that identify the network.
- **NwkSKey**, network session key: A 128-bit key used by the node and the NS to verify the integrity of transmitted messages. It is only used by the node and the NS, where it is used to calculate the MIC.
- **AppSKey**, application session key: A key used to encrypt and decrypt the node’s data (payload), ensuring its confidentiality. It is only used by the node and the AS, guaranteeing end-to-end payload security so no one else can decrypt the data.

Since ABP requires no join procedure, once the keys are configured, nodes can immediately transmit data. While this simplicity can be advantageous in some limited scenarios, it comes at a significant security cost. If a device is compromised, its keys are compromised, potentially affecting the security of the entire network.

OTAA, on the other hand, is the more secure and recommended activation method. In the OTAA method, nodes, NS, and AS must initially be configured with some security keys, which are distinct from those used in ABP and are described below.

- **DevEUI**, device extended unique identifier: A 64-bit identifier, similar to a MAC address in Ethernet networks, is unique to each device and defined during manufacturing;
- **AppEUI**, application extended unique identifier: This code identifies the application responsible for processing the node's data;
- **AppKey**, application key: A 128-bit key used by the application to encrypt the user's data.

When a device wants to join a network, a join process occurs in which the node sends a join-request (encrypted with the AppKey), containing the DevEUI, AppEUI, and a random value, DevNonce, identifying the request [21]. The NS receives the join-request and computes the AppSKey and NwkSKey and must verify if the DevEUI is allowed on the network. If approved, the server responds with a join-accept message containing information such as AppNonce(randomly generated numbe), NetID, and DevAddr. Upon receiving the join-accept, the device decrypts the message using its AppKey [2, 3]. At this point, both the NS and the node have the same information, allowing the node to compute the NwkSKey and AppSKey keys. This dynamic key generation, where each join results in unique session keys, significantly improves network security. It ensures that even if one device's keys are compromised, the security of other devices and the network as a whole remains protected.

2.2.4 Limitations

One of the most significant limitations is the small payload size. LoRaWAN is designed for transmitting small, sporadic data packets. The payload capacity decreases as the data rate decreases, meaning that devices operating with higher spreading factors can send even less data per transmission. Consequently, sending data in formats like JSON or plain ASCII text is inefficient and often impractical. Instead, data must be carefully optimized and encoded into compact binary formats to minimize payload size.

LoRaWAN networks typically operate under a fair use policy. This policy aims to prevent any single device from monopolizing network resources and ensures that all devices have a reasonable opportunity to transmit data. Some networks like TTS have adopted a responsible usage policy on its public network (TTN), allowing a total transmission time of 30 seconds per day and a maximum of 10 downlink messages every 24 hours [21].

Operating in an unlicensed band (863-870 MHz), LoRa technology competes with all existing communications in the same band. However, it allows any entity to deploy and operate its network without additional licensing requirements, which implies limitations on the number of messages transmitted, the maximum power used, and the channel occupancy time (*duty cycle*). The *duty cycle* refers to the percentage of time a device is allowed to transmit. Devices can only transmit for up to 1% of the time within each 1-hour interval (in the EU region). The maximum apparent radiated power for each device is 25 mW, or 14 dBm [3].

With these restrictions in mind, it becomes essential to determine the duration of each transmission, known as ToA, which represents the time the channel is occupied for a given

transmission. The ToA depends on various factors, including the structure of the LoRa physical frame, which follows the format shown in Figure 2.6. This structure consists of four parts: the preamble, header, payload, and payload CRC [3, 21]. The duration of each of these elements contributes to the total transmission time. Although the higher-layer LoRaWAN protocol defines additional MAC fields within the payload, the ToA is determined solely by the physical frame parameters and modulation settings—most notably the selected spreading factor (SF) and bandwidth (Bw). As illustrated in Figure 2.2, higher spreading factors increase the number of chirps per symbol, thereby extending symbol duration and total ToA. While higher SF values improve receiver sensitivity and range, they also consume more of the available duty-cycle budget, effectively limiting how often a device can transmit. Thus, selecting the appropriate SF involves a trade-off between communication reliability, data rate, and regulatory transmission limits.

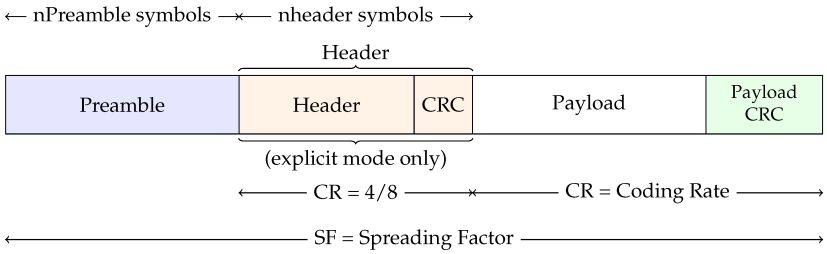


Figure 2.6 Composition of the LoRa physical frame [6].

The preamble is used by the receiver for frame detection and time synchronization. It allows the receiver to recognize the presence of an incoming transmission and synchronize the frequency and timing. The preamble has a minimum duration of 4.25 symbols but is typically configured to a higher value, most commonly eight symbols, to ensure reliable synchronization. The header is optional and follows the preamble, containing information about the transmission, such as the payload length, coding rate, and the presence of (CRC). The payload carries the user data — in the case of LoRaWAN, this corresponds to the MAC frame defined by the upper layer. Its length has a direct impact on the ToA, since every byte added increases the number of symbols transmitted. Finally, the payload CRC provides error detection at the physical layer, allowing the receiver to verify the integrity of the received data before forwarding it to higher layers.

Chapter 3

Related Work

This chapter reviews existing research and developments in the extension of LoRaWAN networks, with a focus on relay systems and standards, hybrid networks incorporating LoRa Mesh, satellite network integration, and advanced modulation techniques such as LR-FHSS. The purpose of this review is to study the current landscape of these solutions, identify gaps, and gather insights that will contribute to the development of our project.

3.1 Relay Systems

A relay in LoRaWAN, as shown in figure 3.1 is a device that extends the communication range between end devices and the gateway by retransmitting messages when direct communication is not feasible. This limitation often arises when the signal weakens due to long distances or physical obstacles. While adding a gateway can be effective for areas with multiple devices experiencing poor connectivity, deploying a relay is a more economical solution when only a few devices need coverage. In remote environments such as the open sea, installing multiple gateways may not be practical, making relays particularly valuable [21].

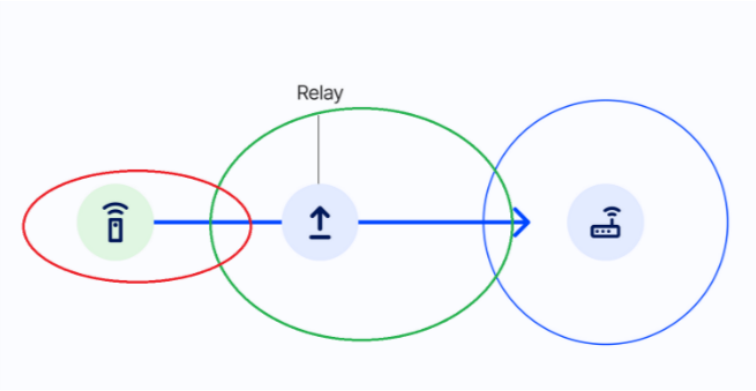


Figure 3.1 Relay operation between end devices and gateways [21].

In 2022, the LoRa Alliance introduced the Relay Specification TS011-1.0.0 [28], which expanded the LoRaWAN protocol with single-hop relays, extending coverage without requiring

a full mesh network. The specification enables certain LoRaWAN devices to function as relays, forwarding both uplink and downlink messages between end devices and the network server. Relays are designed to be cost-effective and energy-efficient, with hardware similar to standard LoRaWAN end devices. They can be battery-powered, simplifying deployment in areas lacking infrastructure. Each relay can support up to 16 end devices, offering a practical solution for extending coverage without additional gateways. However, this method may be insufficient for large-scale IoT networks [28].

The figure 3.2 illustrates the communication process using a LoRaWAN relay. Initially, the end device attempts to establish contact with the relay by transmitting a Wake on Radio, WOR, message. This WOR message activates the relay and conveys key radio parameters such as frequency and data rate for the upcoming communication. The relay acknowledges with a WOR-ACK message, confirming the link. To conserve energy, the relay returns to sleep mode if no activity is detected. It periodically scans the channel for activity using a mechanism called Channel Activity Detection, CAD. This scanning interval, known as CAD periodicity, can range from 25 ms to 1 second. The WOR frame's preamble can also extend up to 1 second to increase the chance of detection during these brief scans. If the relay detects activity, it transitions from CAD to reception mode, known as the CAD-to-Rx delay. Upon successful demodulation of the WOR frame, the relay sends a WOR-ACK frame to the end device.

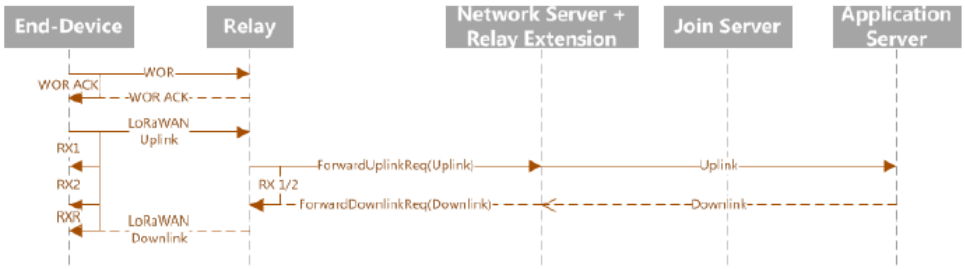


Figure 3.2 LoRaWAN relay message flow with WOR and RXR windows [28].

During uplink transmission, the end device sends a LoRaWAN message, which the relay captures and forwards to the network server using a ForwardUplinkReq message. The network server processes the data and, if needed, forwards it to the application server. For downlink communication, the network server sends a response back to the relay with a ForwardDownlinkReq message. The relay then transmits this message to the end device during the designated RX1 or RX2 reception windows, ensuring message delivery.

This relay-based communication structure enables LoRaWAN networks to maintain bidirectional data flow even when devices are beyond the gateway's direct coverage area, improving network reach and reliability. However, the TS11.0.1 relay specification [28] requires all devices to implement the protocol and perform the Wake-up on Radio (WOR) procedure, which can be incompatible and challenging for large-scale deployments with hundreds of devices.

In contrast, the proposed system in this work uses standard LoRaWAN messaging, where the device functions as both a gateway and a relay, staying continuously powered on. This

eliminates the need for each device to support complex procedures like WOR and CAD. Additionally, the system is designed to support more than 16 devices, addressing the limitations of the specification.

A thesis from Norway, [50], implemented this LoRaWAN relay standard and practical tests demonstrated its ability to extend LoRaWAN coverage. However, several issues were identified during testing. Initially, the relay functioned as expected, but after one hour of operation, it stopped relaying messages. When reset in bootloader mode, previously connected end devices failed to automatically reconnect, necessitating manual intervention. Additionally, the relay was unable to verify the MIC of the uplinks sent by the end node, resulting in packet drops.

The paper [72] presents the e-Node, a transparent LoRaWAN relay designed to improve link reliability while maintaining full protocol compatibility. The authors tested it in an industrial environment, showing signal quality improvements of up to 16 dB in RSSI and higher SNR when messages were relayed. However, the system focuses mainly on improving communication stability rather than extending coverage, leaving its performance in remote areas uncertain. In addition, the design depends on several interconnected components, including 2 Raspberry Pi units with LoRa chips, an Ethernet switch and an external processing unit such as a PC, which increases both cost and deployment complexity.

The paper [73] presents a transparent 2-hop LoRa relay designed to improve connectivity in rural, non-line-of-sight environments without modifying end devices or gateways. The relay operates by observing periodic transmission patterns from nearby devices and waking up accordingly to forward packets. However, the system relies on raw LoRa communication rather than LoRaWAN, which provides no built-in security and offers limited scalability for larger deployments. It does not evaluate range extension or scalability, and its learning-based approach limits performance to periodic traffic only.

3.2 LoRa Mesh

LoRa Mesh is a decentralized communication protocol that extends the range of LoRa networks through multi-hop message relaying. Unlike traditional start topoly LoRaWAN networks, where sensors are connected via single hop to gateways and relies on gateways for communication with a central server, LoRa Mesh acts like a web and allows nodes to communicate directly by forwarding messages between each other. Each node in the mesh network acts as both a transmitter and a relay, helping data reach its destination even if it is out of direct range. In a mesh network like the one in figure 3.3, data from a source node is relayed through multiple intermediate nodes before reaching the destination node, which then communicates with the gateway and forwards the data to the cloud for processing and storage. This process is called flooding, data is broadcasted to all neighbor nodes within range and each node forwards the message further to their own neighbors. This process continues until the message either reaches its destination or a predefined hop limit is met [29].

However, LoRa Mesh has limitations, such as increased network congestion with a high

number of nodes and reduced scalability compared to LoRaWAN. Additionally, depending on the message flooding technique used can lead to redundant transmissions. Since LoRa Mesh is not a standardized protocol, different implementations and deployment approaches are available

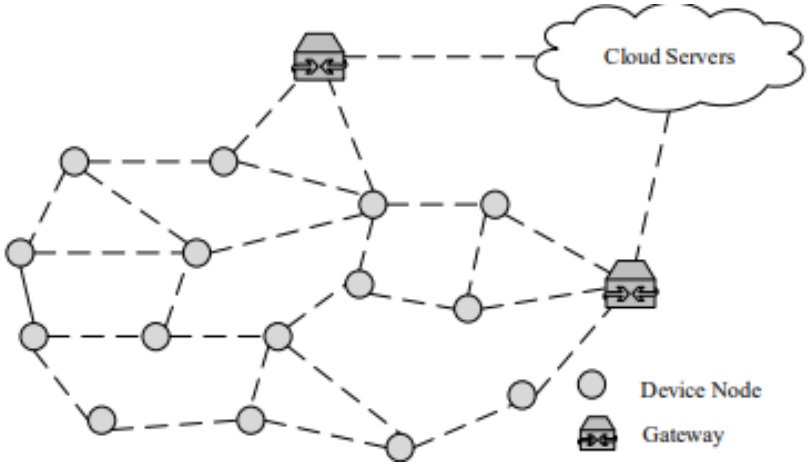


Figure 3.3 Overview of LoRa Mesh Architecture [29].

Meshtastic [31] is an open-source project using LoRa and the capabilities of LoRa Mesh to provide long-range communication. This community-driven project enables users to send text messages across a decentralized mesh network, where each node relays messages to extend coverage. Meshtastic doesn't require a dedicated gateway, instead nodes form a mesh by rebroadcasting received messages. The system uses a simple mesh routing algorithm with managed flooding, where nodes forward messages until the hop limit is reached, ensuring that even distant nodes receive messages. Meshtastic utilizes the full frequency spectrum available to LoRa technology within a given region, in contrast to LoRaWAN predefined set of standardized channels. While Meshtastic offers AES256 encryption, it currently lacks Perfect Forward Secrecy (PFS), making it vulnerable to "Harvest Now, Decrypt Later" attacks. It also lacks message integrity verification, leaving messages susceptible to tampering. Meshtastic reached a maximum range record of 331 km, but remains relatively limited compared to LoRaWAN, which has achieved an impressive record of 1336 km, demonstrating greater range and reliability.

Pymesh [32], developed by Pycom, is a full-mesh network that utilizes LoRa technology for communication. There are four distinct roles in Pymesh's network: The Leader Node is responsible for managing the overall configuration and coordination of the network, ensuring data is routed efficiently being critical for the operation of the network. A child node is typically an end device that transmits or receives data but does not have the ability to forward messages. The Border router serves as a bridge between the pymesh network and external systems, forwarding messages to the cloud via Wi-Fi, Bluetooth or cellular networks. Finally, the Router Node plays a crucial role in extending the network's coverage. These nodes forward messages between child nodes and other key nodes, such as the leader or border routers. Pymesh is compatible with Pycom development boards, such as the LoPy4 and FiPy and users can manage the devices through the Pybytes platform. However, as noted in [33], these devices have

limited battery life, typically lasting only a few hours. Pycom also faces challenges with timing synchronization, making it difficult to ensure consistent low power consumption across all devices in the network.

3.2.1 Hybrid networks

LoRa Mesh operates in a decentralized manner, where devices communicate with each other, forming a multi-hop network. This can be useful in situations where direct access to a gateway is not possible. However, it results in higher power consumption and increased latency due to the need for message relaying. On the other hand, LoRaWAN follows a star topology, where end-devices communicate directly with gateways that forward data to a central network server. This makes it more scalable, energy-efficient, and reliable for large-scale deployments. Additionally, LoRaWAN benefits from standardization by the LoRa Alliance, ensuring interoperability across different devices and vendors. Table 3.1 compares LoRa Mesh and LoRaWAN, two different network architectures utilizing LoRa technology.

Table 3.1 Comparison of LoRa Mesh and LoRaWAN for IoT Applications.

Feature	LoRa Mesh	LoRaWAN
Topology	Mesh, multi-hop communication	Star, gateway-based
Infrastructure	Decentralized	Centralized
Range	Shorter per hop, but extendable via relaying	Long-range, up to several km
Power Consumption	Higher due to multi-hop communication	Low, direct communication with gateway
Latency	Higher, due to multi-hop relaying	Lower, direct transmission to gateway
Scalability	Limited by mesh network complexity	Highly scalable, depends on number of gateways
Complexity	Higher, requires node coordination	Lower, devices only connect to gateways
Data Rate	Variable	Limited by SF
Security	AES, depends on implementation	AES with end-to-end encryption
Standardization	No formal standard	Defined by LoRa Alliance

While LoRa Mesh offers flexibility in certain use cases, such as isolated environments where direct gateway access is challenging, LoRaWAN is the preferred choice for applications requiring long-range communication, low power consumption, and centralized management. This work will adopt LoRaWAN as the communication-based protocol. Given that the focus of this thesis is to extend LoRaWAN coverage, exploring a hybrid model that combines the strengths of both LoRaWAN and LoRa Mesh is particularly relevant for further optimization.

LoRaWAN faces limitations in areas where direct communication with gateways is challenged by obstacles or distance. In such cases, LoRa Mesh can be used to create a multi-hop network, allowing devices to relay messages through intermediate nodes.

In [34], a hybrid architecture is proposed using LoRaWAN as the primary network and LoRa Mesh as a backup for multi-hop communication when direct LoRaWAN links fail. The system begins by checking the network mode stored in the device during startup, if LoRaWAN link fails, the device automatically switches to LoRa Mesh. After successful mesh communication, it restores the LoRaWAN mode, enters deep sleep to save energy, and later retries LoRaWAN. Experiments demonstrated that LoRa Mesh can effectively serve as a backup in dense and foliaged areas where LoRaWAN may fail due to obstacles. However, the article does not adequately analyze the energy consumption during the network switching process or evaluate the performance of the proposed system in large-scale deployments or environments beyond forests.

The research presented in [35] introduces another hybrid network model that combines LoRa Mesh and LoRaWAN technologies to enhance the performance of LPWAN communication systems. A key element of this model is the Proxy node which acts as a crucial intermediary between the LoRa Mesh network and LoRaWAN infrastructure, as illustrated in the figure 3.4. Using the Ad hoc On-Demand Distance Vector (AODV) routing protocol, the Proxy node not only coordinates the mesh network's operations but also ensures seamless data transmission to the broader LoRaWAN network, enabling efficient integration with external systems and applications.

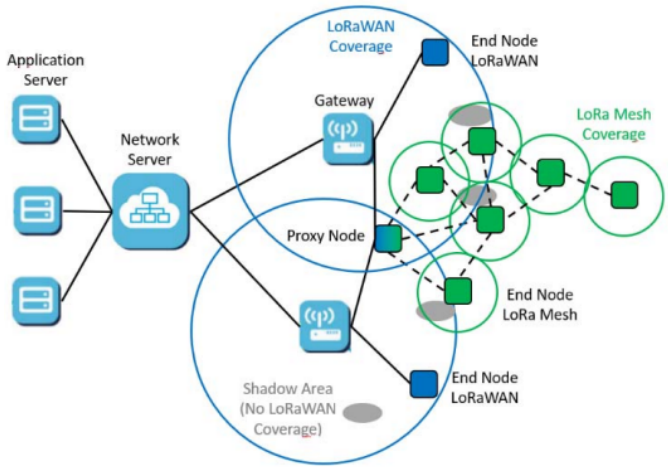


Figure 3.4 Architecture of the Hybrid Network proposed in [35].

Each mesh node is associated to a unique virtual LoRaWAN node, ensuring its distinct identification within the LoRaWAN backend. These nodes are equipped with a DevEUI, AppKey, and an encryption function that allows secure communication with the Proxy node. To optimize network efficiency, the Proxy node aggregates data from all mesh nodes into a single message, reducing the network overhead. In terms of data exchange, mesh nodes encrypt their data before transmitting it to the Proxy node, which signs the packet using the network session

key (NwkKey). On the downlink, the Proxy decrypts messages received from the LoRaWAN network and forwards them to the mesh nodes, which then decrypt the messages using their respective encryption functions. However, this study does not include tests of the full implementation of this solution, as it mainly focuses on evaluating the LoRaMesh routing protocol and the overall network performance. Despite this limitation, the proposal remains highly interesting, offering valuable insights into the potential of integrating LoRa Mesh and LoRaWAN for enhanced communication in LPWAN systems.

The ChirpStack Gateway Mesh [74] also combines LoRa mesh concepts with the LoRaWAN protocol through a software layer that runs directly on gateways. Gateways operate as either Relay Gateways, which forward uplinks and downlinks without internet connectivity, or Border Gateways, which unwrap the mesh payloads and communicate directly with ChirpStack. End devices remain unmodified and transmit standard LoRaWAN frames. Relay Gateways encapsulate these frames with metadata (RSSI, SNR, hop count, etc.) and forward them up to eight hops until reaching a Border Gateway, which unwraps the payload and sends it to ChirpStack as a normal uplink. Downlinks follow the reverse path. All coordination occurs at the gateway layer, protected by a shared AES-128 mesh root key that secures and authenticates each hop. The system can use the same regional band or a dedicated 2.4 GHz link for backhaul. Each frame introduces encapsulation overhead of about 14 bytes (uplink) and 15 bytes (downlink), which affects duty-cycle efficiency. It also depends on specific vendor firmware, such as RAKwireless WisGateOS or ChirpStack Gateway OS, limiting portability and wider adoption. Despite its technical potential, no peer-reviewed studies or experimental analyses have yet evaluated the ChirpStack Gateway Mesh in operational environments, leaving its real-world performance and scalability largely undocumented in the scientific literature.

3.3 Satellite networks

Satellite networks have revolutionized IoT communication by providing connectivity in regions where traditional terrestrial infrastructure is limited or absent, expanding IoT possibilities. IoT network architectures utilizing satellite constellations are generally categorized into two main approaches, Indirect-to-Satellite Internet of Things (ItS-IoT) and Direct-to-Satellite Internet of Things (DtS-IoT), as shown in figure 3.5.

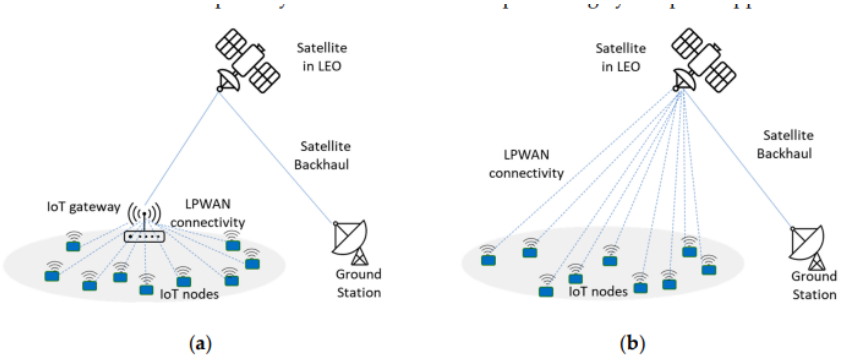


Figure 3.5 Satellite-based IoT Communication: a) ItS-IoT and b) DtS-IoT [39].

In ItS-IoT, satellites act as intermediaries between terrestrial LoRaWAN gateways and the network server. Ground-based gateways first collect data from IoT devices, which is then transmitted to satellites and relayed to the ground station. This architecture is particularly useful in environments where existing terrestrial infrastructure can serve as a bridge to satellite communication, particularly in dense or complex regions where traditional gateways may struggle with coverage [39]. In contrast, Direct-to-Satellite IoT (DtS-IoT) allows IoT devices to communicate directly with satellites, eliminating the need for terrestrial gateways. In this setup, satellites can act as dynamic gateways that collect data from IoT devices and relay it to the ground station.

LEO satellites, including CubeSats, have become the preferred constellation for supporting IoT communications. Positioned between 500 and 2,000 kilometers above Earth, LEO satellites offer several advantages over higher orbit constellations, such as GEO satellites. LEO satellites benefit from much lower propagation delays and path losses due to their proximity to Earth, in contrast to GEO satellites, which orbit at an altitude of approximately 35,786 kilometers, experience higher latency and longer propagation delays. LEO satellites travel at speeds of approximately 7.8 km per second, completing an orbit in about 90 minutes. Due to this fast orbit, a single LEO satellite cannot provide continuous coverage, which is why constellations of hundreds or even thousands of satellites are deployed, ensuring uninterrupted global coverage [36, 37].

Several key satellite service providers are actively contributing to global IoT connectivity by operating LEO constellations.

Lacuna Space [40] stands at the forefront of using LoRaWAN with satellite connectivity, utilizing LEO satellites to establish a DtS-IoT network. It was the first company to deploy LoRaWAN from space. In April 2019, Lacuna Space launched its first satellite, marking a significant milestone in satellite-based IoT communications with plans to grow its constellation to hundreds or thousands of satellites. As illustrated in the architecture shown in Figure 3.6, an end device transmits data to a satellite, which stores the packet until it passes over a ground station. The ground station receives the data and forwards it to a LoRaWAN backend for further processing. In this configuration, the satellite functions as gateways in the sky, capturing and demodulating messages from end devices. Currently, Lacuna Space only supports uplink transmission meaning that it is used ABP for joining the network, rather than OTAA [38, 39, 42].

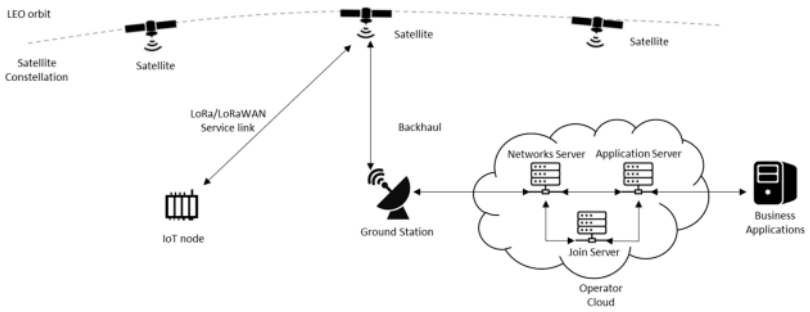


Figure 3.6 LoRaWAN and satellite integration architecture [39].

Implementing LoRaWAN with satellite integration introduces several technical challenges, with the Doppler Effect being one of the most significant. This effect occurs due to the high relative speed between LEO satellites and ground-based IoT devices, causing frequency shifts in transmitted signals. In [41], it was demonstrated that LoRa modulation performs reliably at lower altitudes with SF ≤ 11, exhibiting strong immunity to the Doppler effect when operating with SF values between 7 and 11. However, with SF = 12, disruptions occur in orbits below 550 km due to an increased rate of frequency shift. Another challenge is synchronizing data transmission between devices and satellites, which is affected by signal propagation delay. There are also difficulties in selecting appropriate satellites as gateways, allocating resources effectively, and optimizing communication across different network layers. Inter-satellite communication within LEO systems faces data link and physical layer challenges, as ensuring seamless communication between satellites in low Earth orbit can be complex [37]. It is also important to note that IoT devices such as end nodes need different antennas to communicate with LEO satellites compared to those used for terrestrial LoRaWAN gateways, such as dipole antennas. Lacuna Space, for example, uses right-hand circular polarized antennas that irradiate towards the sky. Devices already deployed in the field may not be equipped with these specialized antennas, increasing the cost for companies to upgrade their equipment. Furthermore, delay tolerance is an inherent aspect of DtS-IoT. Regarding latency, the inherent delays in DtS-IoT satellite communication, as highlighted in [42], may introduce time gaps of minutes between uplink and downlink. This latency could impact real-time applications that require instantaneous data transmission, making it less ideal for certain IoT applications that rely on low-latency communication.

While the integration of LoRaWAN with satellite technology provides an innovative solution for expanding IoT coverage, it also brings challenges making it less suitable for IoT applications that demand real-time data transmission. However, for use cases where latency is less critical, satellite-based LoRaWAN could offer a valuable solution for enabling IoT connectivity in remote regions. However, the use of satellite networks often leads to increased costs, which is an important factor to consider when evaluating the feasibility of such solutions for different IoT applications.

3.3.1 LR-FHSS

LR-FHSS was designed primarily to support satellite-based LoRaWAN communication by increasing capacity and robustness. Instead of transmitting on a single fixed frequency like standard LoRa, LR-FHSS rapidly hops between different frequencies during transmission. It transmits low data-rate streams over a wide bandwidth using a pseudo-random frequency-hopping pattern, which improves resistance to interference and Doppler shifts common in satellite links. Optimized for uplink traffic, LR-FHSS enables thousands of simultaneous transmissions with bit rates between 162 and 325 bps corresponding to four data rates (DR8 to DR11), offering a balance between robustness and spectral efficiency [75].

In [76], the study demonstrates that LR-FHSS improves LoRaWAN's capacity in satellite communication but struggles with scalability as node density increase. With 50000 nodes, the success probability is 74,35% for DR8 and 68,22% for DR9 and at 150,000 nodes drops to around 8,11%. In [77] it was proposed a LR-FHSS transceiver for DtS-IoT systems that offers improved sensitivity and lower SNR requirements by up to 1.6 dB compared to LoRa, highlighting its superior performance for DtS-IoT systems.

3.4 Gaps in current research and technology

Previous research has investigated various aspects of LoRaWAN technology, including its applications in challenging environments and the use of nodes as relays to extend network coverage. Recent studies have also explored hybrid networks combining LoRa Mesh with LoRaWAN, integration with satellite networks, and advancements in LoRa modulation. However, existing research on LoRa relaying have been limited, not considering cases where data is required to be delivered directly at the edge (for local processing, or edge computing related activities), or that for providing message relaying, require new hardware and software for the end devices to be used along with changes to the network server, making this solution not possible for devices already deployed or in the market.

While the reviewed literature demonstrates significant progress in IoT and LoRaWAN applications for remote areas, several key gaps remain:

1. **Scalability and Economic Viability**: Additional research is needed to explore the scalability of IoT and LoRaWAN systems for widespread deployment, including detailed cost-benefit analyses for different operational scales.

2. **Resilience in Harsh Environments:** Although LoRaWAN is well-suited for long-range and low-power communication, there is a lack of studies evaluating its long-term reliability and performance under harsh environmental conditions.

3. **Energy efficiency:** Although some studies have explored power consumption optimization, there is still a gap in developing efficient energy management solutions, particularly

for long-term use in remote locations. For mesh networks, the research tends to focus on technical functionality, often neglecting the impact of energy consumption.

4. **Hybrid LoRaWAN–LoRa Mesh Architectures:** Most research focuses on theoretical approaches to hybrid networks combining these 2 protocols but few have tested a real-world deployment and interaction of these two networks. Hybrid architectures combining LoRaWAN and LoRa Mesh are often addressed only at the conceptual level. Real-world deployments demonstrating interoperability and interaction between these two networks remain limited.

5. **Integration with DtS-IoT (Direct-to-Satellite IoT):** While satellite-enabled LoRaWAN has emerged as a promising approach to overcome terrestrial coverage limitations, its operational costs remain prohibitive for small-scale or community-based applications. Subscription-based satellite services introduce recurrent expenses that are economically unsustainable for small operators, particularly in maritime and agricultural domains.

6. **Lack of Backward Compatibility in the TS011-1.0.0 standard:** The official LoRa Alliance relay specification (TS011-1.0.0) introduces a standardized approach for message relaying within LoRaWAN. However, it requires end devices to implement new procedures such WOR/CAD, as well as new message types. This dependency on firmware and protocol modifications makes the solution incompatible with the vast majority of already deployed LoRaWAN devices. For large-scale deployments, the cost and logistical effort to update or replace devices are prohibitively high, making the TS011 approach impractical in many real-world scenarios.

7. **Limited Network Scalability of TS011-1.0.0:** In addition to its compatibility issues, the TS011 standard supports only up to 16 end devices per relay, severely limiting its use in dense or distributed network topologies. This constraint limits its adoption in large or dynamic environments requiring more flexible and adaptive relaying mechanisms.

8. **Absence of Edge-Oriented Relaying Mechanisms:** Existing LoRa relay or mesh solutions are primarily designed for transparent forwarding of packets towards a central network server. None of them consider edge processing, where data is partially or fully processed at the relay node before transmission.

This project, particularly through the relay component, aims to address some of these gaps, particularly in extending network coverage and improving data collection in remote areas.

3.4.1 Comparative Summary of Relay Approaches

Table 3.2 provides a comparative summary of the approaches discussed in the related work for extending LoRaWAN network coverage.

Table 3.2 Comparative table of proposals for extending LoRa/LoRaWAN coverage.

Solution	Technology	Advantages	Limitations	Why not used
Relay Standard TS011 (single-hop relay) [28]	LoRaWAN	Standardized; low cost; battery-capable; supports UL/DL	Requires WOR/CAD on EDs and NS extensions; limited to 16 EDs/relay; timing-sensitive	Not backward-compatible with deployed devices; scalability insufficient for target scenario.
"e-Node" (transparent repeater) [72]	LoRaWAN	No changes to ED firmware; improves RSSI/SNR; frame compatible with LoRaWAN	Requires several interconnected units (2×Raspberry Pi + LoRa, switch, external PC); costly and complex to deploy; focus on link reliability, not coverage extension	High cost and complexity; unsuitable for wide-area remote environments.
Transparent 2-hop LoRa relay [73]	Raw LoRa	Very low cost; simple; battery-friendly	Not compatible with LoRaWAN core; no security/join; designed for periodic traffic only	Lacks LoRaWAN compatibility and security; unsuitable for project integration.
Increase SF / add gateways [71]	LoRaWAN	Improves link budget and range; simple implementation	Higher SF reduces capacity; gateway densification increases cost and complexity	Financial and logistical constraints make large-scale densification impractical.
ChirpStack Gateway Mesh [74]	Hybrid (Mesh + LoRaWAN)	End-devices unchanged; multi-hop between gateways; per-hop authentication	Depends on vendor firmware; encapsulation overhead affects duty-cycle; limited peer-reviewed validation	Vendor dependency and lack of edge processing.
Hybrid Network (fallback mode) [34]	Hybrid (Mesh + LoRaWAN)	Automatic fallback to LoRa Mesh when LoRaWAN fails; restores LoRaWAN after recovery; improves connectivity in obstructed environments	Energy use during switching not analyzed; untested for large-scale or non-forest scenarios	Limited validation; lacks scalability and energy analysis for broader deployment.
Hybrid Network via Proxy node [35]	Hybrid (Mesh + LoRaWAN)	Proxy node aggregates and secures data between Mesh and LoRaWAN; maintains unique virtual device IDs; ensures encrypted exchange	Higher complexity; lacks full end-to-end validation; energy efficiency and scalability unverified	Unproven full system validation and system complexity.
Meshtastic [31]	LoRa Mesh	Decentralized; low cost; long distances achieved	Not LoRaWAN; lacks PFS and integrity checks; flooding overhead; higher latency/energy	Fails LoRaWAN interoperability and security requirements.
PyMesh [32]	LoRa Mesh	Full mesh; supports cloud bridging; device management tools available	Short battery life; timing sync issues; vendor dependence	Limited autonomy and vendor lock-in; not compatible with LoRaWAN stack.
Satellite (DtS-IoT) Networks [76]	LR-FHSS (DR8–DR11)	Higher capacity; Doppler-robust; suited for LEO satellites	Low bitrates; additional antenna cost; recurring subscription fees	High cost and latency unsuitable for small-scale or near-real-time applications.

After analysing the existing solutions discussed in this chapter, it became evident that none of the current LoRa or LoRaWAN relay approaches fully met the requirements of this project. The official LoRa Alliance TS011 relay specification [28] introduced a standardized

mechanism but required firmware and protocol modifications in all end devices, compromising backward compatibility with already deployed nodes. Alternative academic proposals, such as the e-Node [72] or 2-hop transparent relays [73], demonstrated functional range extension but relied on complex, multi-component hardware or operated outside the LoRaWAN framework.. Hybrid models combining LoRa Mesh and LoRaWAN [34, 35] offered interesting redundancy mechanisms but remained energy-intensive and lacked interoperability reserach with standard LoRaWAN deployments.

Given these constraints, the proposed architecture adopts a relay model based on 2 components interconneted: one that acts as a gateway (C-Mesh) and another that relays messages (C-Point). This relay is designed to maintain full compliance with the LoRaWAN protocol and operate with any vendor specific node, all it is needed is to send a simple lorawan message. This approach ensures retro-compatibility, avoiding firmware changes on end-devices. The relay will integrate a local chirpstack instance that supports more than sixteen nodes per relay, overcoming the scalability limitations of the TS011 specification. Additionally, the system includes a mechanism for local interaction and Over-The-Air Activation (OTAA) key synchronization via BLE, which coordinates the join procedures between the relay and the network server.

Chapter 4

System Architecture

This thesis builds on the ideas of the Custodian Project [43], which demonstrated local (at-sea) and remote (shore-based) monitoring of fishing gear and vessels using low-cost hardware. The project introduced three devices: C-Beacon, C-Point and C-Mesh. C-Beacon was attached to fishing gear and was designed for long deployments at sea. C-Point, installed on vessels, acted as the user-facing interface and bridge to a mobile application via BLE. C-Mesh, also vessel-mounted, operated as both a gateway for offshore coverage and a relay, extending communication with C-Beacons when direct shore connectivity is unavailable.

The focus of this thesis is the implementation and evaluation of a LoRaWAN-based relay with edge processing using the project's C-Mesh hardware. In our prototype, C-Beacon and C-Point are instantiated on TTGO T-Beam boards, each configured for its respective role. Although experiments at sea were originally planned, the evaluation was carried out in a representative urban setting (Lisbon) due to logistical constraints. The scope of this work includes the repeater/relay functionality and a proof-of-concept BLE path for local interactions for OTAA keys synchronization.

4.1 Architecture Overview

This section describes the system architecture of the proposed LoRaWAN relay. The architecture is divided into four logical domains: field devices (C-Beacon), edge relay (C-Mesh and C-Point), backhaul/gateway infrastructure, and the core backend (VM). Figure 4.1 illustrates the main components and data flows.

C-Beacon devices are low-power end devices and operate as standard LoRaWAN Class A nodes. They periodically transmit sensor data using uplinks. These frames may be received either directly by an internet-connected LoRaWAN gateway or indirectly through the edge relay. The repeater has no internet access and is designed to operate in remote areas without mobile networks or Wi-Fi coverage. It consists of two interconnected components: the C-Mesh, which is responsible for continuously receiving LoRa messages and performing local processing and the C-Point, which is responsible for forwarding those packets.

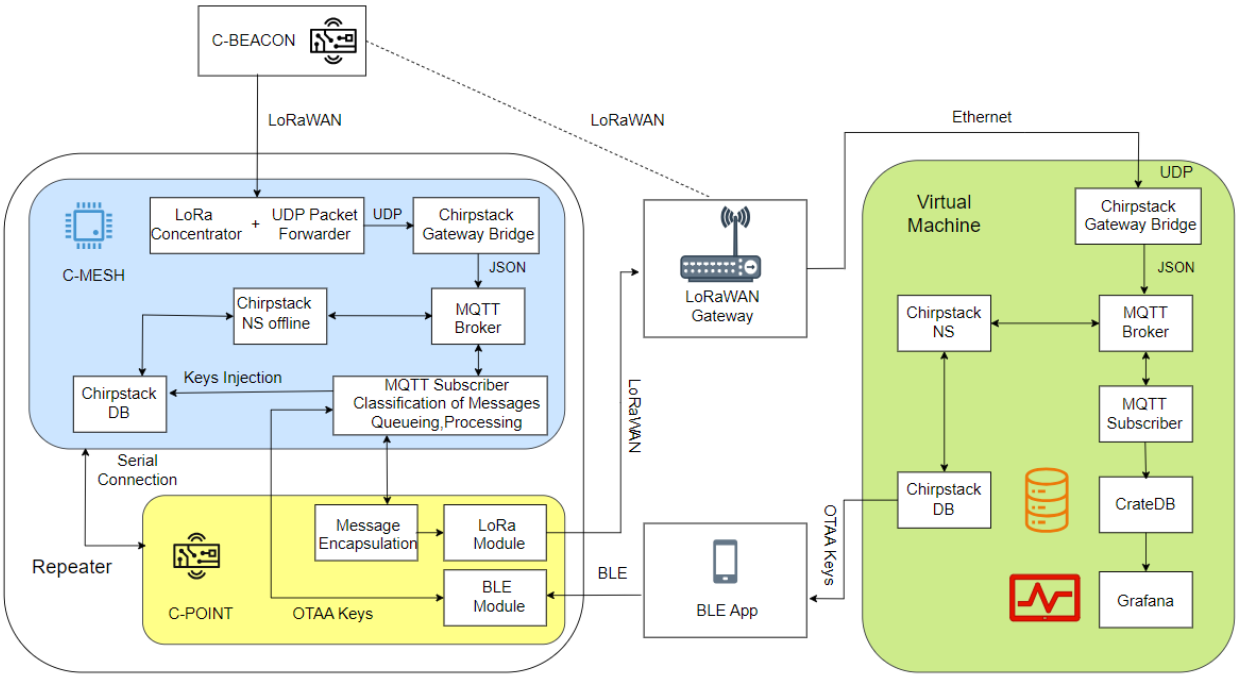


Figure 4.1 Proposed system architecture.

The C-Mesh is equipped with a LoRa concentrator that listens simultaneously to multiple channels, acting as a gateway. On the C-Mesh, a Semtech UDP packet forwarder (section 2.2.1) is running together with a local ChirpStack Gateway Bridge, which translates the received UDP packets into JSON format and publishes them to the local Mosquitto MQTT broker (section 2.2.1). This broker exposes topics to which other services can subscribe. An offline ChirpStack Network Server (section 2.2.1), also deployed on the C-Mesh, subscribes to these topics and consumes the uplinks. It then handles duplicate messages, validates the MIC and frame counters, and manages uplinks from the devices registered in its database. Alongside the ChirpStack NS, a custom Python software stack subscribes to the MQTT topics, extracts and processes the payloads and other relevant fields from C-Beacon devices and converts them into byte streams. These are enqueued and sent over a serial connection to the C-Point. The C-Point listens on the serial port, receives these messages, and encapsulates the data into the payload of a new LoRaWAN message. This message is then transmitted via LoRaWAN to an internet-connected gateway. The uplink is received by the standard gateway and forwarded as in a conventional LoRaWAN architecture to the online ChirpStack NS running on a virtual machine (VM). On the VM, a Python consumer subscribes to the MQTT broker topics and stores the uplinks in a database. A visualization service connected to this database provides dashboards for monitoring and analysis of the collected data. As mentioned, this project relies on two ChirpStack instances:

- **ChirpStack (offline) on C-Mesh**: contains only the C-Beacon devices and the C-Mesh gateway, handling device authentication, payload decryption, key management and dedu-

plication at the edge.

- **ChirpStack (online) on the VM**: acts as the central platform. It manages both C-Beacon and C-Point registrations and is connected to the internet-enabled LoRaWAN gateway.

The ChirpStack network server plays a central role in this work. The use of two ChirpStack instances addresses a key challenge in this project: handling message deduplication, key management, encryption/decryption, and MIC calculation both at the edge and in the central platform. ChirpStack already provides robust support for these functions, eliminating the need to develop a custom NS. Furthermore, its web interface allows for user-friendly device registration. However, to access the C-Mesh instance through a browser, some network connectivity is required, as the web interface is bound to the C-Mesh IP and port. One of the main motivations for using ChirpStack is its flexibility and scalability, overcoming the limitation identified in Section 3.4, where the TS011-1.0.0 standard restricts each relay to a trusted list of only 16 end devices. In contrast, ChirpStack has no such limit, enabling support for a significantly larger number of connected nodes within the same network architecture.

For devices using ABP activation, two ChirpStack instances are not a problem. For OTAA devices, however, additional coordination is required. The offline ChirpStack instance does not accept join requests, OTAA joins are handled exclusively by the online ChirpStack. This raises the question: once a device joins the online ChirpStack, how can the offline instance on the C-Mesh accept its messages and allow the repeater to function? This is where the C-Point plays a crucial role through OTAA key synchronization via BLE. The C-Point is equipped with a BLE module that communicates with a mobile BLE application. This app retrieves the new OTAA session keys from the online ChirpStack database and transmits them to the C-Point via BLE. The C-Point then forwards these keys via serial connection to the C-Mesh, which injects them into its local ChirpStack database. With the updated keys, the C-Mesh can accept uplinks from the device, process them locally, and relay them through the C-Point to the central infrastructure.

The BLE application itself is outside the scope of this work. For testing purposes, an existing and limited BLE app will be used, requiring manual input of the keys. Consequently, OTAA activation experiments will only be performed in a laboratory setting. This component of the system therefore serves primarily as a proof of concept.

4.2 Hardware Selection

The relay system requires hardware capable of simultaneously handling LoRaWAN communication, message forwarding, and edge-level processing tasks such as key synchronization. The selection of each component was guided by three main criteria: compatibility with existing LoRaWAN devices, scalability for future deployments, and sufficient computational resources for running the required software stack.

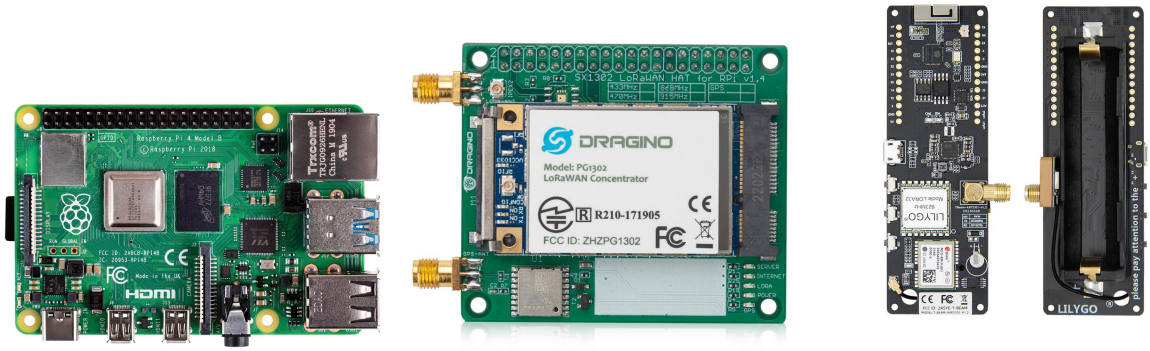
The backbone of the network relies on a dedicated virtual machine (VM) to support the network and application server layer. This VM runs Ubuntu 22.04 with 2 vCPUs, 4 GiB of RAM,

and 40 GiB of disk space, and hosts the ChirpStack NS, CrateDB for data storage, and Grafana for visualization and system monitoring.

A Mikrotik LtAP LR8 LTE6 gateway (Fig. 4.2) was configured to connect to the ChirpStack NS instance running in the VM. This gateway provides reliable LoRaWAN coverage and is responsible for receiving uplink messages from both the end devices and the relay, forwarding them to the network server for further processing.

Figure 4.2 Mikrotik LtAP LR8 LTE6 gateway [78].

The C-Mesh device is composed of three primary hardware components designed to ensure reliable LoRaWAN communication and message processing. At the core of the system is a Raspberry Pi 4 (Fig. 4.3a), equipped with a quad-core ARM Cortex-A72 processor with sufficient RAM and storage for message processing and database management. This unit is responsible for managing the software stack, handling communication protocols, and performing data processing tasks. For LoRaWAN connectivity, the C-Mesh integrates a Dragino LoRaWAN Concentrator HAT (Fig. 4.3b), which incorporates the Semtech SX1302 LoRa concentrator chip. This chip supports multi-channel and multi-data rate LoRa packet reception, enabling simultaneous communication with multiple end devices. The concentrator operates as a gateway for the LoRaWAN network, ensuring efficient data handling from various sources. The C-Point includes a LoRaWAN node module, built around the Semtech SX1276 chip, which is dedicated to relaying LoRa messages. In practice, a TTGO T-Beam board (4.3c) was used, which combines the SX1276 and BLE support. The C-Beacon was also implemented on a TTGO board, in order to demonstrate the compatibility of the relay with LoRaWAN end devices already available on the market, although any Class A LoRaWAN end device could be used instead.



(a) (b) (c)

Figure 4.3 Hardware components used in the relay: Raspberry Pi 4 [79] (a), Dragino PG1302 concentrator [80] (b) and TTGO T-Beam board [81] (c).

The combination of a Mikrotik gateway, Dragino concentrator, TTGO board, Raspberry Pi 4, and a supporting VM provides the necessary radio capabilities, computational resources and deployment flexibility to implement this project.

4.3 Communication Flows

This section describes the communication flows and message formats between the C-Mesh and C-Point components, detailing how application data and control information are exchanged. These flows are essential for the correct operation of the repeater.

4.3.1 Message Formats

The C-Mesh and the C-Point exchange ASCII lines that always start with the literal prefix CMESH: and end with \r\n. Fields are colon-separated and hexadecimal fields are written in uppercase. The first field after the prefix is a two-digit which identifies the type of the message:

- **01 - C-Beacon Forwarding**: application data received by C-Mesh and sent to C-Point.
- **02 - OTAA key update**: Session-keys provisioning from C-Point to C-Mesh.

Each message carries an 8-hex-digit random `MSG_ID` used to match acknowledgements and radio results. Table 4.1 summarises all message lines exchanged in both directions. Field formats are defined as follows: `DevEUI` consists of 16 hexadecimal digits and `DevAddr` of 8. Both session keys, `AppSKey` and `NwkSKey`, are 32 hexadecimal digits long.

Table 4.1 Message formats used between C-Mesh and C-Point.

Type	Direction	Message format
01	C-Mesh → C-Point	`CMESH:01:<MSG_ID>:<DevAddr>:<PayloadHex>`
	C-Point → C-Mesh	`CMESH:01:ACK:<MSG_ID>`
	C-Point → C-Mesh	`CMESH:01:TXOK:<MSG_ID>`
	C-Point → C-Mesh	`CMESH:01:TXERR:<MSG_ID>`
02	C-Point → C-Mesh	`CMESH:02:<MSG_ID>:<DevEUI>:<DevAddr>:<AppSKey>:<NwkSKey>`
	C-Mesh → C-Point	`CMESH:02:ACK:<MSG_ID>`
	C-Mesh → C-Point	`CMESH:01:NACK:<MSG_ID>:<ERR_CODE>`

The subsequent paragraphs detail how each message operates in practice. For type `01`, the exchange consists of the C-Beacon payload transmission to the C-Point. If received and accepted (`ACK`), it is followed by a radio-status report (`TXOK`/`TXERR`) for the same `MSG_ID`. For type `02`, the exchange consists of an OTAA provisioning request and a single acknowledgement emitted only after a successful database update in Chirpstack. In both cases, syntactic validation (fixed hexadecimal lengths and expected prefix) is checked before any action is taken.

Type 01 — C-Beacon forwarding

`CMESH:01:<MSG_ID>:<DevAddr>:<PayloadHex`

Sent by C-Mesh to C-Point to transmit application data to the C-Beacon device identified by `<DevAddr>`. After validating the line, C-Point returns an acknowledgement in case it received and then reports the radio outcome for the same `MSG_ID`:

- `CMESH:01:ACK:<MSG_ID>` (line accepted)
- `CMESH:01:TXOK:<MSG_ID>` (LoRaWAN transmission succeeded)
- `CMESH:01:TXERR:<MSG_ID>` (LoRaWAN transmission failed)

C-Mesh waits for `ACK`, confirming that the C-Point has received and accepted the CMESH line and will proceed with its LoRaWAN transmission. If no `ACK` arrives within the configured timeout, C-Mesh retries 2 more times (3 in total) and eventually drops the message. If the `ACK` arrives then, C-Mesh waits for `TXOK/TXERR` which report the outcome of LoRaWAN transmission for the same `MSG_ID`. A `TXOK` indicates the frame was transmitted successfully and a `TXERR` indicates a transmission failure. If a `TXOK` arrives, C-Mesh sends the next message in the queue. If a `TXERR` arrives, the C-Mesh will reschedule the message for retransmission later on and proceed with the queue.

Type 02 — OTAA key update

`CMESH:02:<MSG_ID>:<DevEUI>:<DevAddr>:<AppSKey>:<NwkSKey>`

Sent by the C-Point to the C-Mesh, this message injects the LoRaWAN OTAA session keys for the device identified by <DevEUI>. The C-Mesh processes the request and updates the device session in the local ChirpStack instance. It then responds with one of the following outcomes:

- CMESH:02:ACK:<MSG_ID> (update successful)
- CMESH:02:NACK:<MSG_ID>:<ERR_CODE> (update failed)

The <ERR_CODE> field indicates the reason for failure and can take one of the following values: (1) = BAD_FORMAT, (2) = UNKNOWN_DEVICE, (3) = OTHER.

4.3.2 Message Flows

The data exchange process within the platform involves multiple stages, with the C-Mesh device playing a pivotal role in each step. When a C-Beacon generates data, such as location coordinates, it encapsulates the information into a LoRaWAN payload and transmits it to the C-Mesh device. The C-Mesh device, equipped with a LoRaWAN concentrator, receives the encrypted payload and initiates the decryption process using the appropriate security keys. Once the message is decrypted, the C-Mesh device extracts the relevant data fields , enqueues the message and prepares it for transmission to the C-Point. The decrypted data is packaged according to the custom protocol messages format explained in section 4.3.1 and sent over serial connection to the C-Point as described in 4.4. After receiving the message, the C-Point returns an acknowledgement (ACK) to the C-Mesh, which remains in a waiting state until it arrives. If the message is valid, the C-Point encapsulates it into the payload of a new LoRaWAN frame and forwards it to the gateway. Upon a successful transmission, the C-Point immediately issues a TXOK, allowing the C-Mesh to proceed with the next message in its queue. Once the gateway receives the uplink from the C-Point, it forwards the frame to the ChirpStack Network Server for further processing. However, if a LoRa transmission error occurs, the C-Point responds with a TXERR to the C-Mesh. Upon receiving this status, the C-Mesh reschedules the message for retransmission while continuing to process the remaining queue. When the retry timer expires, the C-Mesh resends the same message to the C-Point, which again attempts to forward it over LoRa. If the transmission succeeds, the C-Point issues a TXOK and the message is delivered to the gateway; otherwise, it returns another TXERR. At this point, the C-Mesh detects that the same MSG_ID has failed twice and permanently discards the message.

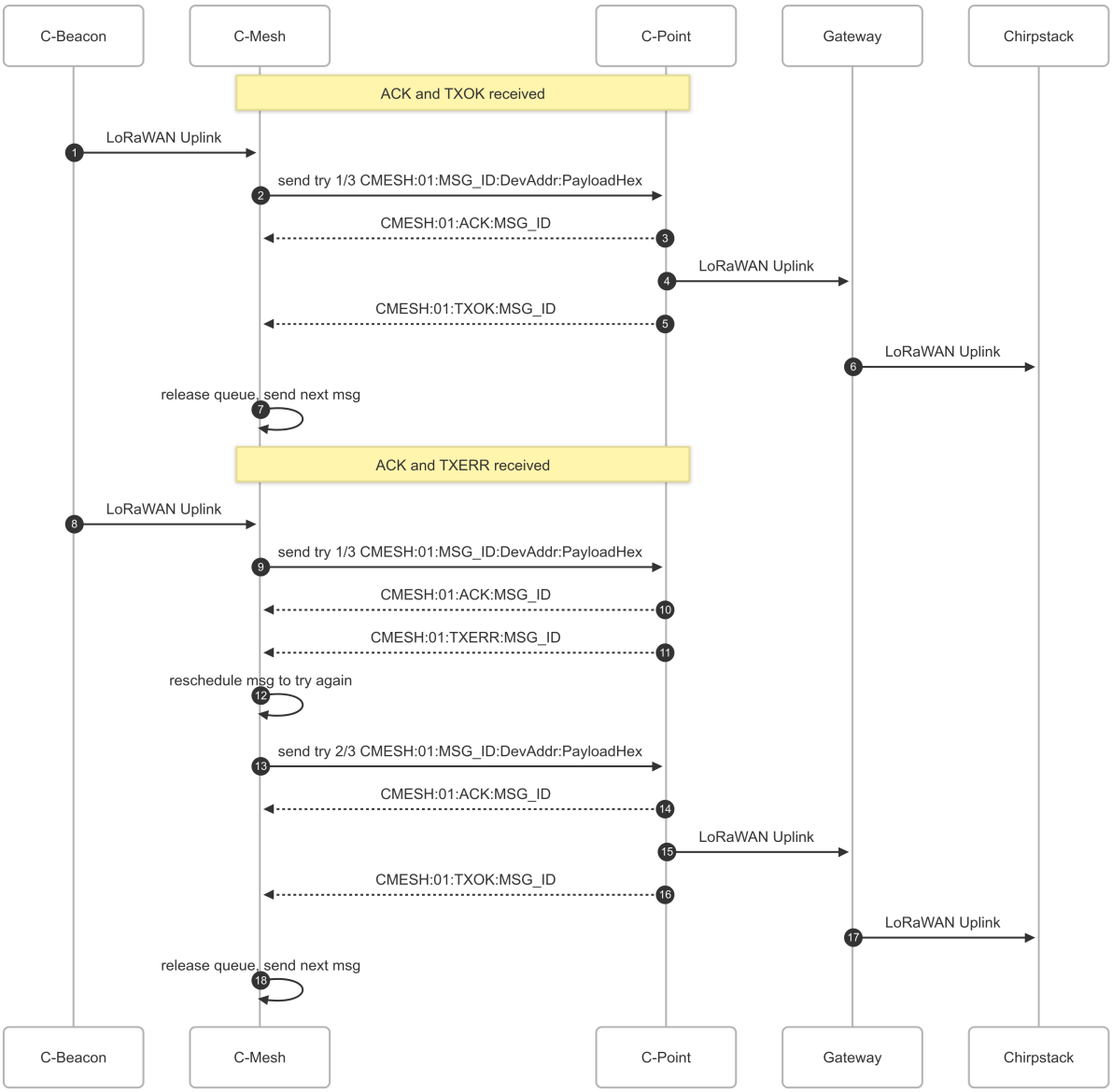


Figure 4.4 Message exchange flow between C-Mesh and C-Point, showing ACK/TXOK success and ACK/TXERR retransmission.

Another scenario is illustrated in Figure 4.5. If the C-Mesh does not receive an ACK, it retries the transmission up to two additional times (three attempts in total), sending the same message over the serial link to the C-Point. If no ACK is received after the third attempt, the message is dropped and the queue advances. If an ACK is received on the second attempt, the process continues as in Figure 4.4, with subsequent `TXOK` or `TXERR` messages.

However, the absence of an ACK at the C-Mesh does not necessarily mean that the C-Point failed to send one, the ACK could have been lost on the serial connection. In such cases, the C-Mesh unnecessarily retransmits the message, causing the C-Point to receive a duplicate. To handle this, the C-Point implements a deduplication mechanism: if the same `MSG_ID` is received within a certain time window, it is treated as a duplicate and ignored, preventing redundant LoRaWAN transmissions.

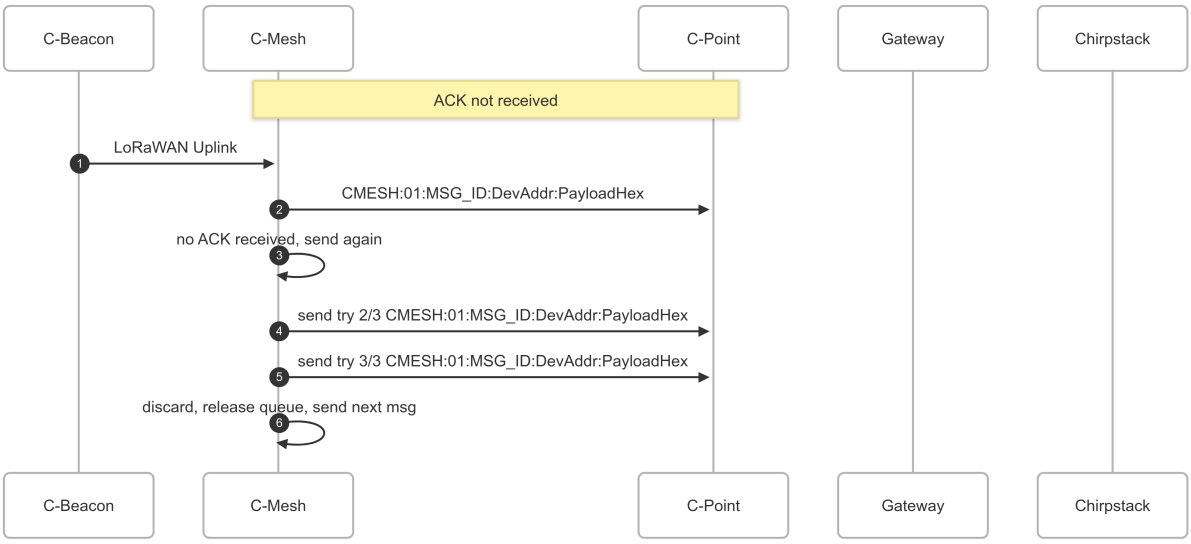


Figure 4.5 Message exchange flow between C-Mesh and C-Point, showing retransmission in case of ACK loss.

As mentioned earlier, the C-Point does not only forward C-Beacon messages but also implements a mechanism to synchronize OTAA session keys after a C-Beacon device performs an OTAA join in the ChirpStack online instance (Figure 4.6). In this case, if the C-Beacon subsequently sends a message to the C-Mesh, it will be rejected because the session keys are no longer valid. As described in Section 4.1, the new OTAA session keys must therefore be injected into the offline ChirpStack database instance.

The process begins by retrieving the OTAA session keys from the online ChirpStack database after the device has joined the network. Since the BLE mobile application was not within the scope of this work and not customized for this purpose, the keys were entered manually into an existing and limited BLE application and transmitted over BLE without encryption. This procedure was used exclusively for laboratory testing and validation, and is not intended for deployment in a production environment. After the application connects to the C-Point, the C-Point receives the `DevEUI`, `DevAddr`, `AppSKey`, and `NwkSKey`, and constructs a `CMESH:02` message with a random `MSG_ID`. This message is sent over the serial link to the C-Mesh, which processes it and attempts to update the session keys in the local ChirpStack database. If the update succeeds, the C-Mesh replies with an `ACK`, which the C-Point forwards to the mobile application. Otherwise, it returns a `NACK` with the cause of the error, which is then passed back by the C-Point to the BLE application.

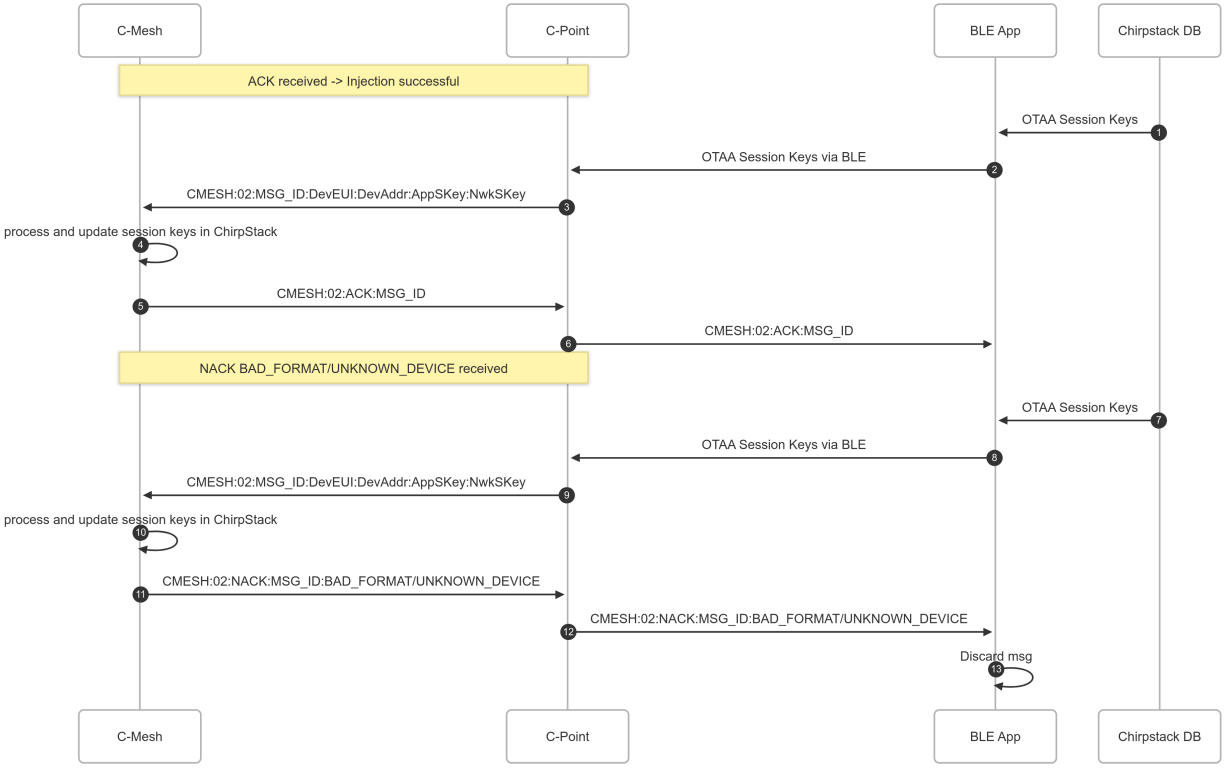


Figure 4.6 Message exchange flow between C-Mesh and C-Point, showing session key injection via BLE.

Chapter 5

System Development

This chapter describes the software components developed and integrated both in the edge device (repeater) and in the virtualized environment (VM) hosting the LoRaWAN network infrastructure. The goal is to provide a clear view of the full software stack, from the Network Server and Application Server configuration to the relay software that runs on the Raspberry Pi.

The chapter starts by describing the infrastructure setup on each domain: installation and configuration of the Semtech packet forwarder, the MQTT brokers, the local and online ChirpStack instances, the CrateDB database, and the Grafana dashboards. We then detail the custom implementation built on top of these components, including the Python code that runs on the relay, the message protocol, and the database ingestion of uplinks for subsequent analysis and visualization.

5.1 Infrastructure Setup

Before describing the individual software modules of the repeater, it is necessary to present the setup and configuration of the main system components, as well as the software stack installed on each. The system software is divided into two domains:

- **Raspberry Pi (Relay Node)**: runs the custom relay software, implemented in Python, which is responsible for receiving, processing and forwarding messages between the C-Point and the LoRaWAN infrastructure.
- **Virtual Machine (VM)**: hosts the ChirpStack Network Server online instance, the Application Server, the MQTT broker and the database.

A key aspect of this work was the implementation of custom software to manage the integration between the C-Mesh and the C-Point, as well as to ensure the systematic storage of data in a database. The chosen implementation language was Python, due to its extensive ecosystem of libraries for MQTT communication, database interaction, and data processing, as well as its simplicity and flexibility for rapid prototyping. Python is particularly well-suited for edge computing scenarios, where lightweight scripts must reliably collect, transform and

forward data under resource constraints.

5.1.1 Relay Node

Before developing and testing the relay mechanisms proposed in this work, it was first necessary to set up the LoRaWAN infrastructure components in the C-Mesh that serve as the foundation of the system. This includes the gateway packet forwarder, the ChirpStack Gateway Bridge, the MQTT broker and the ChirpStack offline Network Server.

The Semtech UDP Packet Forwarder together with the ChirpStack Gateway Bridge were used to connect the Dragino PG1302 concentrator to the local ChirpStack instance. Several alternatives could have been considered, such as using the ChirpStack Concentratord with the ChirpStack MQTT Forwarder, the Semtech UDP Packet Forwarder with the ChirpStack MQTT Forwarder, or the LoRa Basics Station in combination with either the ChirpStack MQTT Forwarder or the ChirpStack Gateway Bridge. The Semtech UDP Packet Forwarder was chosen due to its simplicity, easy configuration and extensive documentation. The ChirpStack Gateway Bridge was preferred over the MQTT Forwarder as it provides a more direct and well-supported integration with ChirpStack, while maintaining compatibility with the Semtech UDP protocol.

The packet forwarder software was obtained from the official Semtech SX1302_hal repository [44], which provides both the hardware abstraction layer (HAL) for the SX1302/SX1303 concentrators and the UDP packet forwarder. After cloning and compiling the repository, the LoRaWAN EU868 regional configuration was selected. The packet forwarder was launched using the provided configuration file `global_conf.json.sx1250.EU868`, which specifies the operating frequencies, channel plan and network server parameters. In the `gateway_conf` section, the `server_address` was set to `localhost` and both the uplink and downlink ports were configured to `1700` to forward data to the ChirpStack Gateway Bridge. The `gateway_ID` was defined using the identifier obtained with the `util_chip_id` tool. With this configuration in place, the packet forwarder establishes a UDP connection to the LoRaWAN Network Server and relays both uplink and downlink frames (Figure 5.1. The full configuration of the packet forwarder can been seen in Appendix B.

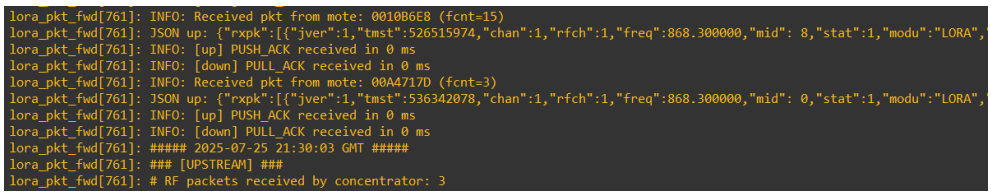


Figure 5.1 Semtech UDP Packet forwarder receiving uplinks.

Next, the ChirpStack Gateway Bridge was installed, as it is required to translate the Semtech UDP protocol into MQTT messages that can be consumed by the Network Server. ChirpStack provides a Debian package repository compatible with Raspberry Pi OS in the official guide [54]. Once the repository was configured, the Gateway Bridge was installed using `apt install chirpstack-gateway-bridge`. After installation, the `chirpstack-gateway-bridge` ser-

vice was available via `systemd`. By default, it listens on UDP port `1700` for Semtech packet forwarder traffic and publishes uplink and downlink frames to the local MQTT broker, which are then consumed by the ChirpStack Network Server.

To interconnect the ChirpStack Gateway Bridge with the Network Server, an MQTT broker was required. The open-source Mosquitto broker was installed on the Raspberry Pi with `apt install mosquitto mosquitto-clients`. By default, Mosquitto runs as a system service and listens on `tcp://127.0.0.1:1883`, which was configured in the ChirpStack Gateway Bridge configuration file so it connects to this local broker. In Appendix A, the full MQTT and packet-forwarder backend integrations with the gateway bridge are provided. With this setup, the Gateway Bridge is able to publish uplink frames and subscribe to downlink commands through the local Mosquitto broker, which acts as the communication hub for the ChirpStack Network Server, as illustrated in Figure 5.2 using MQTT Explorer tool [56].

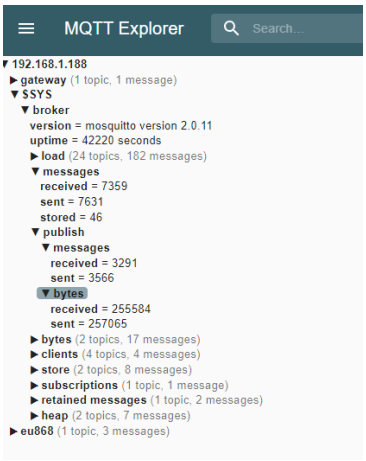


Figure 5.2 Local Mosquitto broker topics from the ChirpStack Gateway Bridge.

The next step was to install the local chirpstack network server that will be running offline in C-Mesh, the full guide is available on [55]. Chirpstack requires some software dependecies in order to function like This includes the Mosquitto MQTT broker and client utilities (already installed), Redis (for in-memory storage), and PostgreSQL (for persistent storage). After installing PostgreSQL, the database and user for ChirpStack were created. The ChirpStack Network Server itself was installed with `sudo apt install chirpstack`. The configuration files are stored in `/etc/chirpstack`, where `chirpstack.toml` provides global parameters and region-specific files contain the LoRaWAN regional band settings. With this setup, the ChirpStack Network Server was fully operational on the Raspberry Pi and could be accessed through its local IP on port `1883`.

Two C-beacon devices were registered in the ChirpStack NS frontend (Figure 5.3) by providing the required keys. One device (`C-Beacon`) was configured with ABP activation and used to perform field tests, while the other device (`C-Beacon2`) was configured with OTAA activation for validation purposes.

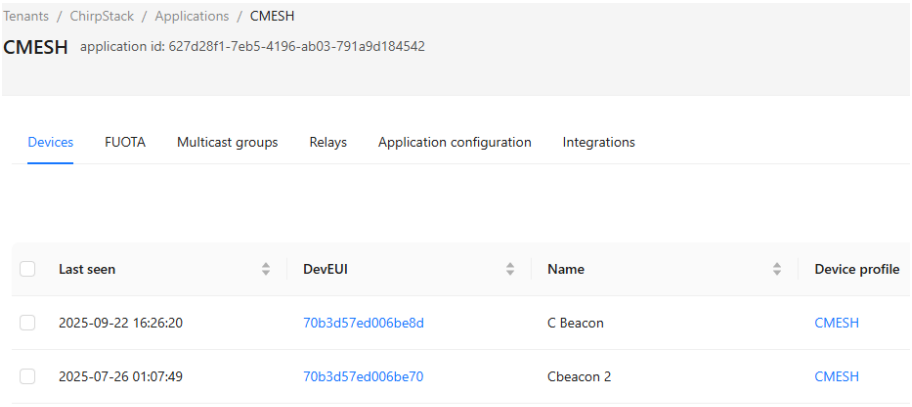


Figure 5.3 C-Beacon devices registration in the local ChirpStack NS frontend.

The C-Mesh (Raspberry Pi with the concentrator) was also registered as a gateway in the local ChirpStack frontend. This step was required because ChirpStack needs each packet forwarder to be associated with a registered gateway entry. By registering the C-Mesh as a gateway, the system is able to correctly map uplink and downlink traffic to this concentrator, monitor its status and manage the integration of the relay software with the LoRaWAN infrastructure.

5.1.2 Virtual Machine

On the VM side, a ChirpStack Network Server instance and a Mosquitto broker were installed and configured in the same way as described in Section 5.1.1, except that in this case the ChirpStack NS was assigned a fixed IP address and remained permanently online. Unlike the previous setup, the Gateway Bridge was not installed on the gateway itself, since the Gateway already has the packet forwarder. Instead, the Gateway Bridge was installed on the VM in order to translate the Semtech UDP traffic from the Gateway into MQTT messages for the Network Server. This VM instance acted as the main Network Server, essential for managing the LoRaWAN infrastructure.

The Mikrotik gateway presented in Section 4.2 was registered in this Network Server so that it could receive uplinks from the repeater and the C-Beacons. In addition, the C-Beacons were registered again in this ChirpStack instance and this time the C-Point was also registered, since it was now required to process uplinks generated by the C-Point device (Figure 5.4).

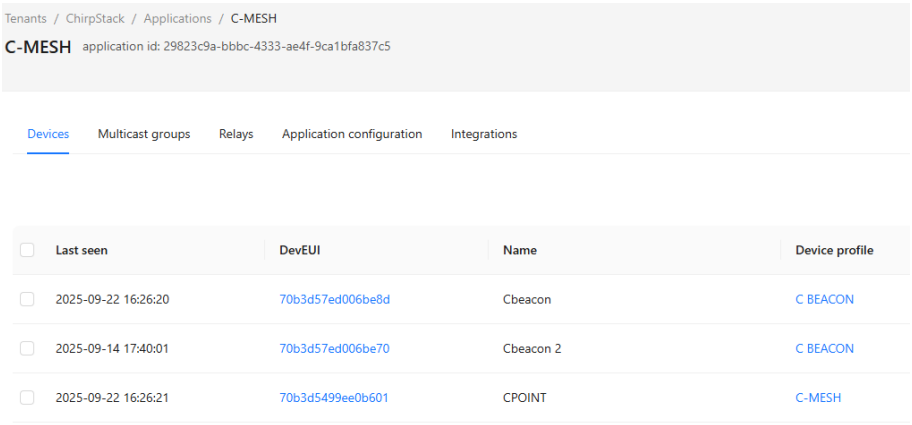


Figure 5.4 C-Beacon and C-Point devices registration in the online ChirpStack NS frontend.

After setting up the repeater, the gateway, and the Network Server, the Application Server was installed, as it represents an essential part of any IoT application. Before implementing the relay software, it was necessary to deploy a database to store the data sent by the C-Beacons and the C-Point. While a visualization dashboard was not strictly required, it provided a useful and user-friendly way to monitor the system in real time. For this purpose, CrateDB was selected as the storage backend and Grafana as the visualization tool.

CrateDB [57] is an open-source SQL database optimized for time-series and IoT data. It combines the scalability of modern databases with the simplicity of using standard SQL queries, making it particularly well-suited for storing large volumes of IoT data. In this work, CrateDB was chosen because it supports efficient storage of time-series data and allows straightforward queries for later analysis. It was installed on the VM using the official Debian/Ubuntu packages and configured to run as a system service on its default ports. The CrateDB service was made available at the virtual machine's IP address on port 4200.

Grafana [58] is an open-source analytics and visualization platform widely used for monitoring and IoT data exploration. It provides a web-based interface for creating real-time dashboards from different data sources, including SQL databases such as CrateDB. Grafana was chosen due to its flexibility, its strong community support and its ability to directly connect to CrateDB through a dedicated plugin. It was installed from the official Debian/Ubuntu repository and configured as a service on the VM. The Grafana service was made available at the virtual machine's IP address on port 3000.

In order to enable the visualization of experimental data, Grafana was connected to the CrateDB database. The connection was established by configuring CrateDB as a data source in Grafana, specifying the database host (the virtual machine's IP address), the port exposed by CrateDB (default 5432 for PostgreSQL), as well as the database name and authentication credentials. Once the data source was successfully added, SQL queries could be written directly in Grafana panels using the PostgreSQL query language supported by CrateDB. This integration enabled the creation of dashboards to visualize uplinks, device activity and relay performance

metrics.

5.2 Implementation

The software was developed in Python [61], a high-level interpreted language known for its simplicity, readability, and versatility. Its extensive libraries make it a popular choice for prototyping, as they streamline development and reduce the amount of code required.

5.2.1 Relay Software

This section describes the C-Mesh edge software modules and how they cooperate to ingest LoRaWAN uplinks, perform edge processing and relay data to C-Point. This implementation happens between the C-Mesh which contains the ChirpStack NS, MQTT broker and the C-Point (TTGO) over serial, implementing two complementary paths:

- **Outbound (MQTT broker → C-Mesh → C-Point)**: Uplink frames received from ChirpStack over MQTT are compacted into a binary payload and sent to C-Point using a CMESH:01 message. This path enforces ACK-based reliability and waits for a radio TX result (TXOK/TXERR) per message.
- **Inbound (C-Point → C-Mesh → ChirpStack DB)**: C-Point sends OTAA session updates (AppSKey/ NwkSKey) using CMESH:02 lines over serial. The C-Mesh ingests these, validates fields and commits the session keys into the ChirpStack database and then it returns an OTAA ACK/NACK to the C-Point.

The system is organized into modules, each with a distinct responsibility. Figure 5.5 illustrates the interactions between these modules.

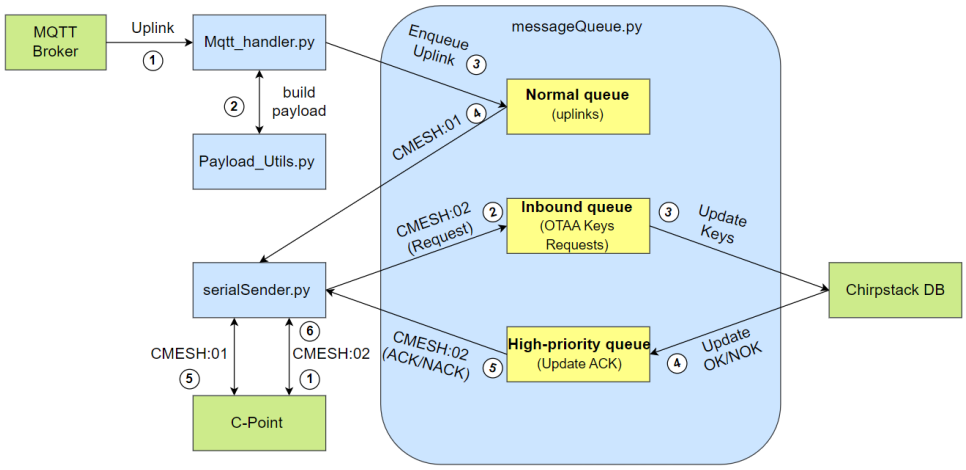


Figure 5.5 Software architecture: modules and their interactions.

The following list details each module's responsibilities:

- **mqtt_handler.py**: Manages communication with the MQTT broker. It connects to the server, subscribes to uplink and downlink topics, and processes incoming uplink messages. Each message is parsed, converted into a compact binary format using the `payload_utils` module, and forwarded to the `MessageQueue` module.
- **messageQueue.py**: Acts as the central coordinator between MQTT and serial communication. Its main function is to manage multiple queues: high-priority (OTAA acknowledgements), normal (C-Beacon messages requiring ACK) and inbound (OTAA update requests).
- **serialSender.py**: Provides robust serial communication with the C-Point. It manages port opening, closing and automatic reconnection, runs a background reader thread to capture line-oriented messages and ensures thread-safe writes with error handling.
- **config.py**: Centralizes all configuration parameters, including serial port settings, MQTT server details, subscribed topics, CMESH protocol constants, timeouts, retry limits and debug mode. It also defines the `cmesh_new_id()` function for generating unique message identifiers.
- **payload_utils.py**: Implements the compact binary message format used to transport uplink frames across the C-Point.
- **UpdateOTAAKeys.py**: Integrates with the ChirpStack database. Its `insert_device_session()` function updates device session records with new AppSKey, NwkSKey and DevAddr values received from the C-Point via BLE.

The system relies on multiple concurrent threads to ensure responsiveness and fault tolerance. Each thread has a well-defined role in handling I/O, processing and coordination, as summarized below:

- **Main thread** — Initializes all components, including startups and shutdowns.
- **MQTT network thread** — Handles socket I/O and MQTT callbacks (`on_message`, `on_connect`, `on_disconnect`).
- **Serial reader thread** (`SerialSender.start_reader`) — Continuously performs line-oriented reads from the serial port with automatic reconnection on errors.
- **Sender worker** (`MessageQueue._run_sender_worker`) — Processes outbound queues, manages acknowledgements and retries failed transmissions.
- **Processor worker** (`MessageQueue._run_processor_worker`) — Handles inbound OTAA messages by updating the database and enqueueing the corresponding OTAA acknowledgement.

The main thread starts all components and coordinates a clean shutdown. MQTT networking runs on the Paho loop thread, which receives uplinks, converts them into the compact payload, and enqueues them without blocking. Serial I/O is handled by a dedicated serial reader thread that streams lines from the C-Point and triggers callbacks. On top of that, the message pipeline is split into two worker threads inside ‘MessageQueue‘: a sender worker that pulls from the outbound queues (prioritizing OTAA ACKs over C-Beacon messages) and enforces the ACK/Tx-result handshake with retries, and a processor worker that consumes inbound OTAA requests, updates the ChirpStack database and schedules the corresponding OTAA acknowledgement. This separation makes sure inbound processing never blocks outbound reliability.

The `MessageQueue` module is the central coordinator, managing all queues and communication between C-Mesh and the C-Point. It maintains three distinct queues:

- **`outbound_normal_queue`** — Handles C-Beacon uplink messages (C-Mesh → C-Point). These transmissions require an acknowledgement (ACK) from the C-Point, which is managed by the sender worker.
- **`outbound_high_priority_queue`** — Used for OTAA acknowledgements. This queue has higher priority because, when an OTAA key update request is received, the keys must be updated immediately so that the local ChirpStack can accept messages from the device without blocking them. This ensures that no uplink messages are lost.
- **`inbound_otaa_queue`** — Collects OTAA update tasks (C-Point → C-Mesh). The processor worker consumes these tasks, writes the new keys to the ChirpStack database and then schedules an OTAA ACK by placing it into the `outbound_high_priority_queue`.

Figure 5.6 illustrates the execution of the system when processing an uplink message from a C-Beacon. The MQTT client receives a message from the ChirpStack Gateway Bridge containing metadata such as device address, RSSI, SNR, spreading factor and payload. The message is then parsed and enqueued into the `MessageQueue` as a C-BEACON uplink task. Subsequently, the `serialSender` component formats the message according to the CMESH protocol and transmits it over the serial interface to the C-Point (`WRITE` operation). The `MessageQueue` supervises this transmission, awaiting an acknowledgement. The C-Point then replies with `ACK:04500B8B`, confirming successful reception. The system retries transmission if no ACK is received, but in this case the first attempt succeeds. Finally, the figure 5.6 confirms that the uplink message is delivered through the C-Mesh layer (`TTGO → PC [CPOINT] LoRa Sent`) and that the corresponding RX message is correctly returned. This sequence demonstrates the integration of MQTT handling, message queue management, serial transmission and acknowledgement processing, ensuring reliable delivery of uplink messages from C-Beacon devices into the C-Mesh infrastructure.

Figure 5.6 Execution of a C-Beacon uplink procedure.

BLE and OTAA keys injection

The app nRF Connect for mobile is provided for free and is available on Google's PlayStore [64]. After downloading and starting the app the app is immediately scanning for BLE devices and will show a device in the Scanner tab. In Figure 5.7 it is possible to see the C-Point device ready to connect.

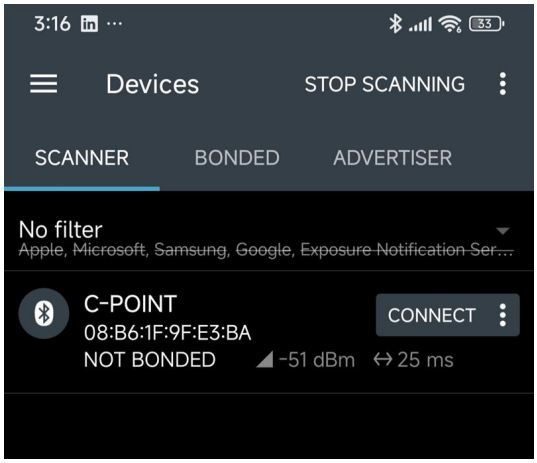


Figure 5.7 C-Point device shown in the nRF Connect scanner tab.

After connecting, we can see general services such as Generic Attribute and Generic Access, but also the C-Mesh Update OTAA Keys service exposed by the C-Point as shown in Figure 5.8.

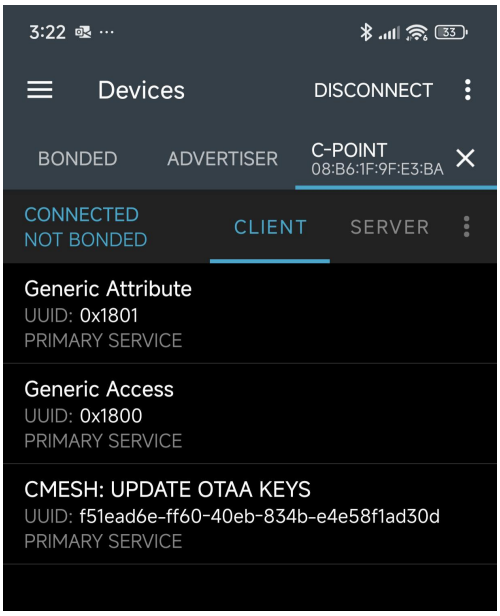


Figure 5.8 BLE services and characteristics exposed by the C-Point, including C-Mesh Update OTAA Keys.

This service has two characteristics: one for the C-Point to receive data from the app (RX, via write) and another for the C-Point to send data back (TX, via read/notify). If the user tries to write something other than `CMESH:02` (Figure 5.9a), the C-Point will not accept it and replies with `ERR:Format` (Figure 5.10a) , indicating that the message format is wrong. However, if the user writes `CMESH:02` (Figure 5.9b), the C-Point accepts it and replies with an `ACK` (Figure 5.10b), meaning it was successfully received.

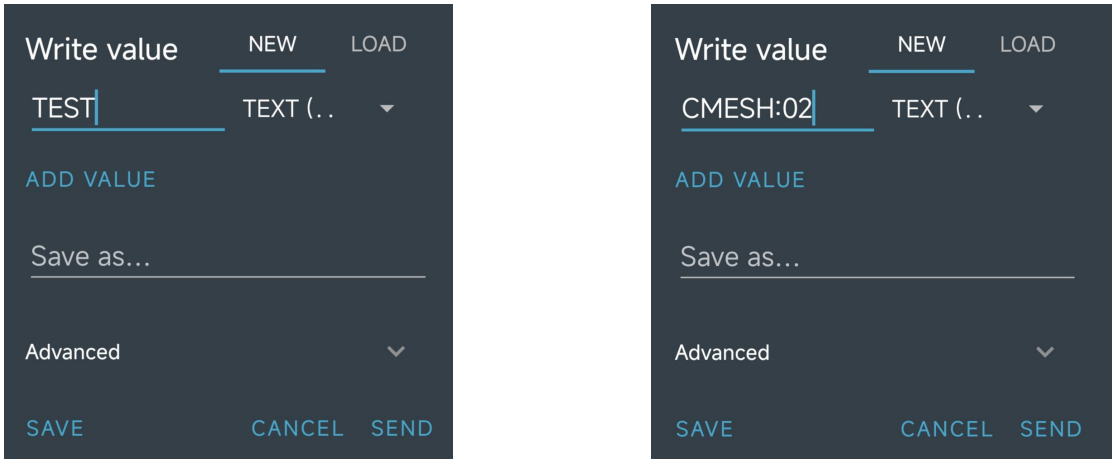


Figure 5.9 Write operations tested with nRF Connect: (a) invalid format; (b) valid CMESH command.

It is important to note that only the command `CMESH:02` is wrote without including the session keys themselves. Sending the entire OTAA keys manually would be impractical and insecure, as they would pass unencrypted over BLE. Instead, once the C-Point receives `CMESH:02`, it appends a pre-stored key set to the message and forwards it via serial to C-Mesh, which then

updates the ChirpStack database. This serves only as a proof of concept.

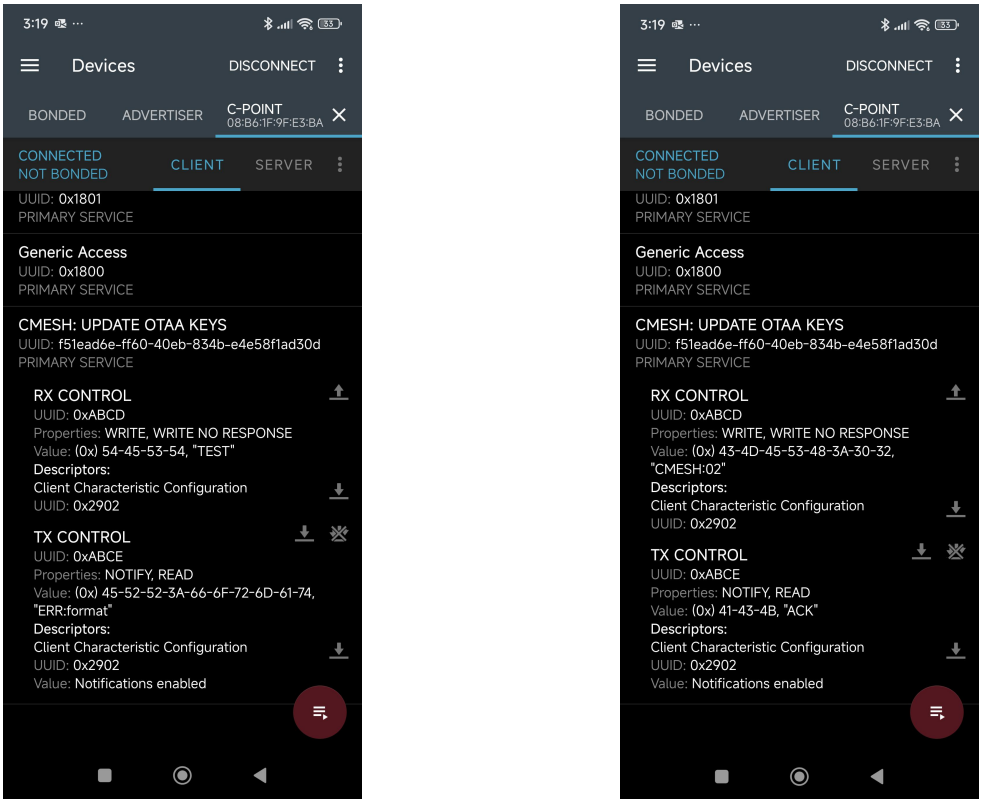


Figure 5.10 Responses from the C-Point after write attempts: (a) invalid format, rejected; (b) valid CMESH command, accepted.

To update OTAA keys, Protocol Buffers (Protobuf) [63] are required because ChirpStack stores the active LoRaWAN session in a table device under the column device_session as a serialized DeviceSession message (binary Protobuf), not as decomposed SQL columns. As shown in Fig. 5.11, the device_session column is of type BYTEA, confirming that the session is persisted as opaque binary rather than text/JSON.

```
chirpstack=> \d+ device
                                                Table "public.device"
       Column       |           Type           | Collation | Nullable | Default | Storage  | Stats target | Description
--------------------+--------------------------+-----------+----------+---------+----------+--------------+-------------
 dev_eui            | bytea                    |           | not null |         | extended |              |
 application_id     | uuid                     |           | not null |         | plain    |              |
 device_profile_id  | uuid                     |           | not null |         | plain    |              |
 created_at         | timestamp with time zone |           | not null |         | plain    |              |
 updated_at         | timestamp with time zone |           | not null |         | plain    |              |
 last_seen_at       | timestamp with time zone |           |          |         | plain    |              |
 scheduler_run_after | timestamp with time zone |           |          |         | plain    |              |
 name               | character varying(100)   |           | not null |         | extended |              |
 description        | text                     |           | not null |         | extended |              |
 external_power_source | boolean                |           | not null |         | plain    |              |
 battery_level      | numeric(5,2)             |           |          |         | main     |              |
 margin             | integer                  |           |          |         | plain    |              |
 dr                 | smallint                 |           |          |         | plain    |              |
 latitude           | double precision         |           |          |         | plain    |              |
 longitude          | double precision         |           |          |         | plain    |              |
 altitude           | real                     |           |          |         | plain    |              |
 dev_addr           | bytea                    |           |          |         | extended |              |
 enabled_class      | character(1)             |           | not null |         | extended |              |
 skip_fcnt_check    | boolean                  |           | not null |         | plain    |              |
 is_disabled        | boolean                  |           | not null |         | plain    |              |
 tags               | jsonb                    |           | not null |         | extended |              |
 variables          | jsonb                    |           | not null |         | extended |              |
 join_eui           | bytea                    |           | not null |         | extended |              |
 secondary_dev_addr | bytea                    |           |          |         | extended |              |
 device_session     | bytea                    |           |          |         | extended |              |
 app_layer_params   | jsonb                    |           | not null |         | extended |              |
```

Figure 5.11 Chirpstack `device` table schema.

This binary data encodes the session keys, uplink and downlink framecounters, LoRaWAN MacVersion, region-specific RX parameters and ADR/relay metadata, all of which must match the exact wire format defined by ChirpStack's .proto files. Fig. 5.12 illustrates the stored blob content at the SQL level, which is expected for Protobuf serialization.

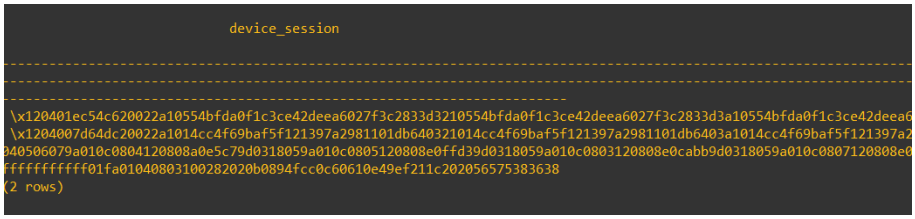


Figure 5.12 Content of `device_session` showing Protobuf-serialized session data.

ChirpStack publishes the relevant proto schemas in its repository [62]: the internal.proto file (which defines DeviceSession) depends on common.proto and gw.proto. These files specify how an active LoRaWAN session—DevAddr, session keys, counters, RX settings, ADR state, relay state, and histories—is structured and serialized for storage and inter-component exchange.

First, the Protobuf bindings are prepared: 'protoc' is installed, the ChirpStack schema files ('internal.proto', 'common.proto', 'gw.proto') were downloaded, and Python modules are generated ('internal/internal_pb2.py', 'common/common_pb2.py', 'gw/gw_pb2.py'). The injection script connects to the ChirpStack database, targets the row by 'dev_eui' and builds an 'internal_pb2.DeviceSession' populated with the session material obtained from the online server: 'DevAddr', 'NwkSKey' and 'AppSKey'. The result is that your offline ChirpStack instance has a complete, usable session—as if the device had just joined via OTAA—using the AppSKey/NwkSKey/DevAddr you obtained from the online server.

Figure 5.13 illustrates the execution of an OTAA update procedure. After receiving a CMESH frame from the C-Point containing new OTAA session information, the system logs the line `RX CMESH:02:55AA33CC:...`, which shows that the payload includes the device EUI, DevAddr, AppSKey and NwkSKey. The `MessageQueue` module identifies the frame as an inbound OTAA update and enqueues it for processing. The processor worker then extracts the session data and updates the ChirpStack database with the new keys, as confirmed by the line `[OTAA] session updated ...`. This ensures that ChirpStack will accept future uplinks from the device without rejecting them due to outdated session keys. Once the update is complete, the system prepares an OTAA acknowledgement (`Queued OTAA ACK`) and sends it via the high-priority outbound queue. The `serialSender` formats this as a CMESH message (`WRITE 23 bytes: b'CMESH:02:ACK:55AA33CC'`) and delivers it to the C-Point. Finally, the C-Point confirms reception with `OTAA ACK received`, completing the update cycle.

This sequence demonstrates the end-to-end handling of OTAA key updates: from reception of the CMESH frame, through database synchronization, to reliable acknowledgement back to the C-Point, ensuring secure and seamless device reactivation in the network.

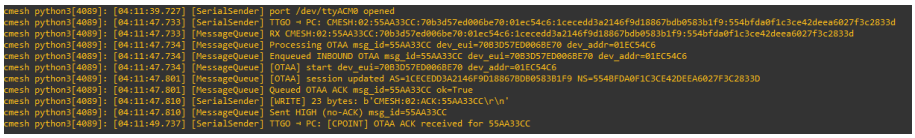


Figure 5.13 Execution of an OTAA Keys update procedure.

5.2.2 VM Software

When the ChirpStack NS receives an uplink, either from a C-Beacon or a C-Point, the associated metadata can be inspected in the message structure (Figure 5.14). Within the `deviceInfo` field, the application name, device name and device EUI are included, identifying the source of the message. Outside the `deviceInfo` block, additional LoRaWAN parameters are provided, such as the device address (`dev_addr`), the adaptive data rate flag (`adr`), the frame counter (`fcnt`), the frame port (`fport`), the confirmation status (`confirmed`) and the encoded payload (`data`). The `rxInfo` block stores reception metrics reported by the gateway, including the RSSI, SNR, reception timestamp, gateway identifier and the geographical location of the gateway. The `txInfo` block provides the transmission parameters used by the device, such as spreading factor (SF), frequency, bandwidth and coding rate, which characterize the LoRa physical layer settings of the uplink. The `object` field contains the decoded payload, which is processed by the decoder defined in ChirpStack and presented in Appendices C and D. The decoded structure differs for the C-Point and C-Beacon, as presented in Figure 5.14a and 5.14b, respectively.

Figure 5.14 Uplinks observed in the ChirpStack NS: (a) C-Point with an encapsulated C-Beacon message and (b) C-Beacon message transmitted without the repeater.

It can be observed that the C-Beacon coordinates are encapsulated within the C-Point payload in the object field, along with other metrics such as RSSI, SNR, devAddr, SF, fCnt and fPort. This indicates that the original C-Beacon message was successfully repeated by the C-Point.

On the VM there is a background service called Mqtt_CrateDB, implemented as a Python script. This script acts as a bridge between the online ChirpStack NS instance and the CrateDB database, using the MQTT protocol to ingest uplink messages. The paho-mqtt library [59] is responsible for subscribing to the Mosquitto broker,listening to the application uplink topics where C-Beacons and C-Point devices are registered. When an uplink is received, the script extracts metadata such as device address, device eui, frame counters, timestamp, data payload, RSSI, SNR, spreading factor, frequency, bandwidth and gateway information. In addition, it computes the “best gateway” by selecting the one with the highest SNR. The script then connects to CrateDB using the crate.client library [60]. If the table cmesh does not exist, it is created with the schema described in Table 5.1. Each new uplink is then inserted into this table, storing both parsed fields and the full JSON uplink for future reference.

Table 5.1 Schema of the `cmesh` table in CrateDB.

Field	Type	Description
timestamp	TIMESTAMP	Time of message reception.
message	TEXT	Hexadecimal representation of the application payload.
dev_eui	TEXT	Unique device identifier (DevEUI).
dev_name	TEXT	Human-readable device name.
dev_addr	TEXT	Device address assigned during session.
f_port	INTEGER	LoRaWAN frame port.
f_count	INTEGER	LoRaWAN frame counter.
best_gateway	TEXT	Identifier of the gateway with the best SNR.
rssi	INTEGER	Received Signal Strength Indicator (dBm).
snr	DOUBLE	Signal-to-Noise Ratio (dB).
sf	INTEGER	Spreading factor used for the transmission.
frequency	LONG	Transmission frequency (Hz).
bandwidth	LONG	Transmission bandwidth (Hz).
gateways	ARRAY(OBJECT)	Metadata from all gateways that received the packet.
dump	OBJECT	Complete JSON payload of the uplink message.

This integration ensures that all LoRaWAN uplinks are systematically logged in a structured format, facilitating advanced queries, performance evaluation and dashboard visualizations. Figure 5.15 illustrates the Grafana dashboard developed to monitor the performance of the system, focusing on the C-Beacon and C-Point devices.

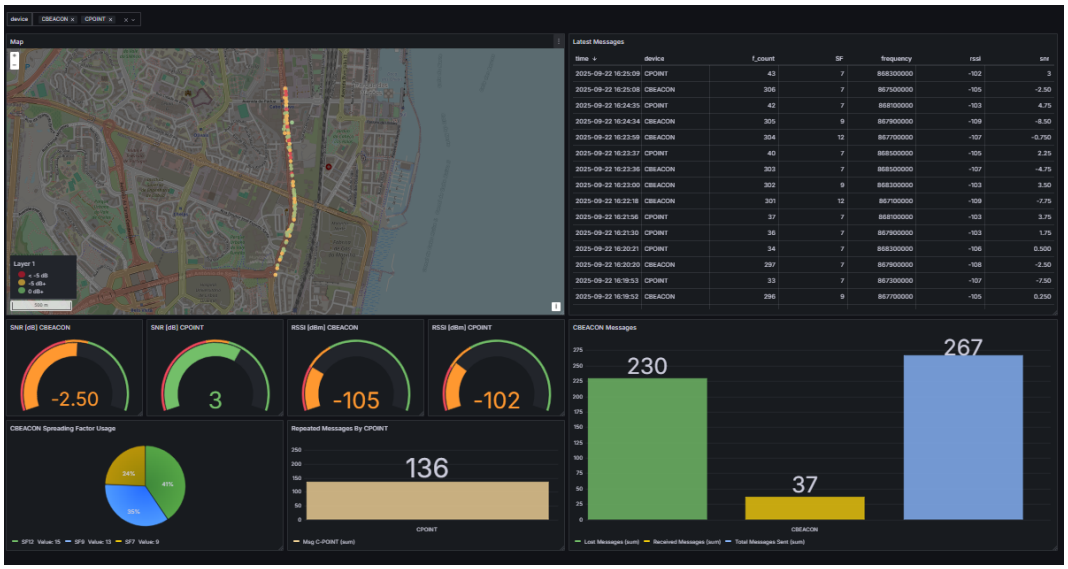


Figure 5.15 Grafana dashboard of the project.

In the upper-left panel, a georeferenced map displays the reported positions of the devices, color-coded according to the measured signal-to-noise ratio (SNR). Green markers represent SNR values above 0 dB, yellow markers indicate values between −5 dB and 0 dB, and red markers highlight poor signal quality below −5 dB. The upper-right side of the dashboard presents a table of the most recent messages received, including timestamp, device name, frame count, spreading factor, frequency, received signal strength indicator (RSSI) and SNR. This enables quick inspection of the latest communication events. The central row of panels contains gauge visualizations that show the latest values of SNR and RSSI for both C-Beacon and C-Point devices. Thresholds are applied to these gauges to provide immediate feedback on signal quality and link reliability. Below the gauges, additional panels summarize key performance indicators. A pie chart shows the distribution of spreading factors used by the C-Beacon device. A bar panel quantifies the number of repeated messages processed by the C-Point relay, highlighting the role of the edge node. Finally, a comparative bar chart presents the total number of messages sent, successfully received and lost from the C-Beacon without the repeater.

Overall, this dashboard provides an integrated view of the LoRaWAN network, supporting both real-time monitoring and post-deployment performance analysis. It is essential for visualizing the project results and proves particularly valuable during field testing.

5.3 Challenges

During the development and integration of the proposed system, some challenges were encountered related to the OTAA key storage in chirpstack, serial communication between the C-Mesh and C-Point, and the BLE interface used for key synchronization between edge and backend components. This section summarizes the main issues faced and the corresponding solutions adopted.

- **ChirpStack Database and Session Management:** A significant challenge was when attempting to inject OTAA session keys (AppSKey and NwkSKey) directly into the offline ChirpStack instance. Since ChirpStack database stores the active session as a serialized Protobuf object in a binary column (‘device_session‘), rather than in plain SQL fields, it made manual key synchronization impossible through SQL commands alone. To address this, the Protobuf schema files (‘internal.proto‘, ‘common.proto‘, ‘gw.proto‘) were compiled into Python bindings.
- **Serial Link Synchronization:** The serial communication between the C-Mesh and C-Point presented synchronization issues due to partial or fragmented reads. In early tests, incomplete CMESH frames occasionally led to invalid parsing or deadlocks in the serial buffer. This was solved by implementing a dedicated reader thread with line-oriented parsing and automatic reconnection. The introduction of thread-safe locks around write operations and explicit ACK/NAK feedback loops improved reliability and reduced transmission errors.
- **BLE/OTAA key updates:** On TTGO T-Beam v1.1, nRF Connect frequently reported `GATT ERROR 133 (0x85)` (and occasionally `GATT_CONN_FAIL_ESTABLISH (62)`) during service discovery or on first write, causing the link to drop before the OTAA characteristic could be used. This behaviour was not observed on v1.2, where discovery and writes executed reliably.

Chapter 6

Evaluation and Results

The experimental tests aimed to evaluate the effectiveness of the proposed repeater in extending LoRaWAN coverage. Both laboratory and field experiments were conducted to characterize the system's performance under controlled and real-world conditions. The laboratory tests focused on validating the correct operation of the developed hardware and software components, while the field tests assessed the impact of the repeater on communication range, message reliability and signal quality in a real urban environment in the city of Lisbon.

6.1 Objectives

The experimental phase of this work aimed to validate the proposed relay system in both controlled and real environments, ensuring that each functional and performance-related objective defined in Chapter 1.3 was experimentally verified. The tests were designed to confirm correct system operation, analyze network performance, and identify the benefits and limitations introduced by the relay mechanism. Table 6.1 summarizes the specific objectives, the corresponding tests performed, the metrics used for evaluation and the minimum success criteria adopted for each case.

Table 6.1 Summary of objectives, performed tests, metrics and results.

Objective	Test Performed	Metric	Success Criteria
Implement and evaluate a relay-based system for LoRaWAN.	Functional validation (C-Beacon → C-Mesh → C-Point → Gateway).	Successful message forwarding (uplink).	Able to see C-Beacon messages encapsulated in C-Point payload.
Ensure compatibility with commercial LoRaWAN devices.	Perform tests with devices used in the market.	Validate sucess with simple LoRaWAN messages.	Able to receive messages in Chirpstack.
Develop and validate a key synchronization mechanism between the relay node and the central platform.	Session key extraction from Online Chirpstack instance.	Session Key injection in offline ChirpStack instances.	Key match verification (AppSKey, NwkSKey) between the 2 chirpstack instances.
Evaluate coverage improvement with relay operation.	Field test along 2.2 km urban route.	PDR, coverage gain.	Average range increase $\geq$ 30%.
Assess link quality improvement.	Comparative analysis of received SNR/RSSI with and without relay.	SNR and RSSI distributions.	PDR should increase ≥20% in weak-signal conditions (RSSI < –115 dBm or SNR < 0 dB).
Determine optimal spreading factor for relay transmission.	C-Point → Gateway tests using SF7, SF9, SF12.	Frame success rate per SF.	Selected SF achieves $\geq$ 90% delivery under test conditions.
Demonstrate scalability and processing capability.	Stress tests with increasing message rates (20–40 msg/s).	Average ACK latency, delivery stability.	Should remain stable and sustain the load without failures or message loss.

6.2 Performance metrics

To evaluate the impact of the repeater on LoRaWAN communication, a set of performance metrics were defined to allow a direct comparison between scenarios with and without the repeater. These metrics provide both quantitative and qualitative insights into the quality of the radio link and the reliability of message delivery and are commonly used in wireless communication studies.

- **Signal-to-Noise Ratio (SNR):** Represents the ratio between the received signal strength and background noise. Higher values indicate better link quality. In LoRaWAN, successful reception is possible even with negative SNR values, depending on the spreading factor.
- **Received Signal Strength Indicator (RSSI):** Indicates the power of the received signal, measured in dBm. Values closer to 0 correspond to stronger signals, while values below –110 dBm are close to the sensitivity limit of typical LoRa transceivers.
- **Messages Transmitted and Received:** The total number of uplink messages sent by the C-Beacon and the number successfully received at the gateway.

- **Packet Loss Rate (PDR):** Calculated as the percentage of transmitted messages that did not reach the gateway.
- **Spreading Factor (SF):** As explained in Section 2.1.1, it represents the trade-off between data rate and coverage in LoRaWAN.

6.3 Controlled Tests

Before conducting the field experiments, laboratory tests were carried out to validate the integration of the C-Mesh and C-Point devices and to confirm the functional behavior of the repeater. This ensured that any limitations subsequently observed in the field could be attributed to radio propagation and urban conditions rather than implementation issues.

Before conducting the field experiments, laboratory tests were performed to validate the integration of the C-Mesh and C-Point devices and to confirm the functional behavior of the repeater. This preliminary tests assured that any limitations subsequently observed in the field could be attributed to radio propagation and urban conditions rather than implementation issues.

Table 6.2 Summary of controlled laboratory tests.

Test No.	Test Description
1	Gateway and Network Server connectivity (ChirpStack Online)
2	Device connectivity (C-Beacon $\rightarrow$ Gateway)
3	C-Mesh reception (C-Beacon $\rightarrow$ C-Mesh)
4	Serial communication (C-Mesh $\rightarrow$ C-Point)
5	C-Point payload encapsulation
6	Relay functionality (C-Beacon $\rightarrow$ C-Mesh $\rightarrow$ C-Point $\rightarrow$ Gateway)
7	C-Point disconnection recovery
8	C-Point and C-Mesh communication flow validation
9	Application server integration (MQTT, CrateDB, Grafana)
10	OTAA key distribution via BLE

In addition to the functional validation performed under controlled conditions, stress tests were conducted to assess the scalability and processing capability of the relay system before proceeding to the field experiments. These tests aimed to verify the stability of the message queue, the serial communication throughput between the C-Mesh and C-Point, and the ability of the system to maintain low latency and avoid message loss under high message production rates. The LoRawan part on C-Point was deactivated in order to not overload the network and be complaint with duty-cycle regulations.

The stress tests were executed using incremental and burst traffic patterns. For the incremental tests (`test1`), the C-Mesh generated and sent an increasing number of messages (10 to 400) at rates of 10 msg/s and 20 msg/s, while the C-Point acknowledged each received frame. These rates were considered sufficient since LoRaWAN end-devices are constrained by regional duty-cycle regulations and typically operate well below these message frequen-

cies. For the burst tests (`test2`), traffic was produced in consecutive groups of 40 messages per burst, repeated five times, to evaluate the system's recovery and buffer handling between transmissions. Table 6.3 summarizes the stress test configurations and outcomes.

Table 6.3 Summary of stress test configurations and results.

Test ID	Rate (msg/s)	Messages (writes/acks)	Avg. ACK Latency [s]	Duration [s]	Observation
`test1.1`	10	20 / 20	2.000	40.11	No loss.
`test1.2`	10	40 / 40	2.001	80.16	No loss; Steady ACK latency.
`test1.3`	10	80 / 80	2.002	160.29	No loss; Steady ACK latency.
`test1.4`	10	200 / 200	2.002	400.69	No loss; Stable under high load.
`test1.5`	10	400 / 400	2.002	801.37	No loss; Stable under high load.
`test1.6`	20	40 / 40	2.001	80.17	Doubled rate handled successfully.
`test2 (bursts)`	10	200 / 200 (5×40)	2.000	403.7	All bursts acknowledged; 0 losses; smooth queue recovery.

Across all test cases, the system maintained a consistent acknowledgment latency of approximately 2.0 s, even at the highest load of 400 messages and doubled transmission rate of 20 msg/s. No message loss, buffer overflow, or serial communication error was detected during the experiments. The results confirm that the relay system can sustain continuous and burst traffic without degradation, demonstrating adequate scalability and processing capability for deployment in denser IoT scenarios.

Across all test cases, the system maintained a consistent acknowledgment latency of approximately 2.0 s, even under the highest load of 400 messages and the doubled transmission rate of 20 msg/s. The total test durations scaled linearly with the number of transmitted messages — from roughly 40 s for 20 messages to about 800 s for 400 messages — indicating stable processing time per message and absence of queue buildup. No message loss, buffer overflow, or serial communication errors were detected during the experiments. Overall, the results confirm that the relay system can sustain continuous and burst traffic for extended periods without degradation, demonstrating adequate scalability, timing stability, and processing capability for deployment in denser IoT scenarios.

6.4 Field Tests

Before conducting the field measurements, it is necessary to describe all the components required for the experiments. Two types of field tests were planned: one without the repeater and

another with the repeater installed. The first test aimed to assess the LoRaWAN gateway coverage along the predefined route in order to identify the optimal location for placing the repeater. The second test was performed with the repeater deployed at the selected position to evaluate its impact on network coverage and signal quality. The components used for the field tests are summarized below:

- **C-Mesh:** A Raspberry Pi 4 equipped with a Semtech SX1302 concentrator board, responsible for receiving uplink messages from the C-Beacon and managing local processing and forwarding logic. This unit was configured as a static device.
- **C-Point:** A TTGO LoRa device (SX1276 transceiver) used to retransmit the encapsulated messages received from the C-Mesh toward the LoRaWAN gateway. This unit was also configured as a static device and operated with three spreading factors (SF7, SF9, and SF12) to determine which configuration would provide the best performance for future relay deployments.
- **C-Beacon:** A TTGO LoRa device (SX1276 transceiver) acting as the end-device node. It was programmed to periodically transmit its GPS coordinates using LoRaWAN messages to evaluate the network coverage. Unlike the other units, the C-Beacon was a mobile device, carried along the predefined 2.2 km route.
- **Gateway:** Installed on the balcony of a first-floor apartment, serving as the main LoRaWAN gateway connected to the online ChirpStack network server.
- **Installation:** The C-Mesh and C-Point were mounted side by side inside the test vehicle, operating together as a single repeater node. The LoRa and GPS antennas were positioned on the vehicle's roof to ensure optimal signal reception and transmission conditions.

6.4.1 Gateway Coverage Evaluation

As said, to test the use of the developed LoRaWAN repeater in a real environment, it was first necessary to assess the gateway's coverage before deciding where to place the repeater. A 2.2 km route was defined along Avenida Infante Dom Henrique, as shown in Fig. 6.1. The gateway was installed on the balcony of a first-floor apartment (Fig. 6.1), where (a) indicates its position on the map and (b) shows the physical setup.

Although this location was far from ideal, no higher site with better coverage was available. It's also important to note that, a tall building directly in front of the gateway obstructed the line of sight, significantly degrading SNR and RSSI, otherwise the C-Beacon would have had a much clearer line of sight.

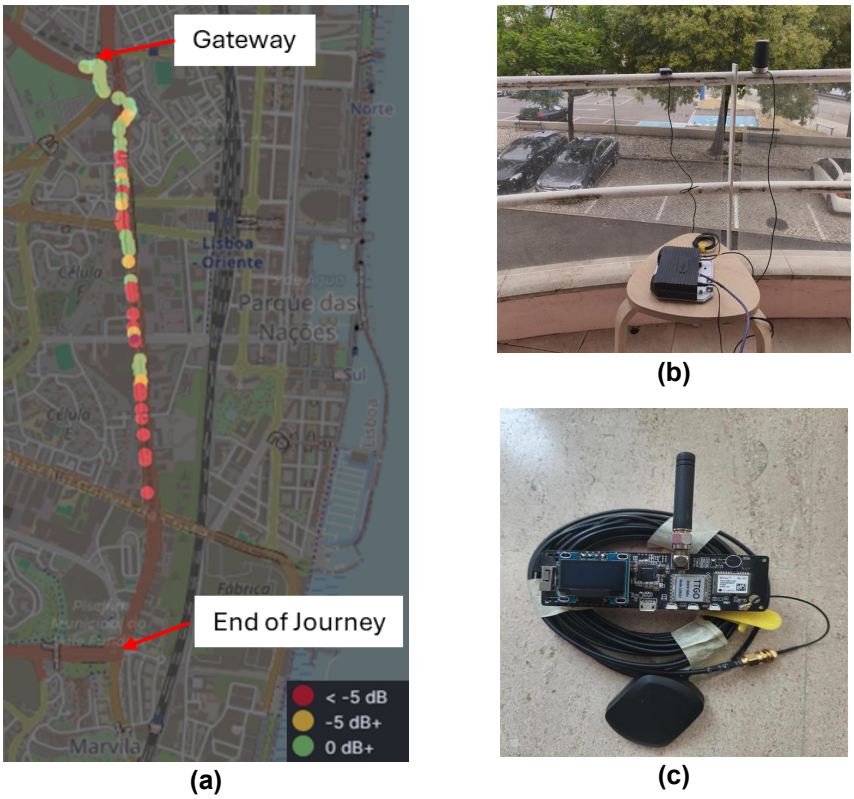


Figure 6.1 Gateway coverage results from field measurements (a) and equipment used: (b) Gateway placement and (c) device used as a C-Beacon.

As previously mentioned, the C-Beacon (Figure 6.1c) was configured to send LoRaWAN messages containing its current GPS coordinates along the route. Messages were transmitted using SF7, SF9 and SF12 to evaluate how different spreading factors influence delivery and reception.

During the test, the C-Beacon transmitted 110 messages: 64 were received and 46 were lost, representing a loss rate of 41.8% due to limited gateway coverage beyond a certain distance. Among the 64 received messages, almost half (47%) were transmitted with SF12, 30% with SF9 while only 23% used SF7. Table 6.4 presents the distribution of received messages by signal-to-noise ratio (SNR) and spreading factor (SF). At lower SNR values (< 0 dB), particularly in the < -5 dB range, SF12 and SF9 were the dominant configurations (12.5% each), showing that higher spreading factors were essential to maintain reliable communication under weak signal conditions. Only a small fraction of messages (12.5%) fell into the intermediate SNR range of [-5, 0] dB, again with SF12 being the most used (6.3%). Overall, SF12 was the most frequently selected spreading factor (46.9% of all messages), highlighting a preference in more challenging radio conditions. SF7, on the other hand, was used primarily in good SNR conditions, suggesting its use to maximize data rate when shorter distances or better channels allowed.

Table 6.4 Distribution of messages (Msg) received by SNR and SF.

SNR [dB]	SF7		SF9		SF12		Total	
	Msg	%	Msg	%	Msg	%	Msg	%
< -5	3	4.7	8	12.5	8	12.5	19	29.7
$[-5, 0]$	3	4.7	1	1.6	4	6.3	8	12.5
> 0	9	14.1	9	14.1	18	28.1	36	56.3
Total	15	23.4	19	28.1	30	46.9	64	100

Table 6.5 presents the distribution of received messages by received signal strength indicator (RSSI) and spreading factor (SF). At very weak signal levels (<−110 dBm), most of the messages relied on SF12 (12.5%), followed by SF9 (10.9%) and SF7 (6.3%), showing that higher spreading factors were essential to sustain communication under poor radio conditions. In the intermediate range of [−110,−90] dBm, SF12 again dominated (20.3%), while SF9 and SF7 accounted for 12.5% and 10.9%, respectively. Only 26.6% of the messages were received with stronger signals (>−90 dBm), whereas the majority (43.7%) were concentrated in the intermediate range.. Overall, SF12 was the most prevalent spreading factor (46.9% of all messages), reflecting its importance in maintaining reliable links under weak signal conditions.

Table 6.5 Distribution of messages (Msg) received by RSSI and SF.

RSSI [dBm]	SF7		SF9		SF12		Total	
	Msg	%	Msg	%	Msg	%	Msg	%
< -110	4	6.3	7	10.9	8	12.5	19	29.7
$[-110, -90]$	7	10.9	8	12.5	13	20.3	28	43.7
> -90	4	6.3	4	6.3	9	14.1	17	26.6
Total	15	23.4	19	29.7	30	46.9	64	100

It is visible that around the halfway point of the journey, the SNR began to deteriorate significantly (highlighted in red in Fig. 6.1), eventually reaching levels where no messages could be received. These initial measurements confirmed the gateway's limited range and the strong impact of line-of-sight obstructions on signal quality.

6.4.2 Repeater Coverage Extension Evaluation

To address this limitation, the C-Mesh repeater was introduced to determine its role in extending LoRaWAN coverage. The device was placed at the halfway point of the route (near Cabo Ruivo subway station), where messages could still be received by the gateway, even with poor SNR. The field tests were conducted from that location to the end of the route defined in Section 6.4.1, as it was important to assess how many messages could be received with and without the repeater, particularly once the C-Beacon began to experience degraded communication with the gateway.

The C-Mesh (Raspberry Pi with concentrator board) and the C-Point (TTGO LoRa device)

were installed inside a vehicle, as shown in the figure 6.2. The devices were fixed on the car's dashboard, while the LoRa and GPS antennas were placed on the vehicle's roof to ensure better signal reception and transmission conditions. The C-Point was configured to transmit the encapsulated C-Beacon messages using three spreading factors (SF7, SF9, and SF12) in order to evaluate which setting would be most effective for the repeater in future deployments. The C-Beacon is a mobile device that moves along the route, transmitting standard LoRaWAN messages.

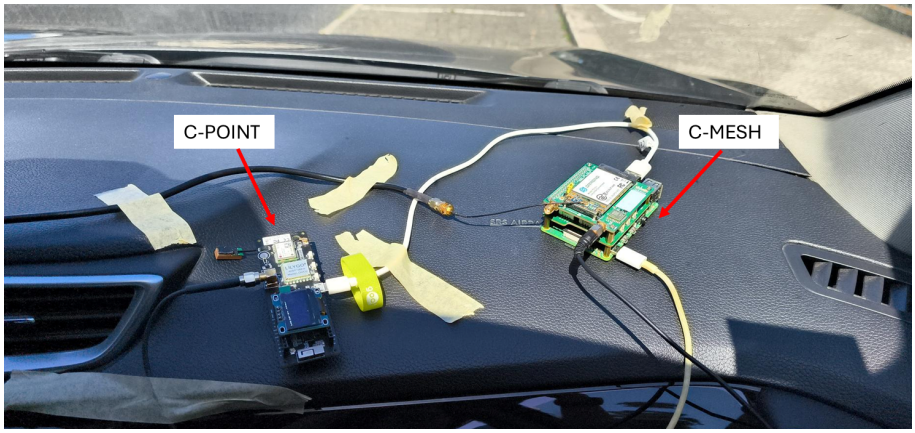


Figure 6.2 Experimental setup of the repeater inside the test vehicle.

The evaluation began from the location of the repeater to verify whether the difficulties observed in Fig. 6.1 persisted. Figure 6.3 compares the distribution of received messages with and without the use of the C-Mesh repeater. In Fig. 6.3a, the C-Beacon's transmissions were captured by the repeater and relayed to the gateway. These encapsulated transmissions resulted in much denser coverage along the route and allowed the finish of the route, demonstrating the repeater's effectiveness in extending LoRaWAN reach. In contrast, Fig. 6.3b shows only direct transmissions from the C-Beacon to the gateway, where very few messages were successfully received and beyond a certain point no messages were received at all, confirming the gateway's limited coverage in the absence of the repeater.

Figure 6.3 Comparison of message reception with (a) and without (b) the repeater.

In total, the C-Beacon transmitted 309 messages. Without the repeater, only 37 messages were received (11.97%) and 272 messages (88.03%) were lost. With the repeater path, 145 frames were delivered (≈46.9%) and 164 were lost (53.1%). The repeater improved the end-to-end delivery by a factor of 3.9 (11.97% → 46.9%). Despite this improvement of 35 percentage points, more than half of the transmissions still failed to be received by the repeater, indicating room for further optimization, for example mounting the antennas higher than the car roof to increase line-of-sight. Overall, the measurements show that the repeater extends by 292% the effective coverage and reliability compared to the direct link.

Table 6.6 Distribution of messages (Msg) by delivery path.

Path	Received		Lost		Total Sent	
	Msg	%	Msg	%	Msg	%
Direct	37	11.97	272	88.03	309	100
Via Repeater	145	46.9	164	53.1	309	100

Analysing signal quality helps show whether the repeater improves the signal or not. Table 6.7 compares the SNR distribution of messages that reached the gateway on the direct path and on the repeater path, for each spreading factor. On the direct path, most receptions occurred at low SNR: 17 of 37 (45.9%) were below −5 dB and only 6 of 37 (16.2%) were above 0 dB. With the repeater, the share above 0 dB increases to 61 of 145 (42.1%), while the share below −5 dB decreases to 55 of 145 (37.9%) and in the intermediate interval [−5, 0] falls from 14 of 37

(37.8%) to 29 of 145 (20.0%). The same pattern happens across spreading factors: the share of receptions with SNR > 0 dB rises from 0.0% (0/9) at SF7, 23.1% (3/13) at SF9 and 20.0% (3/15) at SF12 on the direct path to 44.4% (16/36), 24.4% (11/45) and 53.1% (34/64) with the repeater, respectively. Overall, the repeater not only raises the number of received frames but also shifts reception toward higher SNR at the gateway. Nonetheless, 55 receptions (37.9%) via the repeater remain below −5 dB, indicating that many links still operate close to sensitivity. Once again, it is concluded that improvements in antenna height, placement, or orientation could further increase the SNR profile.

Table 6.7 Distribution of C-Beacon messages by SNR interval and spreading factor (SF), comparing Direct vs. Via repeater paths.

Path	SNR [dB]	SF7		SF9		SF12		Total	
		Msg	%	Msg	%	Msg	%	Msg	%
Direct	< -5	4	10.8	6	16.2	7	18.9	17	45.9
	$[-5, 0]$	5	13.5	4	10.8	5	13.5	14	37.8
	> 0	0	0.0	3	8.1	3	8.1	6	16.2
	Total	9	24.3	13	35.1	15	40.5	37	100
Via repeater	< -5	14	9.7	17	11.7	24	16.6	55	37.9
	$[-5, 0]$	6	4.1	17	11.7	6	4.1	29	20.0
	> 0	16	11.0	11	7.6	34	23.4	61	42.1
	Total	36	24.8	45	31.0	64	44.2	145	100

Table 6.8 reports the RSSI of the messages that reached the gateway, grouped by spreading factor. On the direct path, all receptions fall in [−110,−90] dBm, with none stronger than −90 dBm which indicates a weak link without the repeater. With the repeater, the distribution shifts toward stronger signal levels and spreads across the three intervals. Out of 145 receptions, 76 (52.4%) are < -110 dBm, 37 (25.5%) are in $[-110, -90]$ dBm and 32 (22.1%) are > -90 dBm. Although SF12 still accounts for the largest share of messages, the system also benefits from 36 receptions in SF7 (24.8%) and 45 in SF9 (31%), including 32 packets with RSSI values above −90 dBm (22.1%) compared to 0% in the direct link. This shift indicates that the repeater not only increases the overall number of messages received but also enables the use of lower spreading factors under improved signal conditions.

Table 6.8 Distribution of C-Beacon messages by RSSI interval and spreading factor (SF), comparing Direct vs. Via repeater paths.

Path	RSSI [dBm]	SF7		SF9		SF12		Total	
		Msg	%	Msg	%	Msg	%	Msg	%
Direct	< -110	0	0.0	0	0.0	0	0.0	0	0.0
	$[-110, -90]$	9	24.3	13	35.1	15	40.5	37	100
	> -90	0	0.0	0	0.0	0	0.0	0	0.0
	Total	9	24.3	13	35.1	15	40.5	37	100
Via repeater	< -110	17	11.7	29	20.0	30	20.7	76	52.4
	$[-110, -90]$	10	6.9	13	9.0	14	9.7	37	25.5
	> -90	9	6.2	3	2.1	20	13.8	32	22.1
	Total	36	24.8	45	31.0	64	44.2	145	100

A total of 31 uplinks were observed via both paths (direct and via repeater). The Figure 6.4, shows that for the same packets (same frame counter) the ones forwarded by the C-Point consistently arrive with higher SNR and better RSSI than those received directly. For example, at message 47 the direct path has SNR −4.50 dB and RSSI −106 dBm, whereas the repeater path shows 13.8 dB and −82 dBm and at message 49 from −4.25 dB to 14 dB. This gain is largely explained because the repeater was physically closer to the gateway than the C-Beacon, which further improved path loss. Overall, the figure evidences that the C-Point improves link quality—raising SNR and RSSI and therefore increases the probability of successful delivery.

fcnt	rssi	snr	sf	path	source
47	-106	-4.50	SF7	direct	CBEACON
47	-82	13.8	SF12	via_repeater	CPOINT
48	-108	-2.25	SF12	direct	CBEACON
48	-96	7.20	SF12	via_repeater	CPOINT
49	-109	-4.25	SF9	direct	CBEACON
49	-79	14	SF12	via_repeater	CPOINT

Figure 6.4 Comparison of C-Beacon uplinks received directly and via the repeater.

The C-Point transmitted LoRaWAN messages with 3 SF so we could test which one would be better for the repeater. As shown in Table 6.9, SF12 achieves the highest reliability: 64 of 65 frames were received (98.5%), compared with 45/53 for SF9 (84.9%) and 36/46 for SF7 (78.3%). For SF12, almost every C-Point message reached the gateway (only one did not). This reliability, however, comes with much longer airtime, lower network capacity and higher energy per transmission—factors that matter under EU868 duty-cycle limits and in multi-node deployments where channel occupancy and collision risk grow with the spreading factor. SF7 is the opposite extreme: the shortest airtime and best spectral efficiency, but insufficient margin in these field conditions. Therefore, since the goal is maximum end-to-end reliability at the current coverage edge, SF12 is the safest choice.

Table 6.9 Repeater radio performance per spreading factor (C-Point → Gateway).

SF	**Sent**		**Received**		**Lost**	
	Msg	**%**	**Msg**	**%**	**Msg**	**%**
SF12	65	39.6	64	44.2	1	5.3
SF9	53	32.3	45	31.0	8	42.1
SF7	46	28.1	36	24.8	10	52.6
Total	164	100.0	145	100.0	19	100.0

6.5 Discussion

As defined in Section 1.3, a successful outcome was considered to be a coverage improvement greater than 30% compared to the scenario without the repeater, and at least a 20% increase in packet delivery rate under weak-signal conditions. Both targets were achieved: coverage improved by **35 percentage points**, corresponding to a **3.9× increase** in successful message delivery and an overall extension of communication range and reliability by approximately **292%**. The SNR and RSSI distributions also shifted markedly toward stronger signal regions, **22.1%** of receptions via the repeater achieved RSSI values above −90 dBm, whereas the direct link achieved **0%**, confirming that the proposed system substantially improves link quality and end-to-end reliability. These results demonstrate that the relay not only extends LoRaWAN coverage in harsh NLOS environments but also enhances overall link robustness and delivery probability through improved signal conditions.

Per Table 6.9, SF12 achieved **98.5%** delivery on the repeater hop (C-Point $\rightarrow$ Gateway), clearly outperforming SF9 (**84.9%**) and SF7 (**78.3%**). This aligns with the expected robustness–throughput trade-off: higher spreading factors increase link budget and reliability but reduce channel capacity and raise duty-cycle constraints (as discussed in Chapter 2.2.4). These results confirm that the proposed relay can significantly enhance LoRaWAN performance at the coverage edge, particularly under challenging urban propagation conditions.

In section 3, the study in [50], which implemented the TS011 standard, reported that the relay stopped forwarding messages after approximately one hour of operation, a limitation not observed in this work. In [72], the *e-Node*—a transparent LoRaWAN relay—achieved an RSSI gain of up to 16 dB, particularly at SF7 (the fastest data rate). In this work, as shown in Figure 6.4, the SNR increased from −4.50 dB to 13.8 dB, and the RSSI improved from −106 dBm to −82 dBm. Both studies did not present explicit PDR values, which are provided and analyzed in this work, allowing for a more comprehensive quantitative comparison of end-to-end performance and reliability.

In [34], LoRaWAN connectivity degraded sharply under dense foliage—maintaining communication in only 33% of test locations and showing an RSSI drop of up to 15 dB but the LoRa Meshnet system used by the authors achieved 100% packet delivery over a 1.5 km path through the forest by relying on multi-hop routing. In contrast, the present work focused on a single-hop, LoRaWAN-compliant repeater operating over a longer route in a far more challenging dense urban environment in Lisbon„ characterized by severe NLOS conditions, multipath propagation, and high interference from surrounding buildings and wireless networks. In this work, the share of messages received with positive SNR (>0 dB) increased from **16.2%** to **42.1%**, while the share of receptions with RSSI above −90 dBm rose to **22.1%** compared to **0%** without the relay.

Despite these gains, **53.1%** of frames transmitted via the repeater were still lost. The SNR analysis shows that **37.9%** of receptions remained below −5 dB, but still improved compared to **45.9%** without the repeater. During the tests, it was observed that after a curve along the route (near the Decathlon store in the Oriente area), the repeater path struggled to maintain com-

munication, likely due to stronger NLOS conditions, the high density of surrounding buildings, and the relatively low installation heights of the antennas. It is therefore believed that positioning the antennas at higher elevations—particularly if the repeater were installed on a tall building—would result in significantly better performance. As discussed, the main limitations identified during testing were as follows:

- Antenna height and placement: Both the gateway and repeater antennas were installed at relatively low elevations (balcony and car roof), which limited Fresnel zone clearance and intensified non-line-of-sight (NLOS) losses. Previous studies have consistently shown a strong dependence between antenna height and link quality (3).
- Scenario diversity: It was not possible to evaluate the relay in other environments, such as maritime or rural settings, where line-of-sight (LOS) conditions are typically more favorable and where the potential for further coverage improvement could be assessed.

Chapter 7

Conclusions

7.1 Summary

This thesis addressed the problem of extending the communication range and reliability of LoRaWAN-based IoT networks in environments with limited or no network coverage, such as offshore or remote rural areas. Existing solutions were studied in the state of the art — including LoRaWAN relay standard, other relay works, and mesh-based approaches and satellite networks — have shown potential to increase coverage but remain constrained by several limitations. In particular, many existing relay mechanisms are transparent, require protocol modifications, custom firmware on end devices which is a problem for large-scale deployment and compatibility with the existing LoRaWAN ecosystem.

To overcome these limitations, this work proposed a relay architecture with integrated edge computing that extends LoRaWAN connectivity while retrocompability with standard end devices and gateways. The system was designed to operate autonomously in disconnected environments, using a dual ChirpStack setup — an offline edge instance for local processing and an online instance for centralized management — connected through a BLE-based session key synchronization mechanism. The proposed architecture was implemented using low-cost, open-source hardware from the Custodian Project, integrating the C-Mesh (edge relay) and C-Point (forwarding node) components. The system supports OTAA key provisioning, uplink message forwarding, and local deduplication, all without modifying existing end-device firmware. A lightweight message protocol was developed to enable communication between the C-Mesh and the C-Point while keeping the end devices completely unchanged. It defines two types of messages: CMESH:01, responsible for uplink forwarding with acknowledgements and radio-result feedback (TXOK/TXERR) to guarantee reliable delivery to the C-Point and LoRaWAN message forwarding; and CMESH:02, used for OTAA session updates (DevEUI, DevAddr, AppSKey, NwkSKey), which triggers an ACK or NACK only after the edge network server successfully updates its database. All coordination is contained within the relay components, ensuring that the standard LoRaWAN payload structure remains fully preserved.

Experimental results obtained in an urban scenario (Lisbon) confirmed the feasibility and

effectiveness of the approach. The relay increased the end-to-end message delivery ratio from 11.97% (direct link) to 46.9%, improving coverage by approximately 292% . Signal quality metrics also improved significantly, with higher SNR and RSSI values observed across all spreading factors: receptions with SNR > 0 dB increased from 16.2% to 42.1%, and RSSI > −90 dBm from 0% to 22.1%.

In summary, this thesis demonstrated that combining edge processing and relay functionality within a standards-compliant LoRaWAN architecture can effectively extend network coverage, improve reliability, and maintain compatibility with existing IoT devices already deployed in the field.

7.2 Main Contributions

This thesis presents a novel relay system for LoRaWAN networks with integrated edge processing capabilities, extending the concepts introduced in the Custodian Project [43]. The proposed system addresses key challenges in long-range IoT communication — including coverage extension, key synchronization, and message management — through a modular architecture that combines open-source software and low-cost hardware. The main contributions of this work are as follows:

1. **Design and implementation of a LoRaWAN relay with edge computing capabilities.** The proposed system enables uplink forwarding and local message handling without requiring continuous internet connectivity. It introduces an intermediary node — the relay — composed of C-Mesh and C-Point components, which together allow the reception, validation, and retransmission of LoRaWAN frames in remote environments.

2. **Integration of dual ChirpStack instances for synchronized edge and cloud operation.** The architecture incorporates an offline ChirpStack instance running on the C-Mesh for local message processing and an online instance running on a virtual machine (VM) for centralized management. This dual-instance approach enables key security functions such as MIC validation, decryption, and deduplication to occur both at the edge and in the central platform, improving resilience and scalability.

3. **Development of an OTAA key synchronization mechanism using BLE.** A proof-of-concept BLE communication path was implemented to synchronize session keys between the online and offline ChirpStack servers (section 5.2.1). This mechanism, mediated by the C-Point, allows devices that have joined the network through the central platform to be recognized by the local instance.

4. **Maintaining compatibility with existing LoRaWAN end devices already deployed in the field.** A key design principle of this work was to ensure that the proposed relay operates fully within the standard LoRaWAN protocol stack. The system requires no modification to end-device firmware or message formats, as all message exchanges occur

exclusively between the internal components of the relay. This compatibility allows commercially available and already deployed LoRaWAN nodes to benefit from extended coverage without requiring any changes.

5. **Design of a communication protocol between C-Mesh and C-Point.** A lightweight message protocol was developed for exchanging both application data and control information between the two relay components. The protocol supports message acknowledgment, error reporting, and key provisioning through clearly defined message types and validation rules described in 4.3.

6. **Experimental validation of the system in an urban LoRaWAN environment.** The proposed relay system was experimentally validated in a real urban scenario (Lisbon) as shown in 6.4, characterized by strong non-line-of-sight (NLOS) conditions and multipath propagation. In the initial test without the relay, **88.03%** of the transmitted messages were lost , while with the relay active, the delivery ratio increased to **46.9%**, representing a **3.9× increase** in successful message delivery, effectively extending the communication range and reliability by approximately **292%**. Signal analysis further confirmed the improvement, showing a clear shift toward higher signal quality. With the relay active, the proportion of messages received with SNR values above 0 dB rose from 16.2% to 42.1%, while those with RSSI stronger than −90 dBm increased from 0% to 22.1%, evidencing a substantial improvement in link robustness.

Together, these contributions form a modular and extensible foundation for the future development of relay-enabled LoRaWAN networks with autonomous operation and local intelligence.

7.3 Limitations

Several limitations were encountered during the development and evaluation of this work. It was not possible to test the system with a large number of end-devices (more than 16) due to hardware availability and budget constraints. Field testing was limited to urban environments; maritime trials could not be conducted, as access to a vessel for controlled experimentation was not feasible within the project's resources. The BLE interface and frontend application were outside the scope of this work, which limited the implementation and evaluation of the key synchronization mechanism between the relay and end-user devices. The field experiments were also constrained by antenna placement and elevation. The gateway was installed on a first-floor balcony instead of a rooftop, and the relay node was mounted on a car roof at a relatively low height. These conditions likely reduced the effective communication range and signal quality compared to optimal deployment scenarios. The downlink communication path (from network server to end-devices through the relay) was not extensively tested, as the primary focus of this work was on uplink data forwarding. As a result, further validation of bidirectional

performance remains necessary.

7.4 Future Work

While the results highlight the potential of this relay-based approach, several opportunities remain for further exploration. First, the development of a dedicated BLE mobile application would replace the current proof-of-concept setup using nRF Connect, providing a more intuitive interface for session key provisioning.

Second, designing a custom PCB that integrates both the C-Mesh and C-Point functionalities into a single board would significantly improve practicality and robustness in field deployments, eliminating the need for multiple interconnecting cables and reducing power and space overhead.

Future testing should also be extended to rural and maritime environments, where propagation conditions, interference patterns, and range characteristics differ substantially from those in urban scenarios. These environments typically present more open areas and higher line-of-sight (LOS) availability, potentially resulting in extended coverage compared to urban environments, where non-line-of-sight (NLOS) conditions, building obstructions, and shadowing effects impose stronger limitations.

Another promising direction would be to interesting to expand the system to support multiple relays operating together, exploring communication between them and evaluating the feasibility of forming a mesh network to further enhance coverage and resilience.

Finally, the potential for bidirectional communication between the relays and end devices could be investigated. Although many IoT applications rely primarily on uplink transmissions(and that was the focus of this work), enabling downlink capabilities could broaden the applicability of the system to scenarios requiring remote configuration or actuation.

Bibliography

[1] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, “A comparative study of LPWAN technologies for large-scale IoT deployment,” ICT Express, vol. 5, no. 1, pp. 1–7, Mar. 2019, doi: 10.1016/j.icte.2017.12.005.

[2] R. Marini, K. Mikhaylov, G. Pasolini, and C. Buratti, “Low-Power Wide-Area Networks: Comparison of LoRaWAN and NB-IoT Performance,” IEEE Internet of Things Journal, vol. 9, no. 21, pp. 21051–21063, Nov. 2022, doi: 10.1109/JIOT.2022.3176394.

[3] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, “A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,” Sensors, vol. 16, no. 9, Art. no. 9, Sep. 2016, doi: 10.3390/s16091466.

[4] J. Petäjäjärvi, K. Mikhaylov, M. Pettissalo, J. Janhunen, and J. Iinatti, “Performance of a low-power wide-area network based on LoRa technology: Doppler robustness, scalability, and coverage,” International Journal of Distributed Sensor Networks, vol. 13, no. 3, p. 1550147717699412, Mar. 2017, doi: 10.1177/1550147717699412.

[5] W. Zhou, Z. Tong, Z. Y. Dong, and Y. Wang, “LoRa-Hybrid: A LoRaWAN Based Multi-hop Solution for Regional Microgrid,” in 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Feb. 2019, pp. 650–654. doi: 10.1109/CCOMS.2019.8821683.

[6] T. Buckley, B. Ghosh, and V. Pakrashi, “Edge Structural Health Monitoring (E-SHM) Using Low-Power Wireless Sensing,” Sensors (Basel), vol. 21, no. 20, p. 6760, Oct. 2021, doi: 10.3390/s21206760.

[7] L. Wan, Z. Zhang, and J. Wang, “Demonstrability of Narrowband Internet of Things technology in advanced metering infrastructure,” EURASIP Journal on Wireless Communications and Networking, vol. 2019, no. 1, p. 2, Jan. 2019, doi: 10.1186/s13638-018-1323-y.

[8] C. B. Mwakwata, H. Malik, M. M. Alam, Y. Le Moullec, S. Parand, and S. Mumtaz, ”Narrowband Internet of Things (NB-IoT): From Physical (PHY) and Media Access Control (MAC) Layers Perspectives,” Sensors, vol. 19, no. 11, p. 2613, Jun. 2019, doi: 10.3390/s19112613.

[9] M. Iqbal, A. Y. M. Abdullah, and F. Shabnam, “An Application Based Comparative Study of LPWAN Technologies for IoT Environment,” in 2020 IEEE Region 10 Symposium (TENSYMP), Jun. 2020, pp. 1857–1860. doi: 10.1109/TENSYMP50017.2020.9230597.

[10] M. Kanj, V. Savaux, and M. Le Guen, “A Tutorial on NB-IoT Physical Layer Design,” IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2408–2446, 2020, doi: 10.1109/COMST.2020.3022751.

[11] R. Boisguene, S.-C. Tseng, C.-W. Huang, and P. Lin, “A survey on NB-IoT downlink scheduling: Issues and potential solutions,” in 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Jun. 2017, pp. 547–551. doi: 10.1109/IWCMC.2017.7986344.

[12] “What is Sigfox 0G technology? | Sigfox build.” Accessed: Jan. 09, 2025. [Online]. Available: https://build.sigfox.com

[13] L. Ferreira, ”Sigforgery: Breaking and Fixing Data Authenticity in Sigfox,” in Financial Cryptography and Data Security, 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I, Lecture Notes in Computer Science, vol. 13110, pp. 331-350, Oct. 2021, doi: 10.1007/978-3-662-64322-8_16.

[14] A. Lavric, A. I. Petrariu, and V. Popa, “SigFox Communication Protocol: The New Era of IoT?,” in 2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI), Aug. 2019, pp. 1–4. doi: 10.1109/ISSI47111.2019.9043727.

[15] Yuqi Mo, Minh-Tien Do, and C. Goursaud, “Ultra-narrow-band for IoT”, Big Data-Enabled Internet of Things, pp. 175–198, doi: 10.1049/PBPC025E_ch9.

[16] G. Stanco, A. Navarro, F. Frattini, G. Ventre, and A. Botta, “A comprehensive survey on the security of low power wide area networks for the Internet of Things”, ICT Express, vol. 10, no. 3, pp. 519–552, Jun. 2024, doi: 10.1016/j.icte.2024.03.003.

[17] E. Kadusic, C. Ruland, N. Hadzajlic, and N. Zivic, “The factors for choosing among NB-IoT, LoRaWAN, and Sigfox radio communication technologies for IoT networking”, in 2022 International Conference on Connected Systems & Intelligence (CSI), Aug. 2022, pp. 1–5. doi: 10.1109/CSI54720.2022.9923999.

[18] C. F. Hernandez, O. Iova, and F. Valois, “Downlink Scheduling in LoRaWAN: ChirpStack vs The Things Stack”, in 2024 IEEE Latin-American Conference on Communications (LATINCOM), Nov. 2024, pp. 1–6. doi: 10.1109/LATINCOM62985.2024.10770676.

[19] “TheThingsNetwork/lorawan-stack: The Things Stack, an Open Source LoRaWAN Network Server” Accessed: Jan. 09, 2025. [Online]. Available: https://github.com/TheThingsNetwork/lorawan-stack

[20] “The LoRaWAN Network server for scale”, The Things Industries. Accessed: Jan. 09, 2025. [Online]. Available: https://www.thethingsindustries.com/

[21] “The Things Network”, Accessed: Jan. 09, 2025. [Online]. Available: https://www.thethingsnetwork.org/

[22] “ChirpStack open-source LoRaWAN Network Server”, Accessed: Jan. 09, 2025. [Online]. Available: https://www.chirpstack.io/

[23] “ZeroMQ”, Accessed: Jan. 09, 2025. [Online]. Available: https://zeromq.org/

[24] “LoRa-net GitHub.” Accessed: Jan. 09, 2025. [Online]. Available: https://github.com/Lora-net

[25] “ChirpStack GitHub.” Accessed: Jan. 09, 2025. [Online]. Available: https://github.com/chirpstack

[26] “LoRa Basics GitHub.” Accessed: Jan. 09, 2025. [Online]. Available: https://github.com/lorabasics

[27] “Helium - Own the Air”, Accessed: Jan. 09, 2025. [Online]. Available: https://www.helium.com/

[28] “TS011-1.0.0 Relay”, Accessed: Jan. 09, 2025. [Online]. Available: https://resources.lora-alliance.org/technical-specifications/ts011-1-0-0-relay

[29] V. D. Pham, V. Kisel, R. Kirichek, A. Koucheryavy, and A. Shestakov, “Evaluation of A Mesh Network based on LoRa Technology,” in 2021 23rd International Conference on Advanced Communication Technology (ICACT), Feb. 2021, pp. 1–6. doi: 10.23919/ICACT51234.2021.9370792.

[30] J.-M. Mari and A. Gabillon, “The MauMe network - A LoRa multi-hop collaborative protocol and low-cost implementation example”, Computer Standards & Interfaces, vol. 86, p. 103733, Aug. 2023, doi: 10.1016/j.csi.2023.103733.

[31] “Meshtastic.” Accessed: Jan. 09, 2025. [Online]. Available: https://meshtastic.org/

[32] “Pymesh.” Accessed: Jan. 09, 2025. [Online]. Available: https://docs.pycom.io/pymesh/

[33] M. Hammouti, O. Moussaoui, M. Haissaine, and A. Betari, “Comparative Evaluation of IoT Mesh Network Protocols: NeoMesh Versus LoRa Pymesh”, in 2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), Nov. 2023, pp. 01–06. doi: 10.1109/CloudTech58737.2023.10366151.

[34] H. Kim, H. Kim, S. Baek, R. Melenchuk, J. Soroka, and A. Smith, “Hybrid LoRa Network Architecture: Automatic Switching between LoRaWAN and LoRa Mesh Network in Environments with Dynamic Obstacle Variations”, in 2024 33rd International Conference on

Computer Communications and Networks (ICCCN), Jul. 2024, pp. 1–6. doi: 10.1109/ICCCN61486.2024.10637558.

[35] N. C. Almeida, R. P. Rolle, E. P. Godoy, P. Ferrari, and E. Sisinni, “Proposal of a Hybrid LoRa Mesh / LoRaWAN Network”, in 2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT, Jun. 2020, pp. 702–707. doi: 10.1109/MetroInd4.0IoT48571.2020.9138206.

[36] Z. Qu, H. Cao, Y. Cheng, S. Wu, and G. Zhang, “A LoRaWAN-Based Network Architecture for LEO Satellite Internet of Things,” in 2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), May 2019, pp. 1–2. doi: 10.1109/ICCE-TW46550.2019.8991826.

[37] L. CHAARI, M. FOURATI, and J. REZGUI, “Heterogeneous LoRaWAN & LEO Satellites Networks Concepts, Architectures and Future directions,” in 2019 Global Information Infrastructure and Networking Symposium (GIIS), Dec. 2019, pp. 1–6. doi: 10.1109/GIIS48668.2019.9044966.

[38] A. Castro-Carrera, “A systematic review of LEO satellite services for IoT”, Multidisciplinary Reviews, vol. 8, no. 3, pp. 2025083–2025083, 2025, doi: 10.31893/multirev.2025083.

[39] O. Ledesma, P. Lamo, and J. A. Fraire, ”Trends in LPWAN Technologies for LEO Satellite Constellations in the NewSpace Context,” Electronics, vol. 13, no. 3, p. 579, Jan. 2024, doi: 10.3390/electronics13030579.

[40] “Lacuna Space – Low-cost, simple and reliable global connections to sensors and mobile equipment.” Accessed: Jan. 09, 2025. [Online]. Available: https://lacuna-space.com/

[41] A. A. Doroshkin, A. M. Zadorozhny, O. N. Kus, V. Yu. Prokopyev, and Y. Prokopyev, ”Experimental Study of LoRa Modulation Immunity to Doppler Effect in CubeSat Radio Communications,” IEEE Access, vol. PP, no. 99, pp. 1-1, May 2019, doi: 10.1109/ACCESS.2019.2919274.

[42] M. Centenaro, C. E. Costa, F. Granelli, C. Sacchi, and L. Vangelista, “A Survey on Technologies, Standards and Open Challenges in Satellite IoT” IEEE Communications Surveys & Tutorials, vol. 23, no. 3, pp. 1693–1720, 2021, doi: 10.1109/COMST.2021.3078433.

[43] “Custodian – Custodian Website.” Accessed: Jan. 09, 2025. [Online]. Available: https://-custodian.solvit.pt/

[44] “SX1302/SX1303 Hardware Abstraction Layer and Semtech UDP Packet Forwarder.” Accessed: Sep. 1, 2025. [Online]. Available: https://github.com/Lora-net/sx1302_hal

[45] ”LoRa Basics Station” Accessed: Sep. 1, 2025. [Online]. Available: https://github.com/lorabasics/basicstation

[46] Zakutynskyi, Ihor and Rabodzei, Ihor. ”Microservice Communication for IoT-based Systems. Architecture Review and Performance Test”, Electronics and Control Systems. 4. 73-78, 2022, doi: 10.18372/1990-5548.74.17311.

[47] B. Mishra and A. Kertesz, “The Use of MQTT in M2M and IoT Systems: A Survey,” IEEE Access, vol. 8, pp. 201071–201086, 2020, doi: 10.1109/ACCESS.2020.3035849.

[48] “Eclipse Mosquitto” Accessed: Sep. 1, 2025. [Online]. Available: https://mosquitto.org/

[49] “HiveMQ Platform:: HiveMQ Documentation” Accessed: Sep. 1, 2025. [Online]. Available: https://docs.hivemq.com/hivemq/latest/user-guide/index.html

[50] E. T. Hope, “Using relays for utilizing and extending LoRaWAN networks with the use of Altibox LoRaWAN infrastructure for internet of things,” Master thesis, University of South-Eastern Norway, 2024. Accessed: Mar. 01, 2025. [Online]. Available: https://openarchive.usn.no/usn-xmlui/handle/11250/3136690

[51] K. Kelleher, ”The costs of monitoring, control and surveillance of fisheries in developing countries.”, FAO Fisheries, 2002.

[52] F. S. Alqurashi, A. Trichili, N. Saeed, B. S. Ooi and M. -S. Alouini, ”Maritime Communications: A Survey on Enabling Technologies, Opportunities, and Challenges,” in IEEE Internet of Things Journal, vol. 10, no. 4, pp. 3525-3547, 15 Feb.15, 2023, doi: 10.1109/JIOT.2022.3219674.

[53] N. Cruz, N. Cota, and J. Tremoceiro, “LoRaWAN and Urban Waste Management—A Trial,” Sensors, vol. 21, no. 6, Art. no. 6, Jan. 2021, doi: 10.3390/s21062142.

[54] “Raspberry Pi - ChirpStack Gateway Bridge documentation.” Accessed: Sep. 1, 2025. [Online]. Available: https://www.chirpstack.io/docs/chirpstack-gateway-bridge/install/raspberry-pi.html

[55] “Debian / Ubuntu - ChirpStack open-source LoRaWAN® Network Server documentation.” Accessed: Sep. 1, 2025. [Online]. Available: https://www.chirpstack.io/docs/getting-started/debian-ubuntu.html

[56] “MQTT Explorer” Accessed: Sep. 1, 2025. [Online]. Available: http://mqtt-explorer.com/

[57] “CrateDB” Accessed: Sep. 1, 2025. [Online]. Available: https://cratedb.com

[58] “Grafana” Accessed: Sep. 1, 2025. [Online]. Available: https://grafana.com/

[59] ”Paho.mqtt python library” Accessed: Sep. 1, 2025. [Online]. Available: https://github.com/eclipse-paho/paho.mqtt.python

[60] ”Crate python library” Accessed: Sep. 2, 2025. [Online]. Available: https://github.com/crate/crate-python

[61] “Python language” Accessed: Sep. 2, 2025. [Online]. Available: https://www.python.org/

[62] “Chirpstack Protobuf Schemas” Accessed: Sep. 2, 2025. [Online]. Available: https://github.com/chirpstack/chirpstack/tree/master/api/proto

[63] “Protocol Buffers” Accessed: Sep. 2, 2025. [Online]. Available: https://github.com/protocolbuffers/protobuf

[64] “nRF Connect App for Mobile” Accessed: Sep. 3, 2025. [Online]. Available: https://-play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en

[65] Kelleher, K. ” The costs of monitoring, control and surveillance of fisheries in developing countries.” , FAO Fisheries, 2002.

[66] Pinelo, J.; Rocha, A.D.; Arvana, M.; Gonçalves, J.; Cota, N.; Silva, P. Unveiling LoRa’s Oceanic Reach: Assessing the Coverage of the Azores LoRaWAN Network from an Island. Sensors 2023, 23, 7394. https://doi.org/10.3390/s23177394

[67] R. Mukhia, K. G. Sarambage Jayarathna, and A. Lertsinsrubtavee, “Performance Evaluation of LoRaWAN Forest Fire Monitoring Network in the Wild” in Proceedings of the 18th Asian Internet Engineering Conference, in AINTEC ’23. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 96–104. doi: 10.1145/3630590.3630602.

[68] A. Etaati, M. Bastam, and E. Ataie, “Smart forest monitoring: A novel Internet of Things framework with shortest path routing for sustainable environmental management” IET Networks, vol. 13, no. 5–6, pp. 528–545, 2024, doi: 10.1049/ntw2.12135.

[69] Union, E. The EU fisheries control system gets a major revamp - European Commission, 2024.

[70] J. Brown and G. Macfadyen, “Ghost fishing in European waters: Impacts and management responses” Marine Policy, vol. 31, no. 4, pp. 488–504, Jul. 2007, doi: 10.1016/j.marpol.2006.10.007.

[71] E. Lumet, A. Le Floch, R. Kacimi, M. Lihoreau, and A.-L. Beylot, “LoRaWAN Relaying: Push the Cell Boundaries” in Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, in MSWiM ’21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 217–220. doi: 10.1145/3479239.3485718.

[72] E. Sisinni, P. Ferrari, D. F. Carvalho, S. Rinaldi, M. Pasetti, A. Flammini, and A. Depari, “LoRaWAN Range Extender for Industrial IoT” IEEE Transactions on Industrial Informatics, vol. 16, no. 8, pp. 5607–5616, Aug. 2020, doi: 10.1109/TII.2019.2958620.

[73] M. DIOP and C. PHAM, “Increased flexibility in long-range IoT deployments with transparent and light-weight 2-hop LoRa approach” in 2019 Wireless Days (WD), Apr. 2019, pp. 1–6. doi: 10.1109/WD.2019.8734228.

[74] “Introduction - ChirpStack Gateway Mesh documentation.” Accessed: Jan. 15, 2025. [Online]. Available: https://www.chirpstack.io/docs/chirpstack-gateway-mesh/index.html

[75] R. Sanchez-Vital, L. Casals, B. Heer-Salva, R. Vidal, C. Gomez, and E. Garcia-Villegas, “Energy Performance of LR-FHSS: Analysis and Evaluation,” Sensors, vol. 24, no. 17, p. 5770, Jan. 2024, doi: 10.3390/s24175770.

[76] M. A. Ullah, K. Mikhaylov and H. Alves, ”Analysis and Simulation of LoRaWAN LR-FHSS for Direct-to-Satellite Scenario,” in IEEE Wireless Communications Letters, vol. 11, no. 3, pp. 548-552, March 2022, doi: 10.1109/LWC.2021.3135984.

[77] S. Jung, S. Jeong, J. Kang, J. G. Ryu and J. Kang, ”Transceiver Design and Performance Analysis for LR-FHSS-Based Direct-to-Satellite IoT,” in IEEE Communications Letters, vol. 27, no. 12, pp. 3310-3314, Dec. 2023, doi: 10.1109/LCOMM.2023.3325052.

[78] “LtAP LR8G LTE6 kit | MikroTik.” Accessed: Oct. 1, 2025. [Online]. Available: https://mikrotik.com/product/ltap_lr8g_lte6_kit

[79] “Raspberry Pi 4 Model B.” Accessed: Oct. 1, 2025. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[80] “PG1302 LoRaWAN Concentrator.” Accessed: Oct. 1, 2025. [Online]. Available: https://www.dragino.com/products/lora/item/223-pg1302.html

[81] “T-Beam Board LILYGO ” Accessed: Oct. 1, 2025. [Online]. Available: https://lilygo.cc/products/t-beam

Anexo A

Gateway Bridge Configuration

This configuration provides a Semtech UDP packet-forwarder backend and integrates with a MQTT broker.

```
1
2  # See https://www.chirpstack.io/gateway-bridge/install/config/ for a full
3  # configuration example and documentation.
4
5  # Gateway backend configuration.
6  [backend]
7  # Backend type.
8  type="semtech_udp"
9
10   # Semtech UDP packet-forwarder backend.
11   [backend.semtech_udp]
12
13   # ip:port to bind the UDP listener to
14   #
15   # Example: 0.0.0.0:1700 to listen on port 1700 for all network interfaces.
16   # This is the listener to which the packet-forwarder forwards its data
17   # so make sure the 'serv_port_up' and 'serv_port_down' from your
18   # packet-forwarder matches this port.
19   udp_bind ="0.0.0.0:1700"
20
21
22 # Integration configuration.
23 [integration]
24 # Payload marshaler.
25 #
26 # This defines how the MQTT payloads are encoded. Valid options are:
27 # * protobuf: Protobuf encoding
28 # * json: JSON encoding (easier for debugging, but less compact than 'protobuf')
29 marshaler="protobuf"
30
```

```
# MQTT integration configuration.
[integration.mqtt]
# Event topic template.
event_topic_template="eu868/gateway/{{ .GatewayID }}/event/{{ .EventType }}"
# Command topic template.
command_topic_template="eu868/gateway/{{ .GatewayID }}/command/#"

# MQTT authentication.
[integration.mqtt.auth]
# Type defines the MQTT authentication type to use.
#
# Set this to the name of one of the sections below.
type="generic"

  # Generic MQTT authentication.
  [integration.mqtt.auth.generic]
  # MQTT server (e.g. scheme://host:port where scheme is tcp, ssl or ws)
  server="tcp://127.0.0.1:1883"

  # Connect with the given username (optional)
  username=""

  # Connect with the given password (optional)
  password=""
```

Anexo B

Semtech UDP Packet Forwarder

```
{
    "SX130x_conf": {
        "com_type": "SPI",
        "com_path": "/dev/spidev0.0",
        "lorawan_public": true,
        "clksrc": 0,
        "antenna_gain": 0,/* antenna gain, in dBi */
        "full_duplex": false,
        "fine_timestamp": {
            "enable": false,
            "mode": "all_sf" /* high_capacity or all_sf */
        },
        "sx1261_conf": {
            "spi_path": "/dev/spidev0.1",
            "rssi_offset": 0,/* dB */
            "spectral_scan": {
                "enable": false,
                "freq_start": 867100000,
                "nb_chan": 8,
                "nb_scan": 2000,
                "pace_s": 10
            },
            "lbt": {
                "enable": false,
                "rssi_target": -70, /* dBm */
                "channels":[ /* 16 channels maximum */
                    {"freq_hz": 867100000,"bandwidth": 125000,"scan_time_us": 128,"
                        transmit_time_ms": 400},
                    {"freq_hz": 867300000,"bandwidth": 125000,"scan_time_us": 128,"
                        transmit_time_ms": 400},
                    {"freq_hz": 867500000,"bandwidth": 125000,"scan_time_us": 128,"
                        transmit_time_ms": 400},
```

```
                {"freq_hz": 867700000,"bandwidth": 125000,"scan_time_us": 128,"
                    transmit_time_ms": 400},
                {"freq_hz": 867900000,"bandwidth": 125000,"scan_time_us": 128,"
                    transmit_time_ms": 400},
                {"freq_hz": 868100000,"bandwidth": 125000,"scan_time_us": 128,"
                    transmit_time_ms": 400},
                {"freq_hz": 868300000,"bandwidth": 125000,"scan_time_us": 128,"
                    transmit_time_ms": 400},
                {"freq_hz": 868500000,"bandwidth": 125000,"scan_time_us": 128,"
                    transmit_time_ms": 400},
                {"freq_hz": 869525000,"bandwidth": 125000,"scan_time_us": 5000,"
                    transmit_time_ms": 4000},
                {"freq_hz": 868300000,"bandwidth": 250000,"scan_time_us": 128,"
                    transmit_time_ms": 400}
            ]
        }
    },
     "radio_0": {
        "enable": true,
        "type": "SX1250",
        "freq": 867500000,
        "rssi_offset": -215.4,
        "rssi_tcomp": {"coeff_a": 0,"coeff_b": 0,"coeff_c": 20.41, "coeff_d": 2162.56, "
            coeff_e": 0},
        "tx_enable": true,
        "tx_freq_min": 863000000,
        "tx_freq_max": 870000000,
        "tx_gain_lut":[
            {"rf_power": 12,"pa_gain": 0,"pwr_idx": 15},
            {"rf_power": 13,"pa_gain": 0,"pwr_idx": 16},
            {"rf_power": 14,"pa_gain": 0,"pwr_idx": 17},
            {"rf_power": 15,"pa_gain": 0,"pwr_idx": 19},
            {"rf_power": 16,"pa_gain": 0,"pwr_idx": 20},
            {"rf_power": 17,"pa_gain": 0,"pwr_idx": 22},
            {"rf_power": 18,"pa_gain": 1,"pwr_idx": 1},
            {"rf_power": 19,"pa_gain": 1,"pwr_idx": 2},
            {"rf_power": 20,"pa_gain": 1,"pwr_idx": 3},
            {"rf_power": 21,"pa_gain": 1,"pwr_idx": 4},
            {"rf_power": 22,"pa_gain": 1,"pwr_idx": 5},
            {"rf_power": 23,"pa_gain": 1,"pwr_idx": 6},
            {"rf_power": 24,"pa_gain": 1,"pwr_idx": 7},
            {"rf_power": 25,"pa_gain": 1,"pwr_idx": 9},
            {"rf_power": 26,"pa_gain": 1,"pwr_idx": 11},
            {"rf_power": 27,"pa_gain": 1,"pwr_idx": 14}
        ]
```

```
        },
        "radio_1": {
            "enable": true,
            "type": "SX1250",
            "freq": 868500000,
            "rssi_offset": -215.4,
            "rssi_tcomp": {"coeff_a": 0,"coeff_b": 0,"coeff_c": 20.41, "coeff_d": 2162.56, "
                coeff_e": 0},
            "tx_enable": false
        },
        "chan_multiSF_All": {"spreading_factor_enable": [5,6,7,8,9,10,11,12]},
        "chan_multiSF_0": {"enable": true, "radio": 1,"if": -400000},
        "chan_multiSF_1": {"enable": true, "radio": 1,"if": -200000},
        "chan_multiSF_2": {"enable": true, "radio": 1,"if": 0},
        "chan_multiSF_3": {"enable": true, "radio": 0,"if": -400000},
        "chan_multiSF_4": {"enable": true, "radio": 0,"if": -200000},
        "chan_multiSF_5": {"enable": true, "radio": 0,"if": 0},
        "chan_multiSF_6": {"enable": true, "radio": 0,"if": 200000},
        "chan_multiSF_7": {"enable": true, "radio": 0,"if": 400000},
        "chan_Lora_std": {"enable": true, "radio": 1,"if": -200000, "bandwidth": 250000,"
            spread_factor": 7,
                        "implicit_hdr": false, "implicit_payload_length": 17,"
                            implicit_crc_en": false, "implicit_coderate": 1},
        "chan_FSK": {"enable": true, "radio": 1,"if": 300000,"bandwidth": 125000,"datarate"
            : 50000}
    },

    "gateway_conf": {
        "gateway_ID": "0016c001f1101ee1",
        /* change with default server address/ports */
        "server_address": "localhost",
        "serv_port_up": 1700,
        "serv_port_down": 1700,
        /* adjust the following parameters for your network */
        "keepalive_interval": 10,
        "stat_interval": 30,
        "push_timeout_ms": 100,
        /* forward only valid packets */
        "forward_crc_valid": true,
        "forward_crc_error": false,
        "forward_crc_disabled": false,
        /* GPS configuration */
        "gps_tty_path": "",
        /* GPS reference coordinates */
        "ref_latitude": 0.0,
```

```
        "ref_longitude": 0.0,
        "ref_altitude": 0,
        /* Beaconing parameters */
        "beacon_period": 0,
        "beacon_freq_hz": 869525000,
        "beacon_datarate": 9,
        "beacon_bw_hz": 125000,
        "beacon_power": 14,
        "beacon_infodesc": 0
    },

    "debug_conf": {
        "ref_payload":[
            {"id": "0xCAFE1234"},
            {"id": "0xCAFE2345"}
        ],
        "log_file": "loragw_hal.log"
    }
}
```

Anexo C

C-Beacon decoder on Chirpstack NS

```
1
2  function decodeUplink(input) {
3    const bytes = input.bytes;
4
5    if (!bytes || bytes.length < 10) {
6      return {
7        errors: ["Payload too short (need 10 bytes)"]
8      };
9    }
10
11   const lat_raw = (bytes[0] << 16) | (bytes[1] << 8) | bytes[2];
12   const lon_raw = (bytes[3] << 16) | (bytes[4] << 8) | bytes[5];
13   const alt_raw = (bytes[6] << 8) | bytes[7];
14   const hdop_raw = bytes[8];
15   const sats = bytes[9];
16
17
18   const latitude = (lat_raw / 0xFFFFFF) * 180 - 90;
19   const longitude = (lon_raw / 0xFFFFFF) * 360 - 180;
20   const altitude = alt_raw;
21   const hdop = hdop_raw / 10.0;
22
23   return {
24     data: {
25       latitude: Number(latitude.toFixed(6)),
26       longitude: Number(longitude.toFixed(6)),
27       altitude: altitude,
28       hdop: hdop,
29       sats: sats
30     }
31   };
32 }
```

Anexo D

C-Point decoder on ChirpstacK NS

```
1
2  function decodeUplink(input) {
3    const bytes = input.bytes;
4    if (!bytes || bytes.length < 16) { // 16 header + 10 payload
5      return { errors: ["Payload too short (need >= 16 bytes)"] };
6    }
7
8    // ---- C-Point fields ----
9    const timestamp = (bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3];
10   const devAddr = (bytes[4] << 24) | (bytes[5] << 16) | (bytes[6] << 8) | bytes[7];
11   const fCnt = (bytes[8] << 8) | bytes[9];
12   const fPort = bytes[10];
13   const rssi = ((bytes[11] << 8) | (bytes[12] & 0xFF)) << 16 >> 16;
14   const snr10 = ((bytes[13] << 8) | (bytes[14] & 0xFF)) << 16 >> 16;
15   const snr = snr10 / 10.0;
16   const sf = bytes [15]
17
18
19
20   // ---- C-Beacon encapsulated payload (10 bytes) ----
21   const p = bytes.slice(16);
22   const lat_raw = (p[0] << 16) | (p[1] << 8) | p[2];
23   const lon_raw = (p[3] << 16) | (p[4] << 8) | p[5];
24   const alt_raw = (p[6] << 8) | p[7];
25   const hdop_raw= p[8];
26   const sats = p[9];
27
28   const latitude = (lat_raw / 0xFFFFFF) * 180 - 90;
29   const longitude = (lon_raw / 0xFFFFFF) * 360 - 180;
30   const altitude = alt_raw;
31   const hdop = hdop_raw / 10.0;
32
```

```
33
34
35    return {
36      data: {
37        // C-Point fields
38        timestamp: timestamp,
39        devAddr: "0x" + devAddr.toString(16).padStart(8, "0"),
40        fCnt: fCnt,
41        fPort: fPort,
42        rssi: rssi,
43        snr: snr,
44             sf: sf,
45        // C-Beacon decoded coordinates
46        latitude: Number(latitude.toFixed(6)),
47        longitude: Number(longitude.toFixed(6)),
48        altitude: altitude,
49        hdop: hdop,
50        sats: sats,
51      }
52    };
53  }
```