Sapienza
Università di Roma

Edge2LoRa: A New Paradigm for Enabling Cloud Edge Computing Continuum over LoRaWAN

Department of Computer, Control and Management Engineering – DIAG
Ph.D. in Engineering in Computer Science (XXXVII cycle)

Stefano Milani
ID number 1707181

Advisor
Prof. Ioannis Chatzigiannakis

Co-Advisor
Prof. Domenico Garlisi

Academic Year 2024

Edge2LoRa: A New Paradigm for Enabling Cloud Edge Computing Continuum over LoRaWAN
PhD Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: milani@diag.uniroma1.it

"E sono contento della scelta che ho fatto
Nemmeno un rimorso, nemmeno un rimpianto
Sì, sono contento, che bella scoperta
Non serve nient'altro che fare una scelta"
Michele Salvemini, in arte Caparezza

Abstract

Low-Power Wide-Area Network (LPWAN) offer a low-cost solution for deploying large-scale Internet of Things (IoT) infrastructures by adhering to a classic producer/-consumer model with minimal requirements. However, as these deployments scale, the need for low-latency, distributed, and collaborative data aggregation models becomes inevitable. The Cloud Edge Computing Continuum (CECC) has been proposed as an evolution of traditional centralized, ultra-high-end processing clouds into a continuum of collaborative processing elements, distributed from the cloud to the network edge.

Integrating existing centralized and monolithic LPWAN architectures into the CECC framework, however, poses several security challenges. To address these challenges, I propose Edge2LoRa (E2L), a comprehensive and secure solution that integrates LPWAN architectures into the CECC. This integration facilitates faster data processing while minimizing the transmission of sensitive data. E2L enhances network performance by enabling data pre-processing, optimizing traffic flow, and providing real-time local analysis.

By progressively transforming existing LPWAN deployments into agile and versatile infrastructures, E2L ensures seamless and efficient data processing across the CECC while maintaining service continuity and full backward compatibility.

The E2L implementation is available as open-source and is compatible with hardware that adheres to the Things Stack and Long-Range Wide-Area Network (LoRaWAN) v1.0.4 and v1.1 specifications. The system's performance was evaluated in terms of network and computing resource utilization, Quality of Service (QoS), and security. Results demonstrate significant improvements in both public and private LoRaWAN infrastructures, without any disruption or degradation of existing legacy services. In particular, for public LoRaWAN deployments supporting large-scale IoT data streams and big data analytics, a reduction in core network bandwidth usage by up to 90% is observed, along with data processing latency improvements of up to 10×.

Additionally, as a further contribution, Edge4LoRa (E4L) incorporates a distinct computing module capable of processing data streams at the network edge. This module utilizes a Map/Reduce engine based on Apache Spark, enabling the execution of various processing applications, including anomaly detection and data reduction techniques. Furthermore, E4L enables efficient traffic routing across LoRaWAN gateways, addressing the dynamic nature of IoT data traffic and the mobility of source devices.

The proposed architecture ensures modularity, reliability, scalability, and robustness. The evaluation demonstrates its effectiveness under different testbed configurations. Performance assessments were conducted using a hardware setup in a laboratory, and we evaluated the architecture across three scenarios: data reduction, scaling activation of edge gateways, and mobility-aware scenarios.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) has been a game changer in the way we operate, live, commute, and conduct business thanks to its capacity to monitor in real-time a broad range of environmental parameters in indoor, outdoor, industrial, urban, as well as rural areas [37]. From homes to cities, from farms to factories, in hospitals and vehicles, IoT deployments expand our ability to monitor physical phenomena and offer us the opportunity to significantly enhance the way we control our environment [69]. My research focuses on the expected significant growth in the IoT sector, with IoT device connections predicted to rise from 7 billion in 2018 to 22 billion by 2025.

The proliferation of the IoT applications has significantly increased data generation across various fields such as informatics, urban planning, and economics. The International Data Corporation (IDC) predicts that the global data volume will reach a staggering 175 zettabytes by 2025. This explosion of data has created a growing need for the analysis of vast and diverse datasets produced by a multitude of IoT devices, often referred to as IoT big data. Within this diverse landscape of massive IoT use cases, emerging Low-Power Wide-Area Network (LPWAN) technologies introduce a uniform approach for creating global networks of devices for power-efficient and cost-effective integration of IoT data streams with cutting-edge web services across numerous specialized sectors [16, 33, 51].

Given the realization of the above new/novel application scenarios, the International Data Corporation (IDC) [52] has projected that the global volume of data will reach an astounding 175 zettabytes by 2025. For this reason the analysis of large and diverse datasets generated, which is commonly referred to as IoT big data [59, 60], needs to be done across the entire Cloud Edge Computing Continuum (CECC). Distributed computing plays a crucial role in the reduction of extensive network traffic, enabling processing and real-time response to scale network dimensions.

To support communication between this large number of IoT devices, there are technologies such as Long-Range Wide-Area Network (LoRaWAN) [5], which enables devices to establish connections over large distances with minimal consumption of energy and computational resources. Such scenarios are discussed within the LoRaWAN Alliance through numerous white papers to support various use cases[1]. Although there's a growing necessity for distributed processing, LoRaWAN technol-

[1]https://resources.lora-alliance.org/whitepapers

ogy operates on a centralized architecture, channeling all data produced by devices to a singular network/application server and it uses end-to-end encryption to ensure data confidentiality and security. In light of the existing limitations of current IoT edge processing solutions, this thesis proposes the integration of a processing module into a LoRaWAN architecture, capitalizing on the principles of edge computing.

In this work, I focus on LoRaWAN as one of the most adopted LPWAN technologies [23,30]. LoRaWAN introduces a centralised, monolithic, architecture where network gateways (GWs) bridge traffic arriving from the IoT devices with the network backbone by directly forwarding all the IoT traffic to central cloud services [4]. Following a licence-free ad-hoc deployment model, it provides an ideal solution for connecting a wide range of IoT devices with minimal infrastructure requirements [49]. At the same time, the standard producer/consumer model adopted by LoRaWAN allows the straight-forward deployment of cloud-based analytics services to process the large amounts of IoT generated data streams [62]. For a graphical representation of the key components of LoRaWAN, see Fig. 4.1.

LoRaWAN technology is recognized for its suitability in supporting very large-scale deployments and managing substantial data volumes. The ETSI TR 103 526 (ETSI-EN-300) [18] document underscores the scalability and wideband capabilities of LPWAN LoRaWAN technology, particularly its unique "network densification" feature. The capture effect, an essential part of Long-Range (LoRa) modulation, aids in multi-gateway setups by allowing at least one gateway to decode a packet successfully, even amidst collisions. Increasing gateway density can enhance data collection, improve the capture effect, and balance overall airtime. Another approach is the potential use of Long-Range 2.4GHz (LoRa 2.4GHz) while reduces the range compared to traditional LoRa, it significantly enhances network capacity by enabling higher data rates and eliminating the need to comply with any duty cycle restrictions [32]. Specifically, the data rates for LoRa range from $0.3kbit/s$ to $50kbit/s$ depending on the spreading factor (SF) used [1], whereas LoRa 2.4GHz supports data rates ranging from $0.595kbit/s$ to $253.91kbit/s$. This higher data rate allows LoRa 2.4GHz to facilitate the use of LoRaWAN for applications requiring lower latency, higher data rates, and non-scheduled packet transmissions. Incorporating CECC in such applications is crucial for meeting the demands of real-time operations over LoRaWAN.

Although LoRaWAN has served as an excellent starting point to integrate IoT infrastructures with cloud services and big data analytics, the centralized architecture faces certain bottlenecks when scaling up to large deployment areas. Increasing the number of gateways that forward large volumes of unprocessed IoT data to the centralised infrastructure places significant pressure on the network backbone in terms of bandwidth, energy and security [50]. At the same time, the emergence of new services that rely on new deployment and operational models require a shift from the classical two-tier producer/consumer model to multi-tier models involving client or application consumers, applications, and data sources. Such kinds of models necessitate a paradigm shift in the processing of IoT data for low-latency, distributed and collaborative aggregation. We are now entering a new phase where the fast and low-cost deployment strategy needs to be transformed into a scalable and agile approach that comprehensively considers these service requirements and optimally addresses them to enhance efficiency, sustainability, safety, and resilience.

The main contributions of this thesis can be summarized as follows:

1. Edge2LoRa (E2L) is introduced, a new approach to transform the LoRaWAN GWs into an active element in the CECC by providing computational and storage resources at the edges of the network. The E2L elements are designed in a way such that they respect the existing LoRaWAN specification and guarantee backward compatibility, allowing seamless interoperability between legacy and CECC-native elements, as demonstrated in the detailed performance evaluation included in the thesis.

2. A mechanism that enables the association of IoT LoRaWAN End Devices (EDs) that are CECC-native to a specific CECC-native LoRaWAN Gateways (GWs) that will serve as the intermediate point for processing and storage in the CECC. The mechanism allows different strategies to be used for the association of GWs to EDs based on diverse optimization criteria of the network operation, the hardware capabilities and/or application-level metrics. The design allows the *dynamic association* of EDs to GWs so that current network conditions and application-level parameters can be taken into consideration to optimize the available network, processing and storage resources.

3. An extended rejoin service procedure that allows the CECC-native ED, the associated GW and the LoRaWAN Application Server (AS) to establish a group key. The group key is used by the CECC-native EDs to encrypt the IoT data before transmitting them to the LoRaWAN. Since the associated GW participates in the key generation phase, this mechanism enables the CECC-native GW to access the application payload of the LoRaWAN frames in a secure and privacy-preserving manner. This extended rejoin service is fundamental to allow the GWs to act as storage and processing elements within the CECC. Simultaneously, given that the group key is established after the deployment of the EDs in the area of interest, avoiding to hard-code the keys into the ED firmware and the GW software, ensuring they are not fixed for the entire duration of the network deployment.

4. A mechanism that provides certain *Quality of Service (QoS) guarantees* for the processing and storage of messages within the CECC even in the case of multiple deliveries of frames via legacy equipment. Detailed information are provided on the operation of the network components and how they can coexist with public infrastructures that rely on legacy LoRaWAN Network Server (NS) and LoRaWAN Join Server (JS) compliant with the LoRaWAN v1.0.4 or v.1.1 specifications.

5. The integration of a processing module into a LoRaWAN network using the principles of edge computing. Edge4LoRa (E4L), incorporates a distinct computing module capable of processing data streams at the network edge. The module utilizes a Map/Reduce engine based on Apache Spark, enabling the execution of various processing applications, including anomaly detection and data reduction techniques. Additionally, E4L enables traffic to move across LoRaWAN GWs, facing the nature of the IoT data traffic mining and mobility of the source EDs, and allowing to rely on multiple QoS guarantee.

6. A complete implementation of the proposed mechanisms based on the LoRaWAN specification v1.0.4 and v1.1 that is available as open-source software. The implementation allows third-parties to develop novel services that utilise all the resources available in the Cloud-Edge-Device processing and storage continuum. The service provider can deploy the CECC-native EDs and GWs in existing public LoRaWAN infrastructures, e.g., such as the TheThingsNetwork.

7. A multifaceted performance evaluation is conducted to assess the correctness, security and network utilization of the new system. The services are deployed in real-world devices in combination with an IoT device emulator to recreate massive IoT application scenarios. To demonstrate that legacy and CECC-native services can co-exist without any disruption or performance degradation, CECC-native EDs and GWs are deployed in an existing public LoRaWAN infrastructure.

In the following chapter (Chap. 2), the objectives of my doctoral research and the outline of this thesis are addressed in detail.

Chapter 2

Objectives

My doctoral research centered on optimizing the network performance, security, and scalability of Low-Power Wide-Area Network (LPWAN), especially Long-Range Wide-Area Network (LoRaWAN). Given the escalating deployment of Internet of Things (IoT) applications and devices worldwide, the burgeoning volume of sensor data demands advancements in these areas. Edge computing offers a compelling solution to mitigate cloud burdens, alleviate network congestion, expedite response times for end-users, and bolster security and privacy.

The primary objective of my research is to enable Cloud Edge Computing Continuum (CECC) over LoRaWAN while ensuring data integrity, confidentiality, and privacy. To achieve this goal, the research is structured into three key phases: (1) conducting a comprehensive analysis of the state of the art and the LoRaWAN protocol to identify relevant requirements and challenges; (2) developing a novel CECC protocol specifically tailored for LoRaWAN environments; and (3) integrating distributed applications and exploring potential improvements driven by the needs of real-world use cases. This phased approach facilitates a systematic exploration of the limitations of existing protocols and the development of scalable, secure solutions for edge computing over LoRaWAN.

Phase 1: In-Depth LoRaWAN Protocol Analysis and Requirement Identification

In the initial phase of this research, a comprehensive analysis of the state of the art (Chap. 3) and the LoRaWAN protocol was conducted, focusing on its architecture, security mechanisms, and operational characteristics (Chap. 4). This in-depth review enabled me to pinpoint the specific requirements and challenges that must be addressed to achieve secure and confidential CECC over LoRaWAN (Chap. 5).

By identifying these key areas, I laid the groundwork for developing solutions that strengthen data privacy, enhance security protocols, and ensure the efficient use of computational resources in edge environments. This foundation was essential for overcoming the inherent limitations of LoRaWAN in supporting more advanced applications and secure data handling.

Phase 2: Development of a Novel CECC Protocol for LoRaWAN

Building upon the insights gained from the first phase, I developed and proposed a novel CECC protocol specifically designed for LoRaWAN environments, named Edge2LoRa (E2L). This protocol integrates advanced cryptographic techniques, privacy preserving mechanisms, and efficient communication protocols to ensure the security, integrity, and confidentiality of data processed at the network edge (Chap. 6).

A complete open-source implementation of E2L is available on GitHub[1]. The implementation (Sec. 7.1) has been rigorously tested and is compatible with both real-world and simulated devices, validating the correctness, scalability, and efficiency of the proposed protocol. Testing with real hardware demonstrates its practical applicability, while simulated environments allow for the evaluation of the protocol's performance in large-scale and high-density scenarios (Sec. 7.2). This dual approach ensures that E2L can handle a wide range of use cases, from small deployments to massive IoT networks, offering both security and operational efficiency.

Phase 3: Integration of Distributed Applications & Improvements

The third phase of this research focuses on identifying and integrating a real-world use case into the E2L architecture. This phase aims to leverage the distinct advantages of a distributed application, enabled by CECC, highlighting the significant benefits of this approach in contrast to traditional centralized models, such as those commonly used in LoRaWAN. By employing a distributed framework, scalability is enhanced, fault tolerance is improved, and resource utilization becomes more efficient, ultimately leading to better performance in edge computing environments.

The development of these use cases has driven substantial enhancements in the E2L framework, culminating in an upgraded version known as Edge4LoRa (E4L). This new iteration not only incorporates additional functionalities but also features a refined architecture specifically designed to overcome several limitations identified during the research process (Chap. 8). These improvements include optimized data handling and better support for diverse application scenarios.

The implementation of this innovative solution is available in the same GitHub repository[1], offering researchers and practitioners access to the enhanced framework for further experimentation and application across various scenarios (Sec. 8.4). By making this resource available, I aim to encourage collaboration and exploration of the potential applications of E4L in real-world settings.

The thesis concludes (Chap. 9) with a summary of the results achieved and an overview of the foreseen improvements for the newly proposed framework.

[1] `https://github.com/Edge2LoRa`

Chapter 3

Related Works

Long-Range Wide-Area Network (LoRaWAN) has received significant attention during the past years since it is a crucial and promising Low-Power Wide-Area Network (LPWAN) technology [58]. A comprehensive review of LoRaWAN from the perspective of network operation, security, applications, modelling and experimentation the interested reader is presented in [62].

On the other hand, during the past couple of years only a handful of studies have been proposed to address the centralized architecture of the LoRaWAN v1.1 network standard. Zhou et al. [68] propose to decouple the functionalities of the LoRaWAN Network Server (NS) in four modules: connector, central server, join server and network controller. Such a split can indeed assist in distributing the centralized functionalities to different locations within the network. The scalability however of this approach remains limited, the functionalities that can be decentralized are related to the network management and operation, while at the same time the proposed extensions are not compatible with existing, public LoRaWAN deployments.

Liu et al. [38] propose an extension to Long-Range (LoRa) network standard that allows to migrate the functions of rejoin and media access control (MAC) commands into the LoRaWAN Gateway (GW). Since the GWs are required to store data related to the network operation and a data synchronization mechanism is introduce to allow the NS to have a complete view of the system. Like with the previous work, this allows to mitigate the workloads of the central cloud that have to do with network management and operation. Once again, the proposed extensions are not backward compatible with the current LoRaWAN standard. More recently, Lu et al. [31] proposes to migrate the functions of rejoin and media access control (MAC) commands into the GW however instead of using a local database, a blockchain network with multiple ledgers is designed. The resulting system improves the overall scalability of the network operations by utilising the Edge resources available at the GW, however yet again it is not backward compatible with the current LoRaWAN standard. The LoRaCTP is a flexible protocol based on LoRa that enables data transfer over large distances with very low energy expenditure over a generic FrUgal eDGE (FUDGE)/fog architecture [45]. The solution provides all necessary mechanisms for LoRaWAN reliability and allows edge computing via a lightweight connection, it is not however compatible with existing LoRaWAN deployments.

The ability to introduce Edge/Fog processing for the analysis of the payload of LoRaWAN frames is studied in [6, 24]. The proposed architectures utilise the computational resources already available at the GW deployed to facilitate the analysis of IoT data in smart water distribution networks. A distributed key management system is introduced that shares the Application Session Encryption Key (*AppSEncKey*) with the GWs so that they can decrypt and analyze the application-level messages received. Both studies indicate significant reductions to the overall load to network and cloud resources. However transmitting the *AppSEncKey* to the disparate GWs introduces new vulnerabilities to the system. In contrast to these approaches, the solution presented in this thesis does not require the exchange of any secret key across the network.

The design of a LoRaWAN deployment in a smart campus enabling fog computing on the LoRaWAN gateways giving them the LoRaWAN servers capabilities is presented in [21]. The evaluation shows the benefits of the use of fog computing however the solution proposed is not compatible with existing public networks and its scalability is limited since the functionalities of the GWs and NS are collapsed in a single entity. The focus of the authors was limited to the use of fog computing nodes to support low-latency and location-aware IoT applications, and the evaluation was conducted on a particular smart campus. As a result, there are no indications provided on the scalability of the solution.

A different approach is followed in [20] to minimise upstream network traffic in cases of overlapping network areas covered by multiple nearby GWs. In LoRaWAN frames received by one or more GW are forwarded multiple times through the network backbone to a central NS. To reduce the duplicate transmission, each LoRaWAN End Device (ED) selects the nearby GW with which it has the best signal quality to act as a so-called Rendezvous Point where analysis of the frames payload can also take place. In dense deployments this approach has certain limitation since transmitting large number of down-link frames leads to high loss rates. In contrast to this approach, a scalable Edge2 Gateway (E2GW) selection is proposed that does not require the exchange of any additional down-link frames.

The existing literature emphasizes the integration of processing modules at the edge layer, where data originate, and researchers have been actively exploring and evaluating novel solutions for edge data processing. One notable reference work in the field of edge computing is EDGEWISE [22], which introduces a stream processing engine specifically designed for edge computing environments. The authors demonstrate that EDGEWISE, with its congestion-aware scheduler and pool of fixed-sized workers, significantly enhances performance across various metrics.

Moving in the context of IoT, the authors in [53] present an edge-based programming framework for stream processing in IoT environments. The authors highlight the inefficiency of traditional core-centric data processing methods due to data transfer latencies, particularly for time-sensitive applications. In addition, a framework called Seagull is presented in [11], which focuses on building efficient, large-scale IoT-based applications. Through a smart city scenario, the authors showcase the impact on throughput by varying the amount of processing at the edge and performing various streaming analyses such as filtering or aggregation.

Despite the significant contributions from these works, none of these solutions are specifically tailored for LoRaWAN environments.

Chapter 4

Background: LoRa & LoRaWAN

Before introducing the Edge2LoRa (E2L) and Edge4LoRa (E4L) Network, it is essential to first outline the model, architecture, and key aspects of Long-Range (LoRa) and Long-Range Wide-Area Network (LoRaWAN). This will provide a foundational understanding of their inherent limitations and clarify the improvements that this thesis aims to introduce.

In this chapter, I will introduce the LoRa physical layer and the LoRaWAN architecture as defined in the current specification [4]. This will include a detailed explanation of the roles of each network component, the various LoRaWAN End Device (ED) classes, and the available methods for ED activation.

4.1 LoRa & LoRaWAN

LoRaWAN is one of the most widely used Low-Power Wide-Area Network (LPWAN) technologies, developed by the LoRa Alliance as an open standard operating in unlicensed frequency bands [4]. It utilizes the LoRa physical communication layer, which employs a proprietary modulation technique known as LoRa Modulation—a derivative of Chirp Spread Spectrum (CSS). This technique operates in the Sub-GHz bands and is developed and distributed by Semtech.

LoRaWAN leverages Adaptive Data Rate (ADR) and various Spreading Factor (SF) ranging from 7 to 12, allowing for multiple simultaneous data rates and messages on the same channel [39]. Each SF defines the data rate and maximum payload size, based on the ratio between chip and symbol rates [48]. Data rates range from as low as 22 bps to as high as 27 kbps, while maximum payload sizes vary from 51 to 222 bytes, depending on the SF used [1].

4.2 LoRa Uplink Frame Structure

The LoRaWAN uplink frame structure is designed to facilitate communication between ED and the network through LoRaWAN Gateways (GWs), ensuring secure and efficient data transmission. The frame structure can be broken down into several key components as shown in Table 4.1.

Table 4.1. LoRaWAN uplink frame structure

`Preamble`	`PHDR`	`PHDR_CRC`	`PHYPayload`	`CRC`
6 – 65535 bits	1 Byte	1 – 2 Bytes	1 – 255 Bytes	4 Bytes

- **Preamble**: the initial part of the frame used to synchronize the transmitter and receiver. It helps the receiver detect the beginning of the frame and is critical for precise timing in low-power operations.
- **Physical Header (PHDR)**: contains information about the packet's properties, such as spreading factor, coding rate, bandwidth, CRC presence.
- **Physical Header CRC (PHDR_CRC)**: check for the `PHDR`, ensuring its integrity 1-2 bytes (8-16 bits).
- **Physical Payload (PHYPayload)**: contains the frame actual payload. Composed of a header and a payload,
- **CRC**: integrity check for the `PHYPayload`.

Physical Payload

The physical payload (PHY Payload) contains several fields, which are crucial for the operation of the LoRaWAN protocol. These include:

Table 4.2. LoRaWAN Physical Payload structure

`MHDR`	`MACPayload`	`MIC`
1 Byte	0 – 255 Bytes	4 Bytes

- **MAC Header (MHDR)**: defines the Message type and its format. The structure is presented in Table 4.3.

Table 4.3. The structure of the MAC Header. The first row shows the fields composing the header, while the second row indicates the bits position for each field.

`MType`	`RFU`	`Major`
7..5	4..2	1..0

  Table 4.4 shows the possible values for the `MType` field.

- **MAC Payload (MACPayload)**: carries either the application data or control commands. Contains a frame header (`FHDR`) followed by an optional port field (`FPort`) and an optional frame payload field (`FRMPayload`).
- **Message Integrity Code (MIC)**: 4-byte field that ensures the integrity and authenticity of the message

Table 4.4. MAC message type.

MType	Description
000	Join Request
001	Join Accept
010	Unconfirmed Uplink Data
011	Unconfirmed Downlink Data
100	Confirmed Uplink Data
101	Confirmed Downlink Data
110	Rejoin Request
111	Propetary

MAC Payload

The main body of the message, which carries either the application data or control commands. Table 4.5 shows the structure.

Table 4.5. The structure of the MAC Payload.

FHDR	FPort	FRMPayload
7 – 22 Bytes	0 – 1 Bytes	Variable size

- **Frame Header (FHDR)**: includes information such as the device address (`DevAddr`), frame control (`FCtrl`) field, and frame counter (`FCnt`). The `FOpts` field id used to transport MAC commands, if present shall be encrypted using the Network Session key. Table 4.6 shows its structure.

Table 4.6. The structure of the Frame Header.

DevAddr	FCrl	FCnt	FOpts
4 Bytes	1 Bytes	2 Bytes	0 – 15 Bytes

- **Frame Port (FPort)**: optional 1-byte field that identifies the application port. Ranging from 1 to 223 for application-specific data and port 0 for MAC commands.

- **Frame Payload (FRMPayload)**: the actual payload of the frame, containing either user data or MAC commands. This field is encrypted using the application session keys.

4.3 LoRaWAN Architecture

The main actors in a LoRaWAN 1.1 Network [4] are the ED, the GW, the LoRaWAN Network Server (NS), the LoRaWAN Join Server (JS) and the LoRaWAN Application Server (AS), as shown in **Fig. 4.1**.

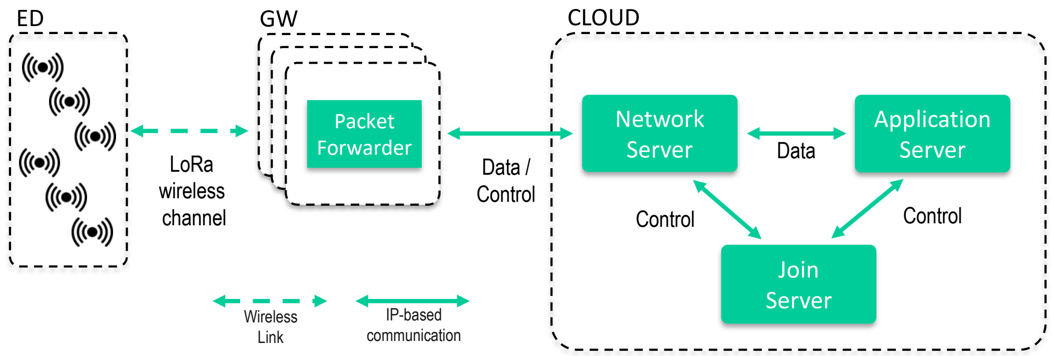


Figure 4.1. Architecture of the LoRaWAN v1.0.4 and v1.1 Network Standards [4]. The figure illustrates the interrelations between the five key components: the LoRaWAN End Devices (EDs), the LoRaWAN Gateways (GWs), the LoRaWAN Network Server (NS), the LoRaWAN Join Server (JS) and the LoRaWAN Application Server (AS). The GWs bridge traffic between the EDs and the NS without accessing the frame payload. The NS manages the network operations, the JS provides authentication, authorization, and access control to the EDs while Internet of Things (IoT) data streams are processed centrally by the AS.

- **LoRaWAN End Device (ED)** often a small and cheap device that periodically collects data from its sensors and broadcasts them using LoRa. It has limited power computation capabilities to maintain its cost as low as possible and offer the most extended lifespan possible, given that it is often battery-powered.

- **LoRaWAN Gateway (GW)**: acts as a concentrator that collects frames from the ED in its radio range and forwards those to the NS. It can connect with many ED but can not access the payload of the received frames. It has two different network interfaces: one for LoRa and one for Internet Protocol (IP) based communications. The subsequent communications are conducted via IP-based technologies; LoRa physical layer use is limited to exchanging frames between the ED and the GW.

- **LoRaWAN Network Server (NS)**: it manages the entire network by dynamically controlling network parameters, such as the SF and the Data Rate (DR), to respond to changing conditions. The NS ensures the authenticity of each ED joining the network and checks the integrity of each frame. However, the NS can not access the application data.

- **LoRaWAN Join Server (JS)**: it manages the Over-The-Air Activation (OTAA) (**Sec. 4.5**) method. It stores the information required to process the Join-Request frame and to generate the Join-Accept in response to it after performing the Network Session and Application Sessions Keys derivation. It is responsible for notifying the NS to which AS an ED shall be connected.

- **LoRaWAN Application Server (AS)**: it securely handles, manages and interprets the application data. It is the only actor that can access the application payload sent by the ED.

4.4 LoRaWAN End Devices Classes

LoRaWAN defines three types of ED [4]:

- **Class A:** The ED opens two receive windows after an uplink message. The second receive window is opened only if no downlink frame is received during the first. It is possible to send a single downlink frame to this type of ED for each uplink frame they send during those receive windows.
- **Class B:** The GW, via beacons, synchronises periodic receive widows of the EDs in its radio range. It is possible to send downlink frames to those EDs even when they do not send uplink frames; however, the overall energy consumption increases.
- **Class C:** The ED continuously listens for downlink frames except when it broadcasts an uplink message. This class is the most demanding energy-wise.

The Class A EDs have the most strict constraint. However, they are the most common since they are the most energy efficient.

4.5 LoRaWAN End Devices Activation Methods

In LoRaWAN, the encryption of the payload is, by default, enabled in every transmission. The LoRaWAN 1.1 specification specifies two methods to agree on a set of session keys.

Activation by Personalization Method – ABP

The Activation By Personalization (ABP) method is the most straightforward; however, it is not secure and shall be used exclusively in the early stages of network design tests. The ABP activation method consists of hard-coding the session keys in all the interested actors (ED, JS).

Over-The-Air Activation Method – OTAA

The OTAA method is more sophisticated than ABP and allows a higher level of security. In literature, lots of work presents some vulnerabilities in the OTAA [10, 14, 15, 17], offering alternative solutions or mitigation [29, 41, 57, 64]. **Fig. 4.2 & Algorithm 1** summarise the steps of the OTAA methods.

Before the activation, two *64-bit Extended Unique Identifiers* shall be assigned to the ED and the JS, respectively, the Device Extended Unique Identifier (DevEUI) and the Join Extended Unique Identifier (JoinEUI).

Two root keys are hard-coded into the device: the Network Key (NwkKey) and the Application Key (AppKey). These root keys are then used to derive two additional keys: the Join Specific Encryption Key (JSEncKey) and the Join Specific Integrity Key (JSIntKey). The JSEncKey is used to encrypt the JoinAccept frame, while the JSIntKey is responsible for computing the Message Integrity Code (MIC) for that frame

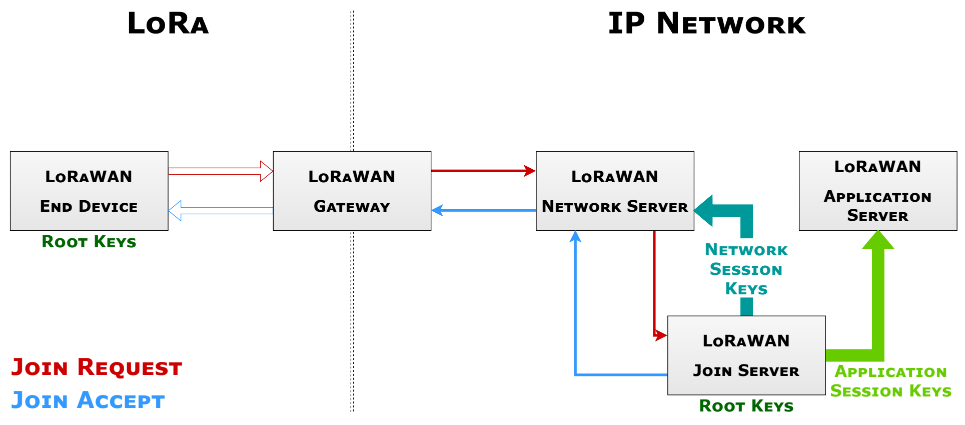


Figure 4.2. LoRaWAN 1.1 Over-The-Air-Activation Method

1. The ED and the JS share the `Root Keys` and the information about the DevEUI and the JoinEUI. Moreover, the JS knows to which AS the ED shall be connected.

2. The ED triggers the activation process via a Join Request frame containing its DevEUI, the JoinEUI and a DevNonce. The ED computes the MIC using a key derivated from the `Root Keys`.

3. The JS checks the MIC and the validity of the pair DevEUI and JoinEUI. If the checks succeed, it replies with a **Join Accept** containing a `JoinNonce`. The frame is encrypted with the JSEncKey, and its MIC computed with the JSIntKey.

4. The ED and the JS can compute the Network and Application Session Keys using the two `nonces`, the `EUIs` and the `Root Keys`.

5. The JS distribute the `Network Session Keys` to the NS and the `Application Session Keys` to the AS.

Table 4.7 summarise the number of up-link and down-link frames exchanged to complete the OTAA method.

Table 4.7. Number of Frames exchanged during the OTAA

Type of Frame	Number of Frames
Uplink LoRaWAN	1
Downlink LoRaWAN	1
Minimum IP frames	6

Algorithm 1 OTAA according to LoRaWAN 1.1 specification

```
  ACTOR: ED
Require:
  Hard-Coded Root Keys: $NwkKey,\ AppKey$
  Hard-Coded JoinEUI: 64-bit Extended Unique Identifier of the Join Server
  DevEUI: its 64-bit Extended Unique Identifier

  upon event $< OTAA,\ Init >$ do
      $NwkKey :=$ from hard-coded Root Keys
      $AppKey :=$ from hard-coded Root Keys
      $JSIntKey = aes128\_encrypt(NwkKey, 0x06|DevEUI|pad16)$
      $JSEncKey = aes\_encrypt(NwkKey, 0x05|DevEUI|pad16)$
      $FNwkSIntKey := \perp$
      $SNwkSIntKey := \perp$
      $NwkSEncKey := \perp$
      $AppSKey := \perp$
      $DevNonce :=$ 2 Bytes Counter
      $jr = JoinEUI|DevEUI|DevNonce$
      trigger $< LoRaReceive,\ JoinRequest|jr,\ GW >$
  end upon event

  upon event $< LoRaReceive,\ JoinAccept|ja >$ do
      $ja = eas128\_encrypt(JSEncKey, ja)$
      $JoinNonce := ja[0:3]$
      $FNwkSIntKey = aes128\_encrypt(NwkKey, 0x01|JoinNonce|JoinEUI|DevNonce|pad16)$
      $SNwkSIntKey = aes128\_encrypt(NwkKey, 0x03|JoinNonce|JoinEUI|DevNonce|pad16)$
      $NwkSEncKey = aes128\_encrypt(NwkKey, 0x04|JoinNonce|JoinEUI|DevNonce|pad16)$
      $AppSKey = aes128\_encrypt(AppKey, 0x02|JoinNonce|JoinEUI|DevNonce|pad16)$
  end upon event

  ACTOR: GW
Require:
  JoinEUI and IP address of the respective Network Server

  upon event $< OTAA,\ Init >$ do
      $ED := \perp$
  end upon event

  upon event $< LoRaReceive,\ JoinRequest|jr >$ do
      $ED := jr[8:16]$
      trigger $< Receive,\ JoinRequest|jr,\ NS >$
  end upon event

  upon event $< Receive,\ JoinAccept|ja >$ do
      trigger $< LoRaReceive,\ JoinAccept|ja,\ ED >$
  end upon event
  ACTOR: NS
  upon event $< OTAA,\ Init >$ do
      $JS :=$ Respective Join Server
      $GW := \perp$
      $FNwkSIntKey := \perp$
      $SNwkSIntKey := \perp$
      $NwkSEncKey := \perp$
  end upon event

  upon event $< Receive,\ JoinRequest|jr,\ gw >$ do
      $GW := gw$
      trigger $< Receive,\ JoinRequest|jr,\ JS >$
  end upon event

  upon event $< Receive,\ JoinAccept|ja,\ js >$ do
      trigger $< LoRaReceive,\ JoinAccept|ja,\ GW >$
  end upon event

  upon event $< Receive,\ NetworkSessionKeys|NwkSKeys,\ js >$ do
      $FNwkSIntKey := NwkSKeys \rightarrow FNwkSIntKey$
      $SNwkSIntKey := NwkSKeys \rightarrow SNwkSIntKey$
      $NwkSEncKey := NwkSKeys \rightarrow NwkSEncKey$
  end upon event

  ACTOR: AS
Require: DevEUI: Indentified of End Device
  upon event $< OTAA,\ Init >$ do
      $AppSKey := \perp$
  end upon event

  upon event $< Receive,\ ApplicationSessionKey|key,\ js >$ do
      $AppSKey := key$
  end upon event

  ACTOR: JS
Require: Hard-Coded Root Keys: $NwkKey,\ AppKey$
         Hard-Coded JoinEUI: its 64-bit Extended Unique Identifier
         DevEUI: 64-bit Extended Unique Identifier of the End Device
         Application Server linked to the ED as AS
  upon event $< OTAA,\ Init >$ do
      $NwkKey :=$ from hard-coded Root Keys
      $AppKey :=$ from hard-coded Root Keys
      $JSIntKey = aes128\_encrypt(NwkKey, 0x06|DevEUI|pad16)$
      $JSEncKey = aes\_encrypt(NwkKey, 0x05|DevEUI|pad16)$
      $FNwkSIntKey := \perp$
      $SNwkSIntKey := \perp$
      $NwkSEncKey := \perp$
      $AppSKey := \perp$
  end upon event

  upon event $< Receive,\ JoinRequest|jr,\ ns >$ do
      $JoinEUI :=$ 3 Bytes Device specific Counter
      $ja = JoinNonce|HomeNet\_ID|DevAddr|DLSettings|RxDelay|CFList$
      $ja = eas128\_decrypt(JSEncKey, ja)$
      trigger $< Receive,\ JoinAccept|ja,\ NS >$
      $FNwkSIntKey = aes128\_encrypt(NwkKey, 0x01|JoinNonce|JoinEUI|DevNonce|pad16)$
      $SNwkSIntKey = aes128\_encrypt(NwkKey, 0x03|JoinNonce|JoinEUI|DevNonce|pad16)$
      $NwkSEncKey = aes128\_encrypt(NwkKey, 0x04|JoinNonce|JoinEUI|DevNonce|pad16)$
      $AppSKey = aes128\_encrypt(AppKey, 0x02|JoinNonce|JoinEUI|DevNonce|pad16)$
      $NwkSKeys := (FNwkSIntKey, SNwkSIntKey, NwkSEncKey)$
      trigger $< Receive,\ NetworkSessionKeys|NwkSKey,\ NS >$
      trigger $< Receive,\ ApplicationSessionKey|AppSKey,\ AS >$
  end upon event
```

Chapter 5

Limitations, Possible Improvements & Requirements

LoRaWAN is one of the most promising and used LPWANs, the benefits listed in **Sec. 4** make that technology particularly suitable for Wireless Sensor Network (WSN). The following summarises the main limitation of LoRa and LoRaWAN:

1. Limited Payload size
2. Low Data Rate
3. Strict Duty Cycle restiction
4. Class A ED receive windows (**Sec. 4.4**)
5. The GW can not access the payload, so performing edge or fog computation over LoRaWAN is impossible.

The limitations described in `Point 1, 2 & 3` arise from the use of LoRa as physical layer. These constraints are crucial to ensuring long-range and energy-efficient communication. However, to expand the range of applications for which LoRaWAN can be used, these limitations can be addressed by switching the physical layer from LoRa to Long-Range 2.4GHz (LoRa 2.4GHz) [32].

LoRa 2.4GHz reduces the range compared to traditional LoRa but significantly enhances network capacity by enabling higher data rates and eliminating the need to comply with any duty cycle restrictions. Specifically, the data rates for LoRa range from $0.3kbit/s$ to $50kbit/s$ depending on the SF used [1], whereas LoRa 2.4GHz supports data rates ranging from $0.595kbit/s$ to $253.91kbit/s$. This higher data rate allows LoRa 2.4GHz to facilitate the use of LoRaWAN for applications requiring lower latency, higher data rates, and non-scheduled packet transmissions.

On the other hand, the limitation in `Point 4` significantly affects the energy efficiency of the ED. Devices in `Class B` or `Class C` have relaxed or no constraints on downlink frames, respectively, which increases energy consumption and thus reduces the lifetime of battery-powered devices.

The improvements for the first four limitations are beyond the scope of this thesis. However, it is crucial to highlight them because any solution aimed at improving LoRaWAN technology must conform to these constraints.

Conversely, I consider `Point 5` to be a significant limitation. Edge/Fog Computing is becoming increasingly critical for deploying efficient and cost-effective WSN, offering well-documented benefits [44, 67]. These advantages include mitigating network congestion caused by the exponential growth of data exchanged as the number of IoT devices increases, reducing the workload on cloud servers, and providing faster response times to end-users due to geographically proximate computation. Currently, state-of-the-art LoRaWAN GWs function merely as concentrators, forwarding frames to the NS.

This thesis proposes a new paradigm to overcome these constraints and enable Cloud Edge Computing Continuum (CECC). In the following section, I will investigate this point in detail, outlining a set of solutions ranging from the most straightforward to the more complex. Additionally, I will collect and analyze the necessary requirements to ensure that the proposed solutions are not only comprehensive and feasible but also aligned with the practical and operational needs of the LoRaWAN ecosystem. This approach will help to address the limitations effectively while maintaining the integrity and performance of the network.

5.1 Simple Edge Architecture

A straightforward solution to enable CECC is to consolidate the functionalities of the GW, NS, JS, and AS into a single entity; some commercial GWs offer such an integrated approach. **Fig. 5.1** illustrates two examples of possible LoRaWAN architectures that can be employed to implement this solution. In both cases, the ED will interact directly with the GW, which also has server capabilities, allowing it to access the payload of both the Network and Application LoRaWAN frames from the device. However, this solution presents several issues:

- The ED will be able to exchange Network and Application LoRaWAN frames solely with the GW to which it is activated.
- The OTAA uses a pair of root keys that need to be hard-coded in both the actors. The ED will not be able to activate with other GWs or NSs without compromising the overall security of the protocol.
- The GW with server capabilities represents a single point of failure. If it crashes, the communication with the ED will be interrupted.
- The solution does not support mobility of the ED and the GW.
- GW computation capabilities are often comparable to a Raspberry Pi and serve many EDs. Using such a solution will increase their workload since they will perform Edge Computing, manage the network, set network parameters for each device linked to them, and store many cryptography materials. Such implication can decrease the number of devices a GW can serve, so it is necessary to increase the number or the computational capabilities of the GWs. Consequently, the cost of deploying and maintaining a LoRaWAN network deployment could rise.

- The solution requires a change at the architectural level of the LoRaWAN Network, causing it to be incompatible with the existing and public LoRaWAN network.

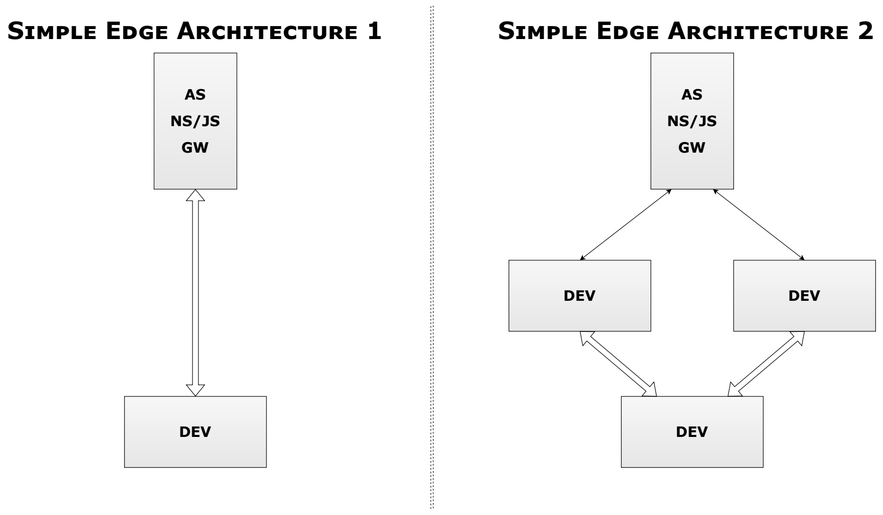


Figure 5.1. Example of possible LoRaWAN Architectures for Solution 1

Collected Requirements

- The GWs shall not be a single point of failure.
- The protocol shall consider the possibility of mobility of both EDs and GWs.
- One of the main advantages of Edge Computing is to optimize the computation, distributing it over the IoT network without increasing network management costs.
- The compatibility between existing and public networks and existing LoRaWAN networks shall be guaranteed. This means that actors using the classical LoRaWAN protocol and the edge-enabled ones shall coexist in the same LoRaWAN Network.

5.2 Group Application Keys Agreement

To enable Edge/Fog Computing on the GWs, the latter need to decrypt and access the Application LoRaWAN frame payload while avoiding becoming a single point of failure and ensuring backward compatibility. One potential solution is to enhance the OTAA method to establish session Application Keys shared among the ED, the GWs within the device's range, and the AS. By encrypting the Application LoRaWAN frames with these shared keys, the ED allows every GW within its radio range, as well as the AS, to access the payload. This approach facilitates the processing of data at the edge while preserving the flexibility and scalability of the LoRaWAN network.

This approach enables CECC in public networks by including only trusted GWs in the key agreement process, with these trusted GWs maintained in an Active Directory on the AS. It also ensures backward compatibility, as a GW can still use the standard LoRaWAN protocol to forward the frame to the NS.

However, if an ED is within the coverage area of two or more GWs, each of these gateways can access the payload and perform the necessary computation. This can result in the same information being processed and transmitted multiple times, potentially leading to overlapping network activities and redundant data handling.

The following algorithms present different approaches to agreeing on a set of Application Session Keys shared among the ED, the AS, and a set of trusted GWs.

Algorithm 2 presents the pseudo-code in case the Group Key Agreement is executed after completing the OTAA. In such a case, the ED, the AS and the set of trusted GWs shall execute a new protocol on top of the OTAA. This will increase the energy consumption of the ED, the number of total LoRaWAN frames exchanged, and the elapsed total time needed for the activation.

Algorithm 2 GKA after OTAA

```
Require: $ED, NS, JS, AS$
      Set of trusted Gateways: $GW_T = \{GW_{T_1}, GW_{T_2}, \dots, GW_{T_m}\}$
  $OTAA(ED, NS, JS, AS)$
  $groupSharedAppSKeys = GKA(ED, GW_{T_1}, GW_{T_2}, \dots, GW_{T_m}, AS)$
```

Table 5.1 summarise the number of LoRaWAN up-link and down-link frames exchanged to complete the Group Key Agreement after the OTAA.

Table 5.1. Number of Frames exchanged during the GKA After OTAA

Type of Frame	**Number of Frames**
Minimum Uplink LoRaWAN	2
Minimum Downlink LoRaWAN	2
Minimum IP frames	$6 + m$

Algorithm 3 presents a tentative solution (**Gateway-Integrated OTAA**) in which the Application Sessions Keys are provided to the set of trusted GWs by the JS. In such an approach, the GWs do not contribute to generating the above keys. For simplicity the algorithm shows only the delta between the OTAA and the `Gateway-Integrated OTAA`.

Table 5.2 summarise the number of LoRaWAN up-link and down-link frames exchanged to complete the Gateway-Integrated OTA Activation process.

Table 5.2. Number of Frames exchanged during the GW-Integrated OTAA

Type of Frame	**Number of Frames**
Uplink LoRaWAN	1
Downlink LoRaWAN	1
Minimum IP frames	$6 + m$

Algorithm 3 Gateways-Integrated OTAA

```
  ACTOR: JS
Require: Set of trusted Gateways: $GW_T = \{GW_{T_1}, GW_{T_2}, \ldots, GW_{T_m}\}$
  upon event $< Receive,\ JoinRequest|jr,\ ns >$ do
    $JoinEUI$ := 3 Bytes Device specific Counter
    $ja = JoinNonce|HomeNet\_ID|DevAddr|DLSettings|RxDelay|CFList$
    $ja = eas128\_decrypt(JSEncKey, ja)$
    trigger $< Receive,\ JoinAccept|ja,\ NS >$
    $FNwkSIntKey = aes128\_encrypt(NwkKey, 0x01|JoinNonce|JoinEUI|DevNonce|pad16)$
    $SNwkSIntKey = aes128\_encrypt(NwkKey, 0x03|JoinNonce|JoinEUI|DevNonce|pad16)$
    $NwkSEncKey = aes128\_encrypt(NwkKey, 0x04|JoinNonce|JoinEUI|DevNonce|pad16)$
    $AppSKey = aes128\_encrypt(AppKey, 0x02|JoinNonce|JoinEUI|DevNonce|pad16)$
    $NwkSKeys := (FNwkSIntKey, SNwkSIntKey, NwkSEncKey)$
    trigger $< Receive,\ NetworkSessionKeys|NwkSKey,\ NS >$
    trigger $< Receive,\ ApplicationSessionKey|AppSKey,\ AS >$
    for $GW_{T_i} \in GW_T$ do
      trigger $< Receive,\ ApplicationSessionKey|AppSKey,\ GW_{T_i} >$
    end for
  end upon event

  ACTOR: GW
Require: DevEUI: Indentified of End Device
  upon event $< OTAA,\ Init >$ do
    $AppSKey := \perp$
  end upon event

  upon event $< Receive,\ ApplicationSessionKey|key,\ js >$ do
    $AppSKey := key$
  end upon event
```

Collected Requirements

- Two or more Trusted GWs shall not execute the same computation on the same frame or group of frames.
- No Edge processing should be duplicated. Each LoRaWAN frame should be processed by only one Edge-Enabled Gateway. This will prevent overlapping networks and reduce potential congestion caused by redundant information.

Chapter 6

Enabling Cloud Edge Computing Continuum in LoRaWAN

Cloud Edge Computing Continuum (CECC) offers a natural evolution of information technology provisioning of computation and storage, which was traditionally bound to centralized data centres, to include resources available at the edges of the network [44, 58]. In the case of LoRaWAN that adopts the design approach of using simple protocols to realise a centralised architecture, one faces multiple implications when trying to incorporate it in the CECC. Forcing the GWs to act as simple bridges, at one hand accommodates the rapid and low-cost deployment of unlicensed LPWANs, on the other hand excludes them from potentially acting as trusted intermediate processing and storage elements in the CECC. Recently some researchers have explored different approaches in modifying the specified operation of the protocols, proposing alternative architectures to reduce the significant pressure imposed on the central cloud services in cases of massive IoT data streams or time-constrained consumption of IoT data [21, 31, 68]. At the same time, introducing changes to the architecture and the specified operation of the protocols while maintaining backward compatibility is a challenging discourse [45].

I here propose a new approach that enables for the first time the efficient and secure integration of LoRaWAN in the CECC while respecting the current specifications and maintaining backward compatibility, thus allowing seamless interoperability between legacy and new elements. In other words, I allow the co-existance of cloud-centric analytics services with the execution of novel services that build upon edge computing concepts that require certain *Quality of Service (QoS) guarantees*. Consider that in the common case where IoT applications rely over public LoRaWAN infrastructures, like for example the TheThingsNetwork[1], it is impossible to expect that the operator will support extended versions of the LoRaWAN Network Server (NS) and/or the LoRaWAN Join Server (JS) that deviate from the current standards and may lead to service disruptions for already deployed devices and services. For this reason, I guarantee that my extensions can be fully integrated in public LoRaWAN infrastructures: CECC-native GWs can co-exist with legacy GWs

[1]`https://www.thethingsnetwork.org/`

and communicate with a legacy NS and a legacy join server JS by respecting the protocols defined in the current LoRaWAN specifications. I also guarantee that the legacy IoT devices already deployed in the public LoRaWAN infrastructure will not experience any service degradation when CECC-native elements are deployed. The extended infrastructure of legacy and CECC-native GWs can support multiple IoT applications, allowing the cooperative allocation of processing and storage of CECC-native resources. Only for the special case where an IoT use case requires that multiple QoS guarantees are respected concurrently, e.g., involving notions of exactly-once and at-most-once processing of messages, the deployment of an NS with extended functionalities is required.

The Edge2LoRa (E2L) [42] solution evolves the LoRaWAN architecture by introducing several elements: the device and GW registries, the table of device-gateway associations, the GW selection module, the group key agreement module, the edge processing executor module, the QoS support mechanism and the driver module. Fig. 6.1 depicts how these newly introduced elements interconnect with the components currently defined in the LoRaWAN network standard. Notice that the new components that constitute the E2L architecture are depicted in blue color, while the components already defined by the LoRaWAN standard are depicted in green color. Remark that the standard components, i.e. those in green color, are also in Fig. 4.1.

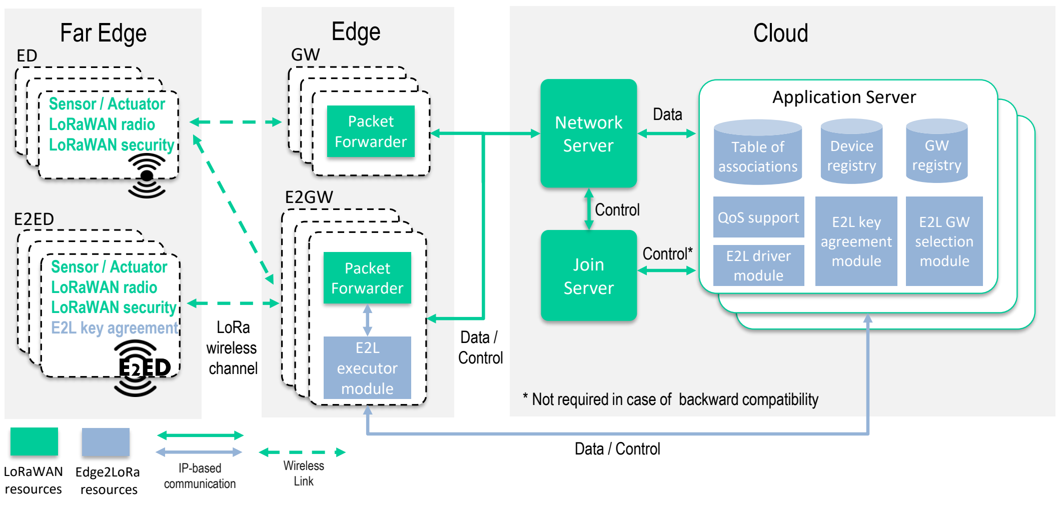


Figure 6.1. Overview of the proposed E2L (E2L) architecture including the key components of the current LoRaWAN network standard along with the ones introduced in E2L and their interconnections. The diagram depicts the data flow for frames transmitted by an E2L end device (Edge2 End Device (E2ED)) that is received by an E2L GW (Edge2 Gateway (E2GW)), which can be processed locally before forwarding through the E2L driver to the AS (blue data link). Concurrently, frames transmitted by an E2ED and received by a legacy GW, or frames transmitted by a legacy ED and received by an E2GW, are forwarded to the NS (green data link), following the current LoRaWAN network standard.

The *far-edge layer* comprises legacy LoRaWAN IoT EDs and EDs that are compatible with the E2L framework, denoted as E2ED. At *the edge layer*, the legacy LoRaWAN GWs provide the *Packet Forwarder module*, as defined in the LoRaWAN network standard, along with GWs compatible with the E2L framework, denoted as E2GW. The E2GW introduces the E2L executor, a lightweight and thin application virtualization layer for the execution of container-based micro-services and/or Big Data analytics operators on the received sensor data streams. Finally, at *the cloud layer*, each AS maintains the device registry, the GW registry and the table of device-gateway associations. These are in constant communication with the NS and JS so that the GW selection module can make Pareto-optimal decisions based on network performance, analytics execution performance, energy efficiency, security, reliability and dependability. The key agreement module establishes a secure communication channel between the E2ED, the E2GW, and the AS. The E2L driver is responsible for the facilitation of the execution flow of Big Data operators on IoT data streams and/or the migration of container-based micro-services to the E2GWs. Finally, the QoS support module ensures that edge-based processing provides certain guarantees.

In the overall system that combines legacy and E2L elements, frames transmitted by an E2ED that are received by a E2GW can be processed at the edge layer before they are forwarded to the E2L driver residing at the AS. In Fig. 6.1 this execution and data flow is depicted using the blue data path. On the other hand, frames transmitted by an E2ED that are received by a legacy GW and/or frames transmitted by a legacy ED that are received by an E2GW will be directly forwarded to the NS (green data link), without any processing at the edge layer, as per the current LoRaWAN network standard. Since LoRaWAN frames may be received by multiple GWs, some of them might be processed at the edge, while duplicates may arrive unprocessed at the AS. Therefore correct distributed processing of multiple sensor data stream processing tasks may require specific QoS guarantees. It is up to E2L driver in coordination with the NS to ensure the correct processing of the data streams that respect different QoS policies, including *"at most once"*, *"at least once"* and *"exactly once"* notions to ensure appropriate data management in different situations. For more details on the QoS guarantees see Sec. 6.4.

Typically frames in LoRaWAN contain *data* of the collected or processed information of a physical phenomenon as observed by an IoT device. In the standard LoRaWAN network standard, the ED encrypts the data included in the payload of a frame using a common key established between the ED and the AS. Therefore in the current LoRaWAN standard, neither the GW nor the NS are capable of decrypting the data part of the payload of the frames received. In E2L, access to the data included in the payload of the frames is enabled by establishing a group key between the E2L driver included in the AS, the E2GW and the E2ED. The group key agreement protocol introduced in this paper relies on light-weight asymmetric encryption techniques that ensure the confidentiality of the exchanged data between E2ED, E2GW, and the E2L driver, while at the same time keeping at minimum the energy and memory consumption. For more details see Sec. 6.3.

In the following sections I look into the modules that make up E2L and how they operate throughout the lifecycle of the network, from the deployment of the gateways and devices to the actual operation and data processing.

6.1 Gateway and Device deployment

The deployment of an E2L GW (E2GW) in a LoRaWAN requires to follow the GW registration process defined in the current LoRaWAN network standard: the 64-bit globally unique identifier of the GW (*GatewayEUI*) is provided to the NS and it is also inserted in the configuration of the legacy packet forwarder module of the GW along with the hostname or IP address of the NS. After this standard step, the *GatewayEUI* is also inserted in the *GW registry* of E2L along with information regarding the processing and storage capabilities of the E2GW as well as the capabilities of the IP-level network in terms of, e.g., available bandwidth and latency. Naturally, the network level parameters are monitored continuously to reflect the current conditions.

Similarly, the deployment of an E2L device (E2ED) requires first to follow the OTAA join process flow defined in the current LoRaWAN network standard (Alg, 1). First, the 64-bit globally unique identifier of the device (*DevEUI*) along with the 64-bit globally unique identifier of the network the device is joining (*JoinEUI*) needs to be hardcoded in the E2ED firmware along with the necessary information to setup the end-to-end encryption between the E2ED and the NS and AS. Second, these information are stored in the JS that is overseeing the standard LoRaWAN join procedure. Third, these information are also inserted in the *device registry* of E2L. In addition, the AS may introduce application-level metrics for each ED. These metrics are also stored in the device registry.

6.2 Gateway and Device association

In the current LoRaWAN network standard, the GWs are acting as bridges that directly forward the packets received from the wireless medium to the NS through the IP-based network. Recall that the GWs are incapable of decrypting the payload of the frames received. Moreover, due to the license-free ad-hoc deployment model adopted, gateways may have overlapping areas of network coverage (a) resulting in an increase of network traffic at the network backbone as frames are relayed to the NS multiple times, (b) may cause unexpected frame collisions and duty-cycle exhaustion.

On the other hand in E2L the E2ED is associated with one E2GW that is enabled to process the data included in the payloads of the frames transmitted by the E2ED. The association of devices with GWs is carried out centrally based on the application specific needs. Such an approach has several benefits.

First, the radio-level statistics received by the NS from the GWs are used for optimizing data rates, airtime and energy consumption in the network for each device individually [4]. Moreover, a global view of the network-level and radio-level performance allows to reach Pareto-optimal assignments that (i) optimize power consumption of the device while ensuring that frames are still received at the associated E2GW; (ii) equalize the Time-on-Air of frames transmitted by the E2ED in each spreading factor's group; iii) balances the assignment of E2ED to E2GW and the use of spreading factors across multiple E2GW and iv) keep into account the channel capture [25].

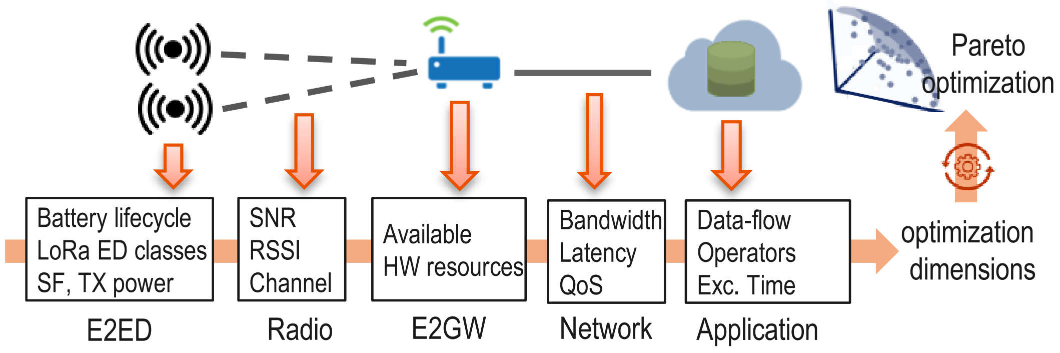


Figure 6.2. Example of a Pareto optimization strategy considered in E2L.

Second, IoT deployments consist of very diverse devices which require system support for a wide range of resource capabilities and security requirements. Some services may require low-latency processing of IoT data utilising specific hardware and software capabilities. Certain GWs may be small, battery operated, possibly mobile or deployed in outdoor environments potentially vulnerable to security threats while others may have access to specific heterogeneous hardware accelerators, such as Graphic Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs), thus achieving higher data processing throughput and energy-efficient execution. A centralized assignment allows a *hardware-conscious* optimization of the available resources in a holistic way [66].

Third, in cases when sensor data stream processing tasks are executed via a centralised Big Data processing framework, the operators of each task are organized in a data-flow graph that models the dependency between the operators. Operators may split data streams into windows of finite size based on criteria like time intervals, element counts, or sliding windows over which the computations are applied. In other cases join operators may combine frames arriving from the same E2ED or frames generated by multiple E2EDs based on common attributes. Ideally, each operator should be executed at the most suitable point across the CECC. Suitability is related to the available processing, storage, energy and communication resources and data access. Big Data frameworks have as an ultimate goal to scale out execution to increase performance by employing elaborate *scheduling* algorithms are used to identify the optimal execution plans that distribute data across the CECC. Information of such optimal execution plans will enable decisions such as where to execute each operator allowing intelligent resource selection and allocation [63].

The above dimensions provide a broad range of design choices for an optimization assignment of E2ED-E2GW. An example Pareto-optimal optimization flow diagram is illustrated in Fig. 6.2. The example indicates several dimensions taken into account along with their respective architectural collection points.

The table of device-gatewey associations provides a comprehensive view of the network topology by storing one entry for each device-gateway pair based on the radio-level statistics provided by the NS. In particular, during the operation of the network, as defined in the current LoRaWAN standard, when a GW receives a frame from an ED it carries out certain link-level budget/measurements, such as the Received Signal Strength Indicator (RSSI) and the Signal-to-Noise Ratio (SNR), and inserts them in the frame's header that is forwarded to the NS. The *table of device-gateway associations* of E2L is continuously listening for such updates from the NS given the latest radio-level statistics received from the GWs. Upon receiving updated radio-level statistics, it updates the corresponding entry related to the specific device-gateway pair.

Since the registries and the table of associations are continuously updated E2L can detect E2EDs and/or E2GWs that are no longer accessible, as well as changes in the topology due to the dynamic nature of the wireless medium, e.g., increase/decrease in noise producing interference, or due to changes of the device positions in case of passive or active mobility. Similarly, the IP-level network is continuously benchmarked in terms of available bandwidth and latency between E2GW-NS and E2GW-AS connections.

The GW selection module will associate each E2ED to an E2GW by consulting the device and gateway registries along with the *table of device-gateway associations* of E2L. Remark that the specific detail of an assignment strategy is presented in the Appendix.

As soon as the assignment is done, the selected E2GW takes on the responsibility of coordinating uplink and downlink data traffic between the E2ED and the AS as well as facilitating local processing and storage of frames following the processing and/or storage tasks assigned to the E2GW. As a result, direct communication via an IP link is established between the E2GW and the AS. VPN channels are preferred for this purpose, as they follow established best practices for providing secure communication between NS and GWs in LoRaWAN infrastructure. Remark that although the E2GWs are connected to only one NS, the above process does not exclude the possibility that each E2GW is connected to one or more AS.

The final step is to provide a secure communication channel between the E2ED and the E2GW so that the latter can have access to payloads of the frames transmitted by the E2GW. This is done using the group key agreement protocol presented in the following section.

6.3 Group key establishment

I here propose a group key agreement protocol that enables the creation of a shared session encryption key between the three involved actors. Remark that at this stage, the E2ED is already activated with the NS and the AS and the communication between the E2GW and the AS is secure. For simplicity I here assume that each E2ED is assigned to a single E2GW, however, the protocol can support any number of E2GWs.

Two shared session encryption keys are created between the E2ED, E2GW and the AS: the Edge Session Encryption Key (*EdgeSEncKey*) and the Edge Session Integrity Key (*EdgeSIntKey*). The former is used to enable secure encryption and decryption of the frame payload. The latter is used to check the integrity of the edge-specific frames. Here I propose the use of Elliptic Curve Cryptography (ECC), a public-key cryptography that uses elliptic curves over finite fields to create cryptographic keys [36, 40]. ECC offers a higher level of security with smaller key sizes compared to other public-key cryptography techniques such as RSA [2]. This ad-

Table 6.1. Proof of the E2L group key agreement protocol

E2ED	$EdgeSKey = Priv_{\mathsf{E2ED}} \times G_{\mathsf{AS-E2GW}} = Priv_{\mathsf{E2ED}} \times (Priv_{\mathsf{AS}} \times Pub_{\mathsf{E2GW}}) = Priv_{\mathsf{E2ED}} \times Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2GW}} \times \mathcal{P}$	(6.1)
E2GW	$EdgeSKey = Priv_{\mathsf{E2GW}} \times G_{\mathsf{AS-E2ED}} = Priv_{\mathsf{E2GW}} \times (Priv_{\mathsf{AS}} \times Pub_{\mathsf{E2ED}}) = Priv_{\mathsf{E2GW}} \times Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2ED}} \times \mathcal{P}$	(6.2)
AS	$EdgeSKey = Priv_{\mathsf{AS}} \times G_{\mathsf{E2GW-E2ED}} = Priv_{\mathsf{AS}} \times (Priv_{\mathsf{E2GW}} \times Pub_{\mathsf{E2ED}}) = Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2GW}} \times Priv_{\mathsf{E2ED}} \times \mathcal{P}$	(6.3)
KEY	$Priv_{\mathsf{E2ED}} \times Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2GW}} \times \mathcal{P} = Priv_{\mathsf{E2GW}} \times Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2ED}} \times \mathcal{P} = Priv_{\mathsf{AS}} \times Priv_{\mathsf{E2GW}} \times Priv_{\mathsf{E2ED}} \times \mathcal{P}$	(6.4)

vantage enables the use of asymmetric encryption in resource-constrained scenarios, such as LoRaWAN EDs, with minimal energy consumption and memory consumption [41].

This approach helps to mitigate the computational overhead, particularly on the device, which is introduced by asymmetric cryptography compared to symmetric cryptography. Moreover, ECC, and hence asymmetric cryptography, is exploited only to compute the secret shared between the three parties. Once the two edge keys are computed, AES-128bit encryption shall be used complying with the actual LoRaWAN specification.

The protocol is triggered by the E2ED, however, to optimize network utilization and reduce latency, the E2GW should share its ephemeral public ECC keys with the AS. Remark that the procedure is enforced on classical LoRaWAN application-specific frames where the payload is encrypted with *AppSEncKey* and integrity is performed with *FNwkSIntKey*. The E2GW and any other GWs in the radio range of the E2ED will forward the frame to the NS. The NS checks the integrity and, if successful, it sends the frame to the AS, discarding possible duplicates of the same frame. To simplify the presentation of the algorithm Alg. 4 shows only the relevant steps for my proposal, not including steps of the LoRaWAN standard frame exchange.

Tab 6.1 shows the proof that the three actors compute the same key. The E2ED (Eq. 6.1), the E2GW (Eq. 6.2), and the AS (Eq. 6.3) compute a value that is the multiplication of the three private ECC key and the same point P of the Elliptic Curve. Since the private ECC keys are scalar, the three actors have computed the same secret as shown in Eq. 6.4.

The *EdgeSKey* shall be used to derive the *EdgeSEncKey* and the *EdgeSIntKey* using a 128-bit hashing function as shown in Alg. 4. The hashing function allows for greater flexibility in choosing the size of the elliptic curve used in the protocol, which impacts the size of the secret. This ensures that the security of the protocol is not compromised due to a constraint on the size of the curve.

The AS shall securely share only the *EdgeSIntKey* with the NS, to check the frame integrity in case the E2GW is not available, or the frame coming from legacy GWs.

Finally, the complete network lifecycle from LoRaWAN device activation to E2L Cloud-Edge-Device coordination establishment is depicted in Fig. 6.3. The figure also includes the E2GW selection process discussed in the previous section.

It is worth noting that the E2GW remains active for legacy EDs, thus frames

Algorithm 4 E2L Group Key Agreement

```
Require: ECC Curve with point $\mathcal{P}$ common to every actor.
  ACTOR: E2ED
  upon event $< E2LGKA,\ Init >$ do
      $Priv_{\mathsf{E2ED}} = random\_bytes()$
      $Pub_{\mathsf{E2ED}} = Priv_{\mathsf{E2ED}} \times \mathcal{P}$
      $EdgeSEncKey = \perp$
      $EdgeSIntKey = \perp$
      trigger $< Receive,\ EdgeJoinRequest|Pub_{\mathsf{E2ED}},\ \mathsf{AS} >$
  end upon event
  upon event $< Receive,\ EdgeJoinAccept|G_{\mathsf{AS-E2GW}} >$ do
      $EdgeSKey = Priv_{\mathsf{E2ED}} \times G_{\mathsf{AS-E2GW}}$
      $EdgeSEncKey = hash_{128}(0x01|EdgeSKey)$
      $EdgeSIntKey = hash_{128}(0x02|EdgeSKey)$
  end upon event
  ACTOR: E2GW
  upon event $< E2LGKA,\ Init >$ do
      $Priv_{\mathsf{E2GW}} = random\_bytes()$
      $Pub_{\mathsf{E2GW}} = Priv_{\mathsf{E2GW}} \times \mathcal{P}$
      $EdgeSEncKey = \perp$
      $EdgeSIntKey = \perp$
      trigger $< Receive,\ PubInfo|Pub_{\mathsf{E2GW}},\ \mathsf{AS} >$
  end upon event
  upon event $< Receive,\ PubInfo|(G_{\mathsf{AS-E2ED}}, Pub_{\mathsf{E2ED}}) >$ do
      $G_{\mathsf{E2GW-E2ED}} = Priv_{\mathsf{E2GW}} \times Pub_{\mathsf{E2ED}}$
      trigger $< Receive,\ PubInfo|G_{\mathsf{E2GW-E2ED}},\ \mathsf{AS} >$
      $EdgeSKey = Priv_{\mathsf{E2GW}} \times G_{\mathsf{AS-E2ED}}$
      $EdgeSEncKey = hash_{128}(0x01|EdgeSKey)$
      $EdgeSIntKey = hash_{128}(0x02|EdgeSKey)$
  end upon event
  ACTOR: AS
  upon event $< E2LGKA,\ Init >$ do
      $Priv_{\mathsf{AS}} = random\_bytes()$
      $Pub_{\mathsf{AS}} = Priv_{\mathsf{AS}} \times \mathcal{P}$
      $EdgeSEncKey = \perp$
      $EdgeSIntKey = \perp$
  end upon event
  upon event $< Receive,\ PubInfo|Pub_{\mathsf{E2GW}} >$ do
      $G_{\mathsf{AS-E2GW}} = Priv_{\mathsf{AS}} \times Pub_{\mathsf{E2GW}}$
  end upon event
  upon event $< Receive,\ EdgeJoinRequest|Pub_{\mathsf{E2ED}} >$ do
      trigger $< Receive,\ EdgeJoinAccept|G_{\mathsf{AS-E2GW}},\ \mathsf{E2ED} >$
      $G_{\mathsf{AS-E2ED}} = Priv_{\mathsf{AS}} \times Pub_{\mathsf{E2ED}}$
      trigger $< Receive,\ PubInfo|(G_{\mathsf{AS-E2ED}}, Pub_{\mathsf{E2ED}}),\ \mathsf{E2GW} >$
  end upon event
  upon event $< Receive,\ PubInfo|G_{\mathsf{E2GW-E2ED}} >$ do
      $EdgeSKey = Priv_{\mathsf{AS}} \times G_{\mathsf{E2GW-E2ED}}$
      $EdgeSEncKey = hash_{128}(0x01|EdgeSKey)$
      $EdgeSIntKey = hash_{128}(0x02|EdgeSKey)$
      trigger $< Receive,\ IntKey|EdgeSIntKey,\ \mathsf{NS} >$
  end upon event
  ACTOR: NS
  upon event $< E2LGKA,\ Init >$ do
      $EdgeSIntKey = \perp$
  end upon event
  upon event $< Receive,\ IntKey|EdgeSIntKey >$ do
      $EdgeSIntKey = EdgeSIntKey$
  end upon event
```

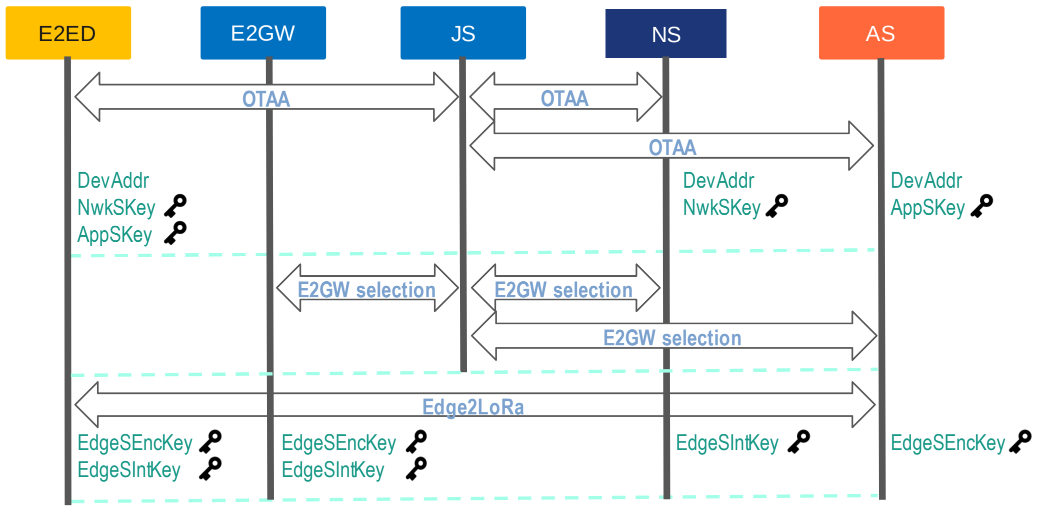


Figure 6.3. LoRaWAN device activation and E2L coordination procedures to enable edge processing while maintaining backwards compatibility.

received from legacy EDs are still forwarded to the NS following the LoRaWAN standard.

6.4 Multiple deliveries and QoS support

In the current LoRaWAN standard the NS is responsible for identifying and removing frames received multiple times, selecting those with the best signal quality to forward to the AS. On the other hand, in E2L deployment, multiple alternatives of frame routing, processing and storage may happen concurrently either at the edge or at the cloud. Frames consumed at an associated E2GW are not forwarded to the NS but instead the AS is notified directly about the outcome of the edge-based processing and/or storage. At the same time, frames received from legacy GWs coexisting in the infrastructures arrive at the AS via the NS. It is therefore crucial to make sure that duplicate processing and/or storage of data is avoided. It is the role of the E2L driver module to coordinate the flow of data in the CECC.

In the following paragraphs I examine the six possible scenarios of up-link frame transmissions. I consider the first three scenarios simple, in the sense that processing and/or storage of frames takes place exclusively a single point in the CECC, e.g., either at the cloud or at the network edge. The other three scenarios constitute more complex cases where some frames may traverse the network by following alternative paths that involve legacy and E2L elements. In these latter scenarios it is crucial to ensure that the frames are not processed and/or stored multiple times throughout the CECC when it is required to respect *"at-most-once" or "exactly-once" QoS guarantees.*

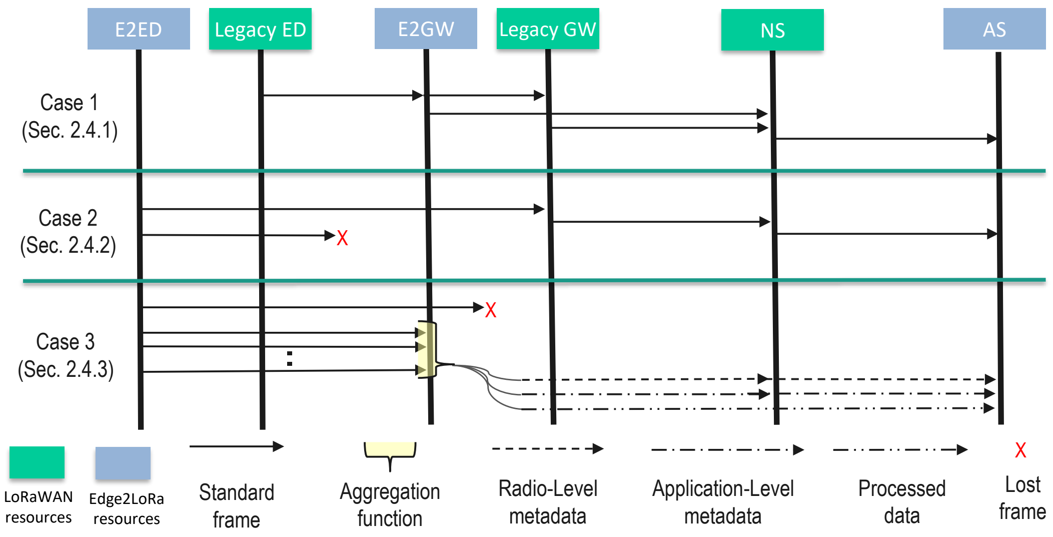


Figure 6.4. Flow of frames in simple scenarios where legacy and E2L elements co-exist in the IoT deployment. In cases 1 and 2 the network operates in full compliance with the LoRaWAN network specifications without being able to execute any processing and/or storage task in the CECC. Case 3 depicts the case where only E2L elements are active and processing and storage tasks are executed at the network edge.

Case 1: Frames transmitted by a legacy ED are received by both legacy GWs and E2GWs

In the first case, frames transmitted by a legacy ED are received by both a legacy GW and an E2GW. In this case the E2GW operates in full compliance with the LoRaWAN network specifications. In other words, the frames along with all the radio-level information collected from all GWs are forwarded to the NS. The NS uniquely selects those with the best signal quality and delivers them to the AS while discarding any duplicate frames implementing the existing LoRaWAN protocols. Fig. 6.4 depicts this standard flow.

Case 2: Frames transmitted by a E2ED are received only by legacy GWs

The second scenario illustrates the case where the frames transmitted by an E2ED are received only by legacy GWs and thus no edge processing can take place. Like in the previous case, the frames are forwarded to the NS which will deliver them to the AS following the standard flow described in the LoRaWAN network specifications. Once again no processing and/or storage of frames can take place at the network edges. The flow is depicted in Fig. 6.4.

Case 3: Frames transmitted by an E2ED are received only by an E2GW

The third scenario considers one more simple case since the frames transmitted by an E2ED are received only by the associated E2GW without the activation of any legacy elements due to continuous failures. Such failures may involve permanent failures on the hardware equipment or continous wireless interference blocking proper reception of the frames. This results in frames delivered to the AS via a single path. In this case the processing and storage of frames may be carried out exclusively at the

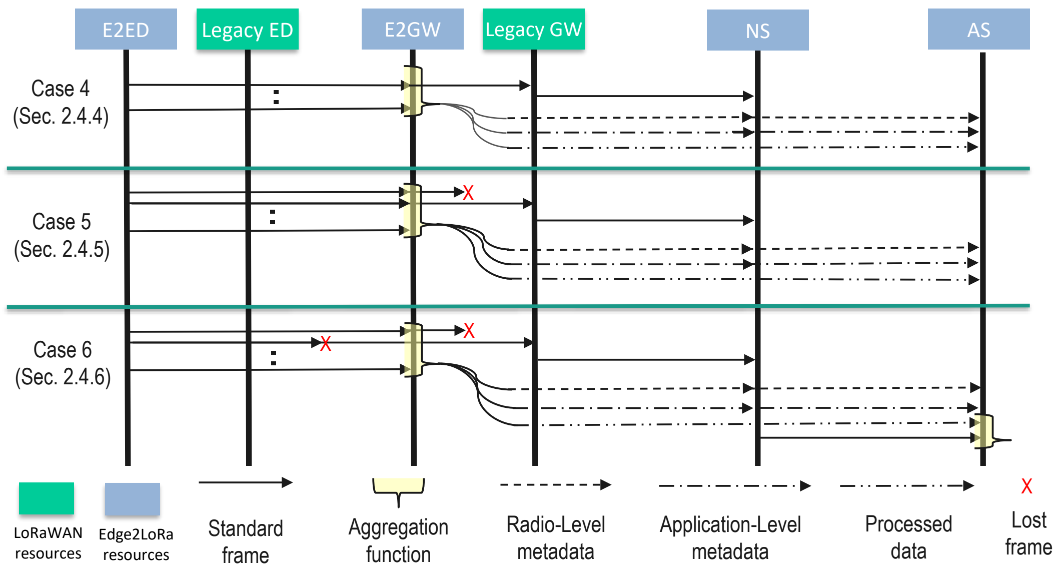


Figure 6.5. E2L frame flow in a mixed scenario through both legacy and E2L elements. The figure also illustrates the flow of metadata involving the NS and the AS, taking into account modifications to the NS and the enabling of *"exactly-once"* QoS.

associated E2GW. Fig. 6.4 depicts how the processing and/or storage of the data included in the payloads of the received frames is carried out at the network edge. The outcome of the edge-based processing and/or storage is forwarded directly to the E2L driver module residing at the AS. At the same time, the collected radio-level information along with application-level metadata are forwarded to the NS so that the device and gateway registries are properly updated to reflect the current network conditions.

To be noted that the NS involvement is not always necessary for the processing of the application-level metadata. For instance, when considering backward compatibility with legacy NS the metadata only needs to reach the AS. The system will maintain its functionality because radio-level metadata will still be forwarded to the NS.

Case 4: Frames transmitted by an E2ED are received by both a legacy GW and by an E2GW

In the fourth scenario up-link frames arrive at the AS through multiple paths as shown in Fig. 6.5 without experiencing any transient of permanent failures. The frames received by legacy GWs are forwarded to the NS and are eventually delivered to the AS following the standard flow described in the LoRaWAN network specifications, similar to case 1. At the same time, the frames received by the associated E2GW may be processed and/or stored at the network edge, in a way similar to case 3. Whenever it is required to support *"at-most-once" or "exactly-once" QoS guarantees* it is necessary to ensure that the frames are not processed and/or stored twice throughout the CECC. However I consider this to be a simple case since the scenario naively assumes that no transmission failures occur and that frames are always delivered through both paths. Under these assumptions, the necessary mechanisms are provided for the AS to properly identify and discard duplicate frames without requiring any modification to the legacy NS.

In order to enable QoS by detecting processed frames at different levels, the E2L protocol works to generate metadata used to notify processed frames together to the aggregation result. The E2L driver module residing in the AS implements a duplicate detection filter (DDF) for identifying whether a given frame has previously appeared in a stream of data. The DDF is used to identify with no errors, duplicate frames in constant time [26]. The DDF relies on metadata produced by the E2GW upon receiving a frame that is about to be processed and transmitted to the AS. As shown in Fig. 6.5, there are two types of metadata generated: radio-level and application-level. The metadata are used to ensure that the device registries and table of associations are properly updated, like in case 3. At the same time application-level are also used to register the frame ID in the DDF maintained by the AS. In this way the AS will be notified about a frame being properly received at the E2GW and discard the one also delivered by the NS. After transmitting the metadata, the E2GW will process the frame as per the assigned processing and/or storage tasks. The outcome of the edge-based processing and/or storage may be transmitted to the AS at a later stage, as soon as the edge-based tasks are completed.

Case 5: All frames transmitted by an E2ED are received by an E2GW while some are also received by legacy GWs

The fifth scenario is similar to the fourth one, with the difference that now I assume that some frames are not received by a legacy GW. That is, I now assume a more realistic case where some transient failures occur either at the hardware level or during wireless transmissions however involving only the legacy GW. On the other hand no failures occur during the reception of the frames by the associated E2GW as depicted in Fig. 6.5.

Like in the fourth scenario, the DDF maintained by the E2L driver module is enough to guarantee that the AS will be in a position to identify and discard the frames received by the legacy GW and arriving through the NS.

Case 6: Some frames transmitted by an E2ED are received only by a E2GW or only by an legacy GW

The sixth scenario represents the most realistic case where due to wireless transmission failures and/or transient failures at the hardware of both legacy and CECC-native hardware, some up-link frames arrive either through both paths or via one of the two paths as shown in Fig. 6.5. The complexity of this case is more evident considering that a processing and/or storage task may be composed by one or a combination of more operators that apply i) *one-to-one and one-to-many transformations* of the received frames that are communicated directly to the AS and/or ii) *many-to-one transformations* potentially aggregating data received from multiple frames where results are communicated after all frames received within a window are processed and/or stored.

When a processing and/or storage task relies on transformations that require the aggregation of multiple frames arriving over a window of time, avoiding inaccurate aggregations due to missing frames or frames being computed multiple times is needed. This can be achieve this by assigning a timeout on the processing and/or storage of each single frame delivered to the AS via the NS. The duration of the timeout is appropriately set by the E2L driver module in a way such that it will allow the E2GW to complete the processing of the operator. The E2L driver module decides on the length of the timeout based on a series of factors taking into consideration the type and combination of the operators used along with the data-flow graph that models the dependency between the operators and the hardware-level, network-level and application-level operation parameters of the E2GW. The E2L driver module relies on the up-to-date information stored in the device and gateway registries. For example, for filter operators the timeout can be set to zero; for map operators that carry out one-to-one or one-to-many transformations the timeout can be set in relation to the latency of the E2GW-AS link and the computational resources of the E2GW; for window operators, the timeout needs to also consider the size of the window.

Depending on the actual task and given the presence of frames being delivered via legacy elements, the AS may have to repeat the processing and/or storage tasks carried out by the E2GW. This will require the transmission of the raw data included in the frames received only by the E2GW to the AS so that the transformations can be accurately executed respecting the required QoS policies. *“At-most-once”* is the cheapest with the least implementation overhead and highest performance because it can be done in a fire-and-forget fashion without keeping the state in the E2GW or at the E2ED. Guaranteeing *“at-least-once”* requires multiple transmission attempts in order to counter transport losses which means keeping the state at the device and utilising the LoRaWAN acknowledgement mechanism. The *“exactly-once”* is the most expensive and has consequently the worst performance because, in addition to *“at-least-once”* that relies on the LoRaWAN acknowledgement mechanism, it requires the state to be kept at the NS in order to filter duplicate deliveries. Additional implementation details are provided in Sec. 7.1.3.

Chapter 7

Implementation Details & Performance Evaluation

7.1 Implementation Details

The implementation of the E2L services depicted in Fig. 6.1 is available as open-source project[1] to freely download and execute in public or private LoRaWAN deployments. To be noted that a previous version of the code has been demonstrated at MobiCom2023 [43]. This demonstration involved the use of five physical end devices (`Heltec Cubecell HTCC-AB01`[2] & `HTCC-AB02`[3]), two of which were implementing the legacy LoRaWAN components while three included also the E2L extensions. The demonstration also included a NS based on the Things Stack[4], compatible with the LoRaWAN 1.0.4 specification. The demonstration aimed to validate the system's functionality within private deployments in a scaled-down scenario.

Tab. 7.1 summarizes the complete implementation details for the different components deployed through the three layers of the architecture.

Table 7.1. E2L module implementation details.

Component	**Programming Language**	**Main Libraries**
E2ED [43]	Arduino and RIOT-OS	Crypto (AES), micro-ecc (ECC)
E2GW	Rust	tonic (RPC), p256 (ECC), lorawan-encoding (Packet parsing, decryption, encryption)
AS	Python	Eclipse Paho (MQTT), gRPC (RPC), pycryptodome (Crypto)
ED emulator	NodeJS	lora-packet (Packet parsing, encryption)

[1]https://github.com/Edge2LoRa
[2]https://heltec.org/project/htcc-ab01-v2/
[3]https://heltec.org/project/htcc-ab02/
[4]https://github.com/TheThingsNetwork/lorawan-stack

7.1.1 Far-Edge layer

At the far-edge layer I implement the E2L Key Agreement module based on the `secp256r1`[5] elliptic curve. I selected this curve as it employs 256-bit private keys and 512-bit public keys. Remark that the public keys can be compressed to 33 Bytes, making them suitable for transmission over a single LoRaWAN frame. The computed secret key is of the same size as the private key. Subsequently, it undergoes two SHA256 hashing processes, each concatenating the secret with a byte, differing in the two rounds. The first 16 bytes of the result are then used to generate the final *EdgeSEncKey* and *EdgeSIntKey*.

I provide a hardware-agnostic implementation of the E2L Key Agreement module using the *RIOT Operating System* [55] since it supports many different architectures for 8bit, 16bit, 32bit and 64bit processors, provides a simple process manager with support for multi-threading, provides a generic network stack and also power management [56]. I also use *Arduino* to provide a hardware-specific implementation of the module for the `Heltec Cubecell HTCC-AB01 and HTCC-AB02` as they provide a native implementation of LoRaWAN 1.0.4[6]. Both RIOT OS and Arduino incorporate the `micro-ecc` library [34] that implement ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors. The implementation of the 128-bits AES encryption was based on the `crypto` module provided by *RIOT OS* [54] and by *Arduino*[7]. These cryptographic functions can be used within security protocols at the system level by providing seamless crypto support across software and hardware components [27]. A detailed evaluation of the energy consumption overhead incurred by the E2L Key Agreement module due to the use of the ECC on different common IoT chips is available in [41]. Moreover, this implementation can be easily replicated on other hardware/software platforms. This is because the LoRa driver, which is a key component in the system, is not modified. This means that the changes are largely agnostic to the underlying hardware and software, making them versatile and easy to implement across different systems.

7.1.2 Edge layer

At the Edge layer I implement the E2L executor module that is deployed at the E2GW. The executor is interconnected with the standard *Packet Forwarder* of the Semtech[8] through a light-weight proxy component implemented in Rust. The proxy intercepts traffic from the *Packet Forwarder* and redirects it to the E2L executor module after it has been decrypted by the E2L Key Agreement module executed within the E2GW. The LoRaWAN packet parsing, decryption and encryption are implemented with the *lorawan-encoding* crate[9], while the Elliptic Curve cryptographic primitives are implemented using the *p256* crate[10]. The communication over the RPC Protocol is implemented using the *tonic* create[11], which is a production-ready

[5]https://neuromancer.sk/std/secg/secp256r1
[6]https://github.com/HelTecAutomation/CubeCell-Arduino
[7]https://rweather.github.io/arduinolibs/crypto.html
[8]https://github.com/Lora-net
[9]https://docs.rs/lorawan-encoding/latest/lorawan_encoding
[10]https://docs.rs/p256/latest/p256/
[11]https://docs.rs/tonic/latest/tonic/

implementation of *gRPC*[12] for Rust.

7.1.3 Cloud layer

For the implementation of the E2L extensions residing at the cloud layer I use the Thing Stack. It is implemented using Python3 and it interfaces with Thing Stack using the MQTT integration that the latter offers. Using the MQTT integration the E2L components residing in the AS can receive the uplink LoRaWAN packets and schedule downlink ones. The MQTT interface is implemented with the Eclipse Paho library[13]. The communication between the AS and the E2GW exploit the RPC Protocol, implemented using the *gRPC Python3 SDK*[14]. All the cryptographic primitives are implemented using the *pycryptodome* library[15] which offers support for both symmetric and asymmetric cryptography and for hashing functions.

For the implementation of the different QoS policies, as mentioned in Sec. 6.4, modifications to the Network Server (NS) are not always required. This is particularly true when considering backward compatibility with legacy NS systems. The provision of the *"at-least-once"* and *"at-most-once"* QoS guarantees depends on the version of the LoRaWAN network specifications implemented by the NS. The main difference between the two versions of the specification lies in the check of frame integrity:

- **LoRaWAN 1.0.4**: No integrity check is computed by the NS. The NS checks the FCnt of the E2L frame received, and if the check passes, it forwards it to the AS.
- **LoRaWAN 1.1.0**: The NS checks the frame integrity using the FNwkSIntKey. The MIC, used for this integrity check, is computed using the new EdgeSIntKey. Consequently, the integrity check of the NS will fail, resulting in the frame being dropped. Given that the NS will reject all E2L frames, certain precautions must be taken by the E2ED or the E2GW to ensure the NS can effectively manage network parameters and downlink scheduling. Two potential solutions are proposed:

  (a) The E2ED periodically sends a legacy frame to the NS, allowing it to update the FCnt and schedule downlinks accordingly. While this solution does not compromise security, it mandates the E2ED to intermittently transmit a legacy frame, impacting the benefits of Edge Computing.

  (b) Using the secure channel established through the group key agreement, the E2ED shares the FNwkSIntKey with the E2GW. Subsequently, the E2GW can periodically generate simulated legacy frames containing the updated counters and an aggregation of the network metrics. These frames are then transmitted to the NS to facilitate updates. Although the latter solution does not compromise the advantages of Edge Computing, it introduces a potential security impact.

[12]https://grpc.io/
[13]https://eclipse.dev/paho/
[14]https://grpc.github.io/grpc/python/
[15]https://www.pycryptodome.org/

The choice between these solutions involves a trade-off between security and uninterrupted edge-level operations.

Given these considerations, E2L supports the *"at least once"* QoS guarantee in the case of LoRaWAN 1.0.4, while it supports the *"at most once"* QoS guarantee in the case of LoRaWAN 1.1.0, both fully supports the existing deployed structures, ensuring that the system's operational continuity is maintained. However, to support the *"exactly-once"* QoS guarantee the NS needs to support the DDF discussed above requiring the coordination between E2GW, the NS and the AS in combination with advance mechanisms for data buffering and retransmissions [35]. Table 7.2 summarizes the QoS guarantees that E2L is capable of supporting, detailing all possible combinations of E2L and legacy components.

Table 7.2. QoS support for different configuration scenarios.

Network Server	**LoRaWAN Specification**	**QoS**	**Backward compatibility**	**E2L**
Legacy	1.0.4	*at least once*	✓	
Legacy	1.1.0	*at most once*	✓	5✓
E2L support	1.0.4	*exactly once*	✗	✓
E2L support	1.1.1	*exactly once*	✗	✓

7.1.4 Hybrid testbed for Large scale experimentation

In contrast to the small-scale evaluation presented in [41,43] here I wish to look into large-scale LoRaWAN deployments. Moreover, in contrast to the previous evaluations, I am interested to evaluate the backwards compatibility of the provided E2L extensions with the LoRaWAN network standard. It is therefore crucial to provide an environment that allows to experiment using a public infrastructure rather than a privately deployed one. For this reason, I developed a detailed emulator module for the End Devices, a tool designed to reproduce LoRaWAN frames received in a real-world network environment.

The tool is a comprehensive emulator software of LoRaWAN EDs that allows the creation of extensive testing environments, encompassing thousands of EDs on a controlled network setup. The emulated EDs adhere to LoRaWAN v1.0.4 and v1.1 specifications and support the OTAA activation method. The modular architecture allows different types of LoRaWAN EDs (Class A, B, and C) to be modelled, each with the ability to send different payloads as defined by the configuration scripts. Emulated LoRaWAN EDs transmit their PhyPayload frames via UDP to the GWs, which encapsulate them into either Semtech UDP frame Forwarder (GWMP) messages for delivery to the NS, in case of legacy frame, or to the assigned E2GW executor according to the logic implemented in the E2L driver.

The tool provides separate configuration parameters to control and fine-tune the activity of the legacy EDs and the E2ED. Parameters such as the EDs source rate and the message payload can be configured for each device individually. Moreover, the emulator can be connected to a real dataset of sensor values so that it can encapsulate the sensor values in the payloads of the transmitted frames, achieving in this way a higher fidelity. This feature allows for more accurate and realistic testing and evaluation of the system's performance under conditions that closely mimic real-world scenarios.

At the network level, real LoRaWAN network traffic is reproduced by incorporating the detailed frames loss model introduced in [25]. The interference model considers EDs deployed in geographic areas that are covered by multiple GWs. The coverage of each GW is represented by a circular radius. The model considers uniformly deploying EDs in areas where multiple GWs operate with possible overlapping coverage areas. The emulator is fully aligned with the methodology described in the referenced paper [25] and is modelled by the following formula, reported as formula [25](13).

$$\begin{aligned} S_c(G_{sf}) = 2\pi \int_0^{R/\alpha} \delta_{sf} e^{-2\frac{\alpha^2 r^2}{R^2} G_{sf}} r \cdot dr + \\ +\delta_{sf}(\pi R^2 - \pi R^2/\alpha^2) e^{-2 \cdot G_{sf}} \end{aligned} \tag{7.1}$$

The above formula (Eq. 7.1) generalises the medium access control in LoRaWAN that follows Aloha scheme in a scenario where the EDs are uniformly distributed. The model is specifically formulated for a circular coverage area and the throughput is calculated in the presence of a channel capture effect. In the formula $\delta_{sf} = G_{sf}/(\pi R^2)$ is the density of traffic load offered to a SF=sf and R is the cell radius. Due to the scenario configuration, a target ED is actually competing with a fraction of devices generating the total load G_{sf}. Indeed, neglecting the effect of random fading and assuming an attenuation law of type $r^{-\eta}$, all the interfering nodes at a distance higher than $\alpha \cdot r$, with $\alpha = 10^{SIR/10\eta} > 1$, do not prevent the correct demodulation of the signal from the target ED. Here, SIR represents the Signal to Interference Ratio and η is the propagation coefficient. The smaller the α coefficient, the lower the number of competing EDs. Therefore, the throughput can be obtained by $S_c(G_{sf})/G_{sf}$. Finally, the model supports the presence of multiple GWs with overlapping areas, where M is the number of GWs deployed in the coverage area and $S_c(G)$ the throughput perceived under load G, the total capacity can be by approximated by $M \sum_{sf} S_c(G_{sf}/M)$.

Fig. 7.1 illustrates the developed testbed setup for the purposes of the large-scale performance evaluation. The ED layer and part of the GW layer are replaced by the emulator that communicates with software components executed in real-hardware elements. In the figure, light-blue blocks represent emulated components, while yellow blocks represent real deployment hardware elements. To validate the backward compatibility of the proposed architecture, utilizing the TheThingsNetwork public LoRaWAN infrastructure.

7.2 Performance Evaluation

In this section, I present a comprehensive evaluation in terms of the network performance improvements and the security guarantees provided by the E2L extensions

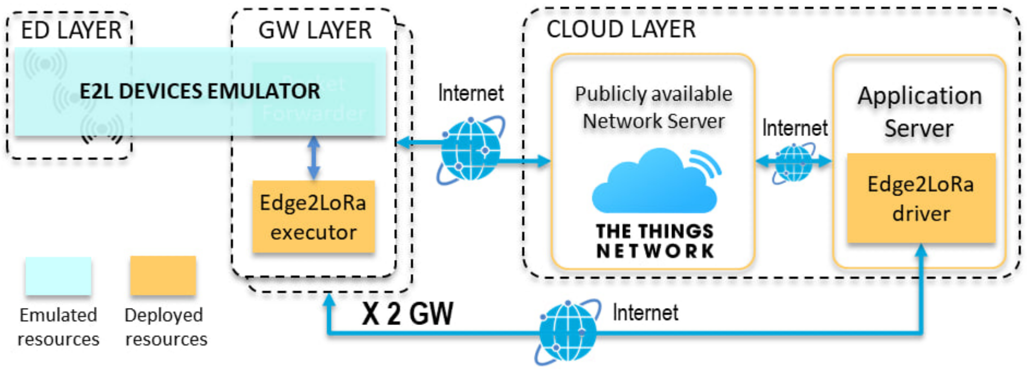


Figure 7.1. Overview of the testbed that combines emulated elements with real-hardware communicating with the TheThingsNetwork, a public LoRaWAN infrastructure. The diagram is color-coded to distinguish between the emulated components (light-blue) and the actual hardware elements (yellow).

to the current LoRaWAN network specifications. The results presented here provide insights on the advantages of my design choices and how they can be used to improve the operation of large-scale LoRaWAN deployments.

Several sectors can benefit from the capabilities of E2L. For instance, in Water Metering and Flow Monitoring, edge processing improves computing latency. The E2L solution enables faster data processing by leveraging the substantial computing capacity of network servers, thus enhancing the efficiency of water network monitoring [6]. Moreover, privacy preservation can benefit from edge processing as it reduces the transmission of sensitive data to cloud servers, thereby mitigating privacy risks and potential misuse or theft of user data. In the realm of water management, edge computing allows for the implementation of solutions to detect user anomaly patterns and identify water leakage, enhancing the overall security and efficiency of the water network [24].

Smart Buildings is another vertical application that can benefit from the E2L solution that performs data pre-processing aggregation at IoT gateways, reducing network bandwidth consumption and addressing issues such as transmission delay and packet loss [3]. Furthermore, traffic flow control is enabled by migrating data processing and aggregation tasks to the edge, optimizing traffic flow, minimizing bandwidth requirements for end users, while maintaining data quality. Real-time local analysis is made possible by E2L, enabling local data traffic analysis and real-time notifications to a local entity, facilitating efficient monitoring and management of smart building systems.

In the Smart Industry vertical, the enhanced security of the E2L approach implements security measures to protect against attacks and data manipulation, ensuring the robustness of edge nodes, servers, and networks, thereby safeguarding critical industrial infrastructure. Finally, in Agricultural Applications, the solution takes into account power resource optimization [49]. It improves power resources and battery capacity in agricultural settings by incorporating a flexible task offloading scheme that considers the power resources of each device.

7.2.1 Network Performance

In terms of network performance, first, I evaluate the performance of a large-scale LoRaWAN deployment that follows the standard network specification to serve as baseline. Then I introduce the E2L extensions and measure the improvements achieved under various traffic loads. My goal is to highlight the benefits of the E2L proposal and also demonstrate the backward compatibility of all the E2L extensions when operated in a LoRaWAN infrastructure that is fully compliant with the current network standard.

Large-scale Deployment Scenario

In this chapter, I focus on massive IoT scenarios that encompass dense deployments of a large number of IoT devices that require massive connectivity to efficiently transmit streams of sensor data to cloud services [16,23,33]. I look into deployments that are expected to reach device densities ranging from 1k to 10M devices per square kilometer [19]. Apart from the device density, other factors that need to be taken into consideration are the device duty-cycle, the sensor sampling rate and the message generation frequency. Furthermore, the density of the gateways and the resulting overlap in the network coverage needs to be taken into account since they affect the overall performance of the network [12].

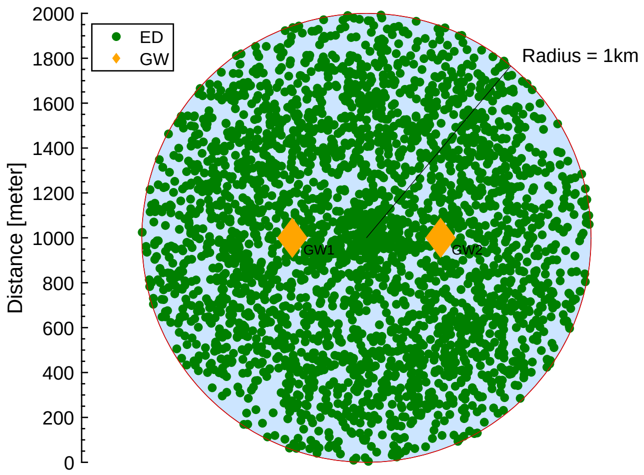


Figure 7.2. A large-scale IoT network scenario comprised from 3000 EDs and 2 GWs uniformly deployed in a circular area of 1 km of radius.

I design a large-scale IoT network scenario where 3000 devices are uniformly deployed in a circular area of 1 km of radius, producing a total of 500 frames of 24 bytes each and using a transmission rate of 1 frame every 3 sec. The LoRaWAN data rate is set to $DR = 5$ (spreading factor set to $SF = 7$ and bandwidth $125KHz$). Moreover, two GWs are deployed equally spaced at 0.15 km from the centre of the circular area. The devices are progressively activated with an interval of 0.1 sec, resulting in a transitory period of approximately 5 minutes before all the devices become active. The selected scenario produces a density of approximately 950 devices per square kilometre and generates 1000 frames per second. According to the formula (13) of [25], each GW perceives on average a network Frame Delivery Ratio (FDR) of 31%. The complete list of parameters used for the evaluation scenario is summarized in Table 7.3. For a graphical representation of the deployment scenario refer to Fig. 7.2.

In the experiments presented here, the devices are configured to act as legacy ED or as E2ED. Similarly, the gateways are either set to legacy mode or E2GW. In the experiments that involve E2ED and E2GW, the E2L driver is configured to assign half of the E2ED to one E2GW and the other half to the other E2GW. I use this simple approach of partitioning the E2ED to E2GW so that I can better understand the improvements in comparison with the performance of the legacy LoRaWAN components. The performance of different Pareto-optimal strategies, as those discussed in Sec. 6.2, are left as future work.

In terms of the high-level application, I introduce only one application where the edge-computing task deployed to the E2GW comprises of a series of one-to-one and many-to-one operators applied on a device-based window. I configure the window size to 30 seconds. As a result, I expect that for every 10 frames received from each E2ED the task will produce a single message containing the aggregated value of the 10 measurements included in the payloads of the original frames. Once again, other kinds of edge-computing tasks that reflect specific use-case scenario are left for future work. Since in LoRaWAN each device can belong to only one application, the gateways will execute the group key agreement module for each device only once. Therefore, the scalability of E2L is independent of the number of applications but depends only on the number of devices. Moreover, since each device is assigned to only one gateway, there is no repetition of the execution of the group key establishment module across multiple number of gateways.

All experiments are carried out for a total duration of 30 minutes. Repeating each experiment multiple times until the certainty is above 95% to ensure the repeatability of the results.

In terms of hardware used throughout the experiments, for the gateways and E2GW I use Raspberry Pi 4 Model B units, each equipped with a Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz and 4GB of RAM running Debian GNU/Linux 11. The AS is deployed on an Intel NUC, supplied with an Intel i5-6260U CPU @ 1.80GHz and 8GB of RAM running Ubuntu 22.04.3 LTS. For the experiments that rely on the TheThingsNetwork public infrastructure, the NS is provided by the TheThingsNetwork. On the other hand, when I deploy a NS within the laboratory, I use an Intel NUC, similar to the one used to host the AS, that is an Intel i5-6260U CPU @ 1.80GHz and 8GB of RAM running Ubuntu 22.04.3 LTS.

Evaluation in Different Scenarios

I have selected four metrics to evaluate the performance of the E2L system: the total number of frames transmitted over the IP-based network, the number of frames delivered solely by one gateway, and the computational resources utilized by the host machine, specifically the percentage of CPU usage and the amount of memory usage measured in Gigabytes (GB). The first two metrics define the volume of traffic arriving from each delivery route and help us identify the data reduction due to edge

Table 7.3. LoRaWAN scenario simulation parameters for the realistic large-scale scenario.

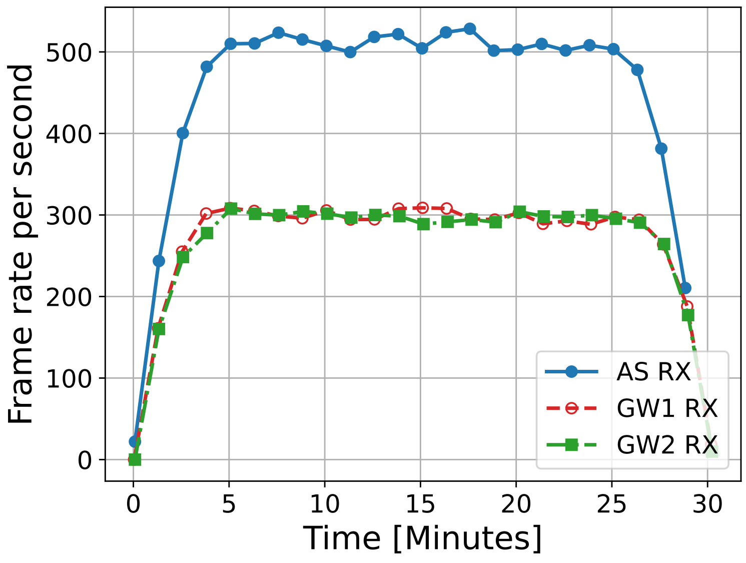
Parameter	**Value**
Deployment area	circular area of 1km radius
Number of Gateways	2
Gateways deployment	equally spaced at 0.15 km from the center
Number of Devices	3000
Device deployment	uniform
Device activation pattern	progressively activated with an interval of 0.1 sec
Device activation method	OTAA
Transmission Data Rate	$DR = 5$
Bandwidth	$BW = 125KHz$
Spreading Factor	$SF = 7$
Propagation Coefficient	$\eta = 2.9$
Frames transmitted	500
Frame transmission rate	1 frame every 3 seconds
Frame size	24 bytes
Frame delivery ratio	based on interference model defined in [25]
Aggregation window size	30 seconds
Experiment duration	30 minutes

processing being executed at the E2GW. Moreover, they help us identify the usage frequency of the DDF for detecting duplicate deliveries to support *"at-most-once"* and *"exactly-once"* QoS guarantees.

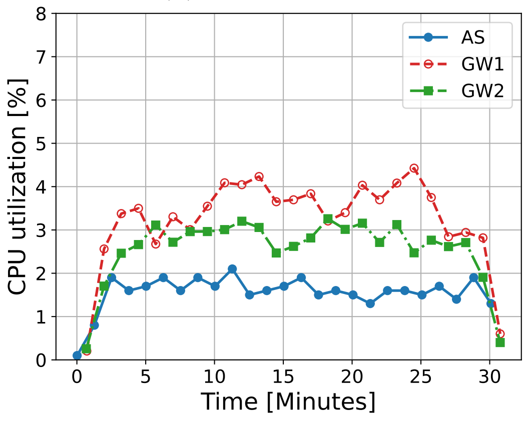
First scenario: baseline performance of Legacy elements

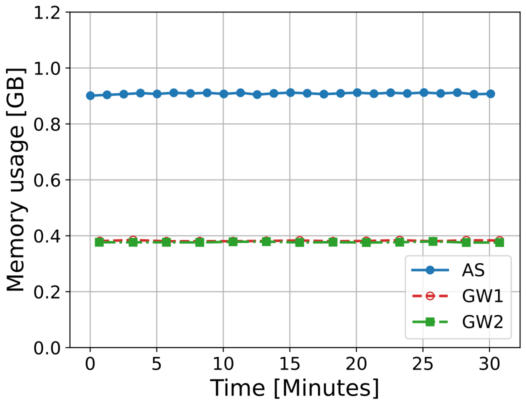
I start by measuring the performance of the LoRaWAN deployment when only the legacy components are activated. Fig. 7.3 depicts the overall performance in terms of the performance metrics defined in the previous section. Since the devices are progressively activated, during the first five minutes the number of frames received by each gateway and consequently transmitted to the AS increases. During the next period of 20 minutes, given that all the devices are activated, a steady reception of frames is observed from both gateways and their forwarding to the AS. During this period, all devices collectively transmit 1000 frames per second. Finally, during the last 5 minutes of the experiments, after having transmitted all 500 frames, the devices deactivate. Thus during this last period the total number of frames received by the gateways eventually drops to zero.

Looking into the period where all the devices are activated in Fig. 7.3a, to be noted that although a total of 1000 frames are transmitted every second by all the devices, it can be observed that each gateway correctly receives approximately 300 frames per second. The number of lost frames is due to the integrated FDR model used in this study, as presented in Sec. 7.1, which is about 31% at each GW, equivalent to approximately 300 frames per second. This is an indicator of the number of transmission failures at the wireless medium due to interference resulting from the given device density. This kind of visualization is essential for



(a) Frame rate received per second by each GW and by the AS





(b) CPU usage of the host machines accommodating the GW and the AS

(c) Memory usage of the host machines accommodating the GW and the AS

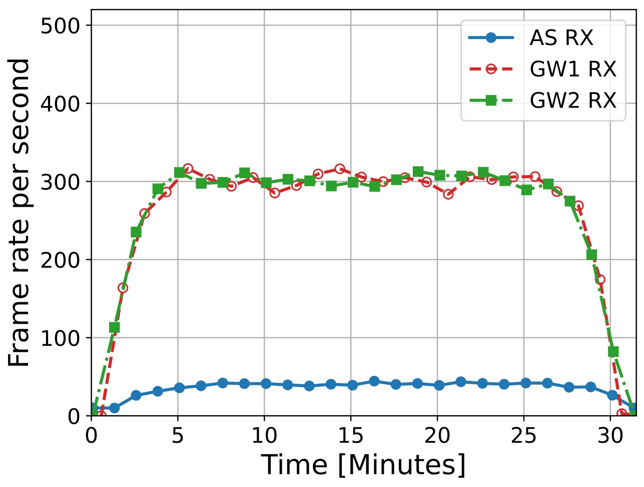
Figure 7.3. Baseline performance for ED, GW and AS when only legacy components are activated via the TheThingsNetwork public infrastructure.

understanding the load distribution among different gateways in the network.

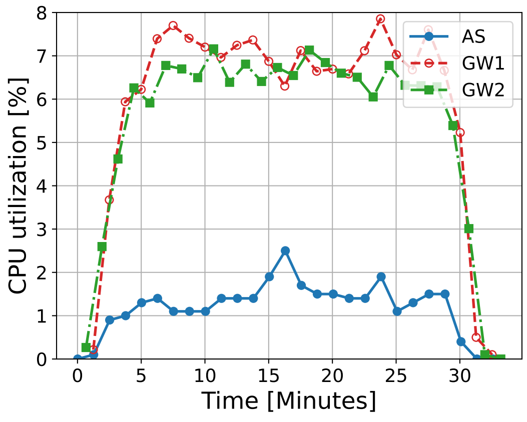
Among these 300 frame transmissions properly received by each gateway every second, some frames are received by only one gateway while others are received by both. Those frames received by both gateways are detected as duplicate frames when they reach the NS and are thus dropped. Therefore these frames represent the union probability of the frames received at each GW and expressed as $P(GW_1)$ and $P(GW_2)$ respectively, and computed $P(GW_1 \cup GW_2) = P(GW_1) + P(GW_2) - (P(GW_1) \cap P(GW_2))$, where the intersect probability represents the duplicate frames. This equation accounts for the probability of frames received at both gateways, ensuring no double counting of the overlapping frames removed by the NS. Finally, the AS receives approximately 520 frames per second, that accounts for about the 50% of the frames emitted from the emulated devices.

During the experiment, it is observed that the resource utilization of both GWs and also for the AS are maintained at minimum levels. In terms of computational load, Fig. 7.3b the CPU usage is lower in the host machine accommodating the AS, average of 1.8% concerning 3.2%, due to the more powerful hardware. Fig. 7.3c shows the memory usage at the host machines that accommodate the GW and the AS. Again, the memory usage at the AS is higher than at the GWs. This is because it runs an Ubuntu operating system, as opposed to the Raspbian operating system deployed on the two host machines where the GWs are deployed. The Ubuntu system is more resource-intensive, which contributes to the observed higher memory usage of 1GB with respect to the 0.4GB perceived from the host machines accommodated by the GWs.

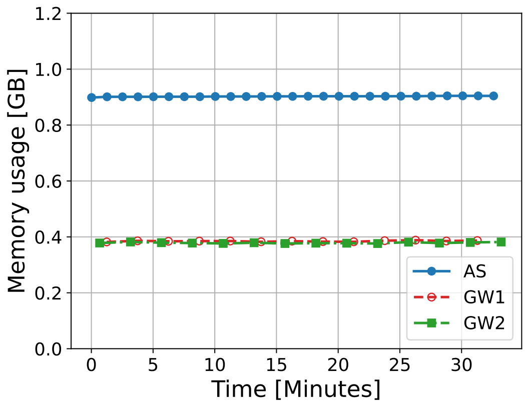
Second scenario: performance of the E2L extensions



(a) Frame rate received per second by each E2GW and by the AS



(b) CPU usage of the host machines accommodating the E2GW and the AS



(c) Memory usage of the host machines accommodating the E2GW and the AS

Figure 7.4. Overall performance for E2ED, E2GW and AS when E2L extensions are activated via the TheThingsNetwork public infrastructure.

I now proceed with the performance evaluation when the E2L extensions are activated. Fig. 7.4a depicts the number of frames received by each E2GW, which is more or less the same as in the case when legacy GWs were used as depicted in Fig. 7.3a. On the other hand, the number of frames transmitted to the AS is drastically reduced due to the edge-computing tasks deployed to the E2GW module executor. In comparison to the legacy scenario, the number of frames received by the AS is reduced to approximately 10%: on average, about 52 messages compared to 520 frames. This drastic reduction illustrates the efficiency of the aggregation function, which can help conserve network resources and reduce data transmission volume without losing essential information.

In terms of the resulting computational load of the E2GW when the E2L components are activated, Fig. 7.4b indicates an increase on the CPU usage by about an average of 6.8 percentage points. This increase in the CPU usage is due to the edge-computing task being executed that carries out a series of one-to-one and many-to-one operators on the 30-seconds windows. Moreover, the increase in the CPU usage is also due to the deciphering and ciphering of each frame so that the data payload can be extracted. Given all these considerations, the shift of processing from the cloud to the edge does not result in any significantly additional resource consumption in terms of CPU usage. On the other hand, the CPU usage of the AS is slightly reduced by an average of 1.2 percentage points since the AS is no longer aggregating the frames as this is now done at the edge of the network. Similarly, in terms of memory usage, it seems that the activation of the E2L components does not present any significant increase when compared with the legacy scenario.

These experiments provide evidence that the E2L extensions can help reduce the network traffic of the core backbone network and also reduce the processing load on the AS, without significantly increasing resource utilization at the edge level. This finding is crucial as it demonstrates the potential of edge computing when combined with LoRaWAN in managing network resources more efficiently.

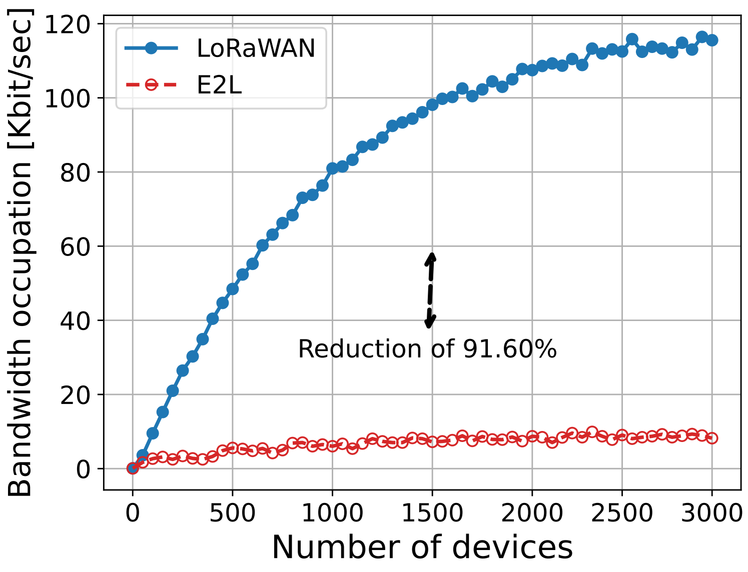


Figure 7.5. Assessment of the benefits for the proposed approach in terms of bandwidth utilization using the legacy LoRaWAN standard as baseline in comparison with the E2L extensions when the edge-computing task is deployed to the E2GW.

To further illustrate the benefits provided by the E2L approach, the first five minutes of the experiment are examined as the number of devices are activated. I calculate the bandwidth utilization of the IP-level network backbone connecting the gateways with the NS of the public infrastructure. Fig. 7.5 depicts the drastic reduction in bandwidth utilization as the number of active devices increases from 1 to 3000. Using this specific reference, it can deduce that there is an average bandwidth reduction of 91.60% when the number of active E2ED is 1500. This demonstrates that the E2L approach, with its strong data aggregation capabilities at the edge level, can significantly reduce network bandwidth usage. This is particularly beneficial in large-scale deployments or high-density environments with numerous active devices, where the gateways utilise internet connections provided by various access technologies, e.g. 3G/4G access technology with lower bandwidth. This also applies to situations where multiple gateways share the same backhaul through a single central NS.

Third scenario: verifying the backwards compatibility of the E2L

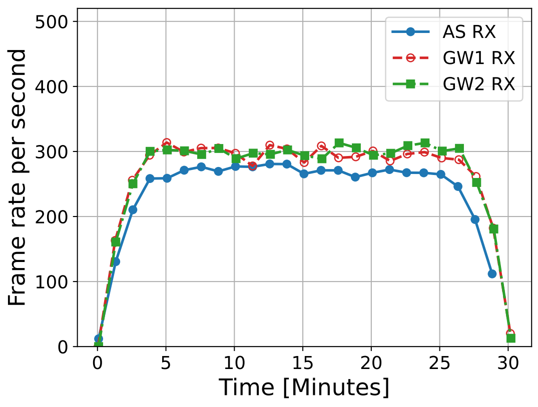


Figure 7.6. Assessment of the backwards compatibility of the proposed approach.

The previous experiments were conducted by utilising a public infrastructure, that is TheThingsNetwork. I wish to look further into the backwards compatibility of the E2L extensions by conducting an experiment in a scenario with 1500 legacy EDs and 1500 E2ED deployed over the same area where two E2GW are operating. In this case, the E2L driver equally assigns the 1500 E2ED to the two E2GW. At the same time, the 1500 legacy EDs are connected by both E2ED. Fig. 7.6 depicts the number of frames received by each E2GW. The results are substantially similar to before for what concerns the received frames at each E2GW. However, the number of frames received at the AS falls somewhere in between the values observed in purely legacy and E2L scenarios. Since only half of the deployed devices are E2ED, the edge-computing task executed at the two E2GW can only aggregate the sensor values included in the frames transmitted by the 1500 E2ED. Specifically, since the E2ED transmit on average 500 frames per second, these are reduced, on average, to 50 messages that are transmitted to the AS. The remaining 500 frames per second received on average by the legacy EDs, are directly forwarded for processing to the AS through the NS. I remark here that a subset of these frames are not received due to the interference phenomena. This scenario illustrates how the system can handle both legacy EDs and E2ED devices simultaneously, eliminating the need to replace existing devices during a transition period. This feature enhances the system's flexibility and adaptability, making it more feasible for real-world implementations.

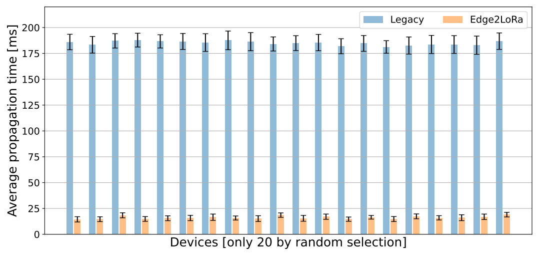


Figure 7.7. Impact of the proposed method on data latency to the AS. The bar plot represents the average latency for each terminal in the legacy scenario, with the standard deviation indicated by the error bar.

Latency in data reaching the AS is another key factor to consider. As illustrated in Fig. 6.1, the direct connection between the gateway layer and the AS bypasses the NS, reducing overall latency since the NS no longer processes the frames. To quantify this reduction in processing overhead, a private infrastructure is used where the NS, AS, and E2GW are located within the same local network, minimizing network transmission delays. For the experiment, 20 out of 3000 terminals are randomly selected, and the average delay time comparison between the legacy and E2L scenarios is plotted. The results are depicted in Fig. 7.7. In the legacy setup, the average time for frames to travel from the gateway to the AS, passing through the NS, is around $180ms$, shown in a bar plot with standard deviation presented via error bars. In contrast, in the E2L setup, where the E2GW connects directly to the AS, the average delay drops to approximately $16ms$ for the same 20 devices configured as E2ED. Note that, in the legacy case, frame aggregation happens at the AS, while in E2L, aggregation occurs at the E2GW. This direct gateway-to-AS data transfer results in a 92% reduction in delay when comparing the two scenarios.

Fourth scenario: handover in case of failures

During the execution of the experiment where all the devices are E2ED and both gateways are E2GW, a failure occurring at one of the two E2GW at the 15^{th} minute is introduced. The E2L driver notices that the gateway registry has marked the failing E2GW as being offline and re-assign the E2ED to the other E2GW. Fig. 7.8 shows how the frame rate of the failing E2GW drops to zero while the traffic of the other E2GW almost doubles after the re-assignment of the E2EDs. The experiment illustrates the resilience of the proposed approach. It is evident that the processed frames at E2GW 1 drop to zero immediately at the 15-minute mark. Despite this, the number of received frames at the sink perceived a lower influence because all the E2ED are now managed from the E2GW 2.

7.2.2 Security Analysis

I now proceed with the analysis of the security properties of the E2L extensions. Focusing on the three cryptographic properties that the group key establishment

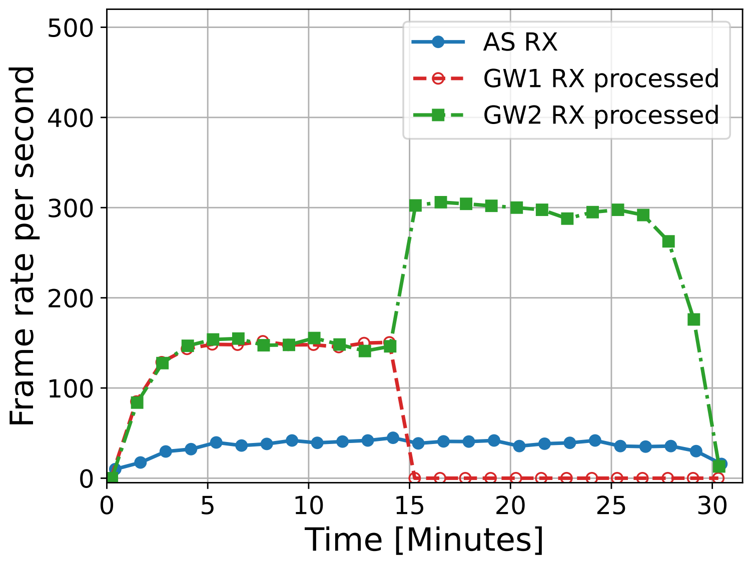


Figure 7.8. Simulating a failure of E2GW 1 after 15 minutes of activity to showcase the resilience of the proposed method.

protocol shall guarantee, which are: i) the computational key secrecy; ii) the decisional key secrecy; iii) key independence. The E2L security analysis including the three properties is reported in the following sections.

Computational & Decisional Key Secrecy

It must be computationally infeasible for any passive adversary to discover any key, while no information may be leaked other than the public key. The security of the protocol is based on the use of ECDH, an algorithm that enables secure communication between two parties over an insecure channel.

The security of ECDH is derived from the computational difficulty of solving the Elliptic Curve Discrete Logarithm problem. This problem involves finding the discrete logarithm of a point on an elliptic curve, which is considered computationally difficult to solve. The large size of the elliptic curve group and the complex mathematical operations involved make it challenging for an attacker to derive the private key from the public key [61]. By utilizing ECDH, the E2L ensures the confidentiality and integrity of the communication, making it resistant to attacks attempting to compromise the shared secret key.

To evaluate the computational and decisional key secrecy, I consider a Man-in-the-Middle (MITM) attack scenario. In the proposed protocol, an attacker attempting a MITM attack would be unable to eavesdrop on the LoRaWAN frames because they are encrypted using the *AppSEncKey*. Even if an attacker manages to break this key by exploiting a vulnerability in the OTAA process, it would still be computationally challenging to guess the *EdgeSKey*.

Key Independence

A passive adversary that manages to acquire a subset of the keys must not be able to discover any other information about the remaining keys. This property is further decomposed into:

- Forward Secrecy: A passive adversary that knows a subset of keys must not discover any subsequent keys.
- Backward secrecy: A passive adversary that knows a subset of keys must not discover any preceding keys.

To enhance the overall security of the activation methods, my proposal offers a way to periodically refresh the **Edge Session Keys** by performing a new execution of the protocol. Since a new pair of ephemeral ECC keys is computed by every actor in each execution of the protocol, the key independence is guaranteed. There is no correlation between the previous set of keys and the new one, or any subsequent ones.

Remark that only a single uplink and downlink LoRaWAN frame are needed for the re-execution of the protocol, it does not significantly impact the performance and energy consumption of the E2ED.

Finally, it is important to note that no security analysis of the communication over IP-based networks is performed, as the solution does not impose any specific protocol requirements.

Chapter 8

Integrating Distributed Application

The International Data Corporation (IDC) [52] has projected that the global volume of data will reach an astounding 175 zettabytes by 2025. For this reason the analysis of large and diverse datasets generated, which is commonly referred to as IoT big data [59, 60], needs to be done across the entire Cloud Edge Computing Continuum (CECC). Distributed computing plays a crucial role in the reduction of extensive network traffic, enabling processing and real-time response to scale network dimensions.

Edge4LoRa (E4L), enhances the pre-existing Edge2LoRa (E2L) [42] architecture by incorporating a distinct computing module. This module, capable of processing the data stream received at the network edge, ensures both modularity and reliability. It embodies a Map/Reduce engine, based on Apache Spark [8], and is capable of executing multiple processing applications, including anomaly detection algorithms and data reduction techniques such as aggregation.

My solution introduces an automated traffic flow management system that leverages real-time monitoring of radio coverage and computational capacity. This system enables seamless traffic redirection between gateways without relying on a central server. I have developed a logic that accounts for device mobility and the resource utilization of each gateway, allowing the system to dynamically select the optimal location for data processing.

In addition, the proposed architecture demonstrates excellent scalability, particularly suited for larger environments, and robust performance, ensuring the accurate execution of processing algorithms under diverse conditions. The enhancements provided by E4L effectively address the limitations in QoS support outlined in Table 7.2, enabling an *exactly once* QoS guarantee in every scenario. Furthermore, I assess the performance of this approach under various configuration settings and scenarios in the testbed, demonstrating its effectiveness.

8.1 System Architecture

This section provides a detailed description of the key components that make up the architecture of the proposed approach. Fig. 8.1 illustrates the process of stream processing, where, from left to right, IoT sensors generate data continuously that is ingested and buffered by the stream processing engine. The data undergoes processing actions such as filtering, mapping, joining, or analytics operations like counting and averaging. After processing, the data are sent to other application services for further analysis or storage.

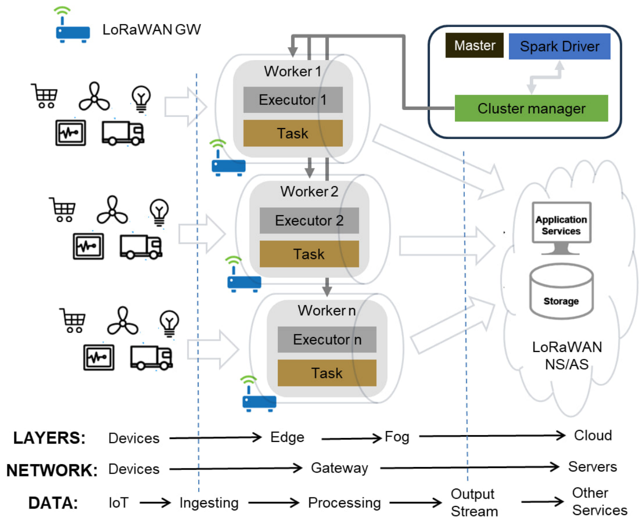


Figure 8.1. IoT collection/processing architecture is presented including devices, edge, fog and cloud. The figure illustrates the mapping between the IoT collection architecture, the Spark infrastructure, and the deployment of the LoRaWAN network.

In the recent past, a variety of frameworks have been developed to facilitate dynamic data processing. These include renowned platforms such as Apache Spark [65], Apache Flink [7], Apache Storm [9], and Nebula Stream [47]. Taking into account several critical factors, Apache Spark stands out as the preferred choice for stream processing. Its superiority stems from its ability to provide rapid in-memory processing, real-time capabilities, minimized latency, effective lazy evaluation, and simplified code implementation. These features make it a highly beneficial solution for managing data streams both effectively and efficiently, particularly in the context of IoT. Essentially, Apache Spark [65] encompasses batch processing along with real-time streaming and also offers APIs for Java, Python, and R, finding its streaming processing ability particularly useful, utilizing them in a distributed fashion.

As depicted in Fig. 8.1, in the Spark architecture, the cluster manager driver node is represented to the right. The cylinders represent the daemon processes that are active on and manage each worker node. These processes originate from the Cluster Manager. The Spark Driver acts as the central command center for a Spark application. It is responsible for managing the execution of the Spark application and maintaining the application states across the Spark cluster.

It operates with its driver, called 'master' and 'worker' abstractions, which are linked to physical machines, unlike Spark's processes.

This work has focused on the capabilities of Apache Spark, specifically those relevant for the integration of edge processing within the LoRaWAN architecture. The aim has been to incorporate Apache Spark directly onto the GW, thereby enhancing the processing capabilities at the edge of the network, Additionally, I designed a global message broker architecture to move IoT traffic from one GW to another to implement low latency application (e.g. [46]). As visible in Fig. 8.1, the GW capability are extended to support Spark worker.

8.2 Protocol Definition

E4L is built on top of the E2L framework. E2L comprises several key elements, including the device registry, the group key establishment method, and sensor data stream processing. The EDs compatible with the E2L framework, denoted as E2ED and the GWs compatible with the E2L framework, denoted as E2GW. Each E2ED is served by an E2GW, which is responsible for performing data processing tasks on the received sensor data streams. In this improved implementation, I undertake a substantial reconstruction of the GW layer to facilitate edge processing using the Spark Engine. Additionally, an interface was developed to facilitate optimal flow redirection and load balancing between GWs using the MQ Telemetry Transport (MQTT) protocol.

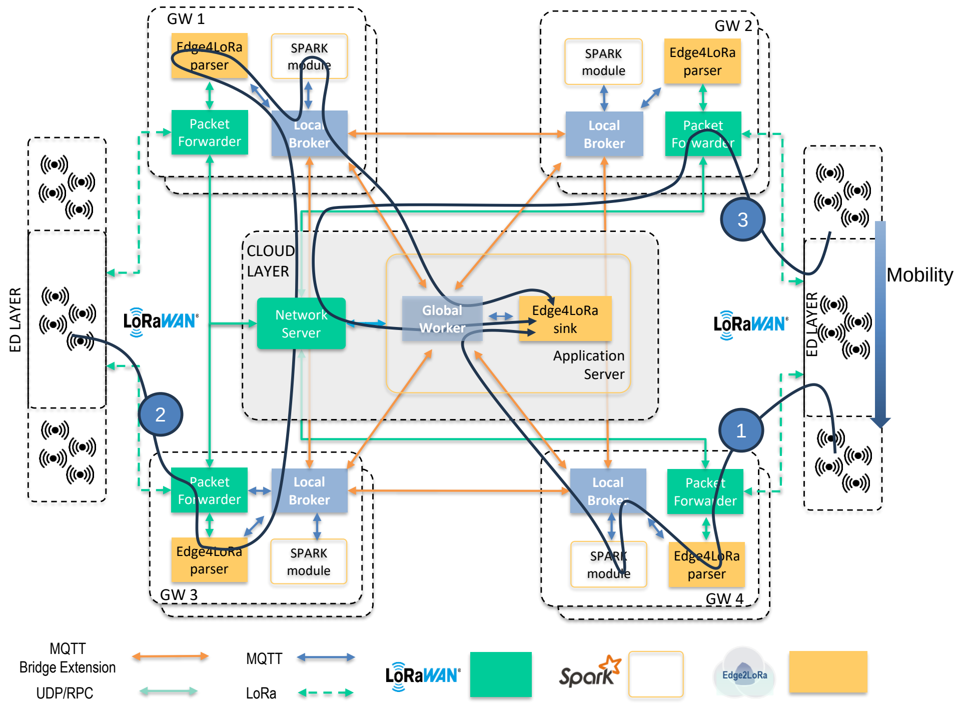


Figure 8.2. The figure shows a global overview of the E4L architecture.

In order to achieve this, E4L introduces three new components at the GW layer: the Spark engine, the message broker, and the E4L parser. These newly created elements are illustrated in Fig. 8.2. As can be seen in the figure, the E2GW structure plays a pivotal role in enabling the E4L mechanism. To ensure efficient frame exchange between E2GWs, a hierarchical broker system is integrated into the system. Each E2GW is paired with a dedicated local broker, which facilitates the exchange of frames between the E2GW and the assigned E2GW responsible for handling the frame. Additionally, a global broker is introduced at the AS level, serving as a bridge between the AS and the hierarchical broker system. The global broker acts as a central hub for receiving processed frames from the designated E2GW and forwarding them to the AS for further processing and analysis. This configuration ensures streamlined data flow and enables effective coordination between the E2GW and the AS.

Simultaneously, a logical implementation is directed towards the AS to select the E2GW in charge of processing frames from a specific E2ED. At the base of this structured E2GW architecture, the E4L mechanism optimizes the computational balance between the E2GWs and the edge processing components within the LoRaWAN network.

In E4L deployment, multiple alternatives of frame routing, processing, and storage may happen concurrently either at the edge or at the cloud. In the following paragraphs, the three possible scenarios of up-link frame transmissions are detailed. For the sake of clarity, I will confine this discussion to up-link traffic, which constitutes the majority of LPWAN traffic.

Case 1: Frames sent by an E2EDs are received by the E2GW that is tasked with the processing (label 1 in Fig. 8.2)

Considering a simple scenario first, where processing and/or storage of frames takes place exclusively at a single point in the CECC. Frames consumed at an associated E2GW are not forwarded to the NS but instead, the AS is notified directly about the outcome of edge-based processing. Access to the data within the frames is facilitated by establishing a group key among the AS, edge (E2GW), and E2ED. Two shared session encryption keys are created between the E2ED, E2GW and AS. They are used to enable secure encryption and decryption of the frame payload as well as the integrity of the edge-specific frames (Sec. 6.3).

Case 2: Frames transmitted by an E2EDs are received by E2GWs and forwarded to the E2GW responsible for the process (label 2 in Fig. 8.2)

This scenario constitutes more complex cases where some frames may traverse the network by following alternative paths that involve 2 E2GWs elements. This alternative represents the novelty introduced from E4L that enables supporting to balancing of the computational process. In these latter scenarios, it is crucial to ensure that the frames are not processed and/or stored multiple times throughout the CECC when it is required.

Case 3: Frames transmitted by a EDs are received by E2GW and considered as traditional LoRaWAN frames, that can not be processed at the edge (label 3 in Fig. 8.2)

At the same time, frames transmitted from legacy ED and/or received from legacy GWs coexisting in the infrastructures arrive at the AS via the NS. No access to the data can be affected on this flow.

Here, E4L operates over the conventional LoRaWAN architecture, ensuring backward compatibility.

8.3 Implementation Details

In the following section, I introduce the implementation of a distributed application on E4L, demonstrating how it operates through the inclusion of two distinct applications. The first application performs time window aggregation, calculating the average of Received Signal Strenght Indicator (RSSI) and Signal to Noise Ratio (SNR) within a predefined time interval. The second application employs a Hampel filter [28] to detect anomalies or outliers in high-level application values. These applications provide practical examples of the functionality and versatility of the E4L system.

An example of the main code section for configuring the E4L process module is as follows:

```
spark_conf = SparkConf().setAppName("e4l-process")
sc = SparkContext(conf=spark_conf)
ssc = StreamingContext(sc, 1)  # 1-second batch interval
#using MQTTUtils to receive the stream mqtt
readings = MQTTUtils.createStream(ssc, BROKER_ADDRESS,
MQTT_TOPIC, MQTT_USERNAME, MQTT_PASSWORD)
readings.foreachRDD(
    lambda rdd: rdd.foreach(process_reading)
)
ssc.start()
ssc.awaitTermination()
```

the code described above begins by initializing `SparkConf` and `SparkContext` objects to define the application. Subsequently, a `StreamingContext` is created with a batch interval, indicating that data will be processed in time-defined intervals (e.g. 1-second). Furthermore, since the spark module acts as a subscriber/publisher of the MQTT broker defined within the E2GW, it is essential to specify the broker parameters. Finally, each Resilient Distributed Dataset (RDD), which represents the distributed collection of data, is processed via the `process_reading` function. This function executes the defined processing algorithm and returns the results.

Time-Window Aggregation: Average Network Values

The first application algorithm is designed for time-window aggregation of network level information. The algorithm processes frames containing RSSI and SNR values and computes their average within a predefined time window, in this case set to 1 minute. Using streaming data processing methodologies, it efficiently calculates the average of RSSI and SNR over successive time intervals, grouped by ED address. The aggregated data is then forwarded to the AS, providing concise data reduction.

Anomaly Detection With Hampel Filter

As a second application, a Hampel filter is employed for anomaly detection in high-level application values. This type of filter represents a robust outlier detection and correction technique, widely used in various environments to mitigate the impact of spurious data generated by sensor measurements. The algorithm operates by analyzing data streams within a moving window to identify outliers or anomalies.

The function that performs the steps described above is defined as follows.

```
hampel_filter(input_data, window_size, n_sigma)
```

where the `input_data` represents the data stream that will be processed, along with the `window_size`, which determines the size of the moving window within which anomalies can be detected and the `n_sigma`, which represents the number of standard deviations for outlier detection.

8.4 Performance Evaluation

This section offers an in-depth examination of various performance facets of the proposed architecture. The evaluation is carried out using a hardware setup available in the laboratory. The performance metrics presented in this section are derived from the previously discussed applications, which are executed within the E2GWs at the network's edge. In addition, I consider three distinct scenarios for this evaluation. The first scenario focuses on data reduction, the second scenario illustrates the edge scaling activation E2GWs, and the final scenario is specifically designed to demonstrate the advantages in a context where EDs mobility is taken into account.

Data injection into the system is facilitated by a LoRaWAN device simulator[1] designed to closely mimic the network environment of a large-scale LoRaWAN system. This sophisticated tool acts as a complete emulator of LoRaWAN terminals, enabling the creation of detailed test environments comprising thousands of terminals capable of injecting authentic LoRaWAN frames. At the same time, the module can be programmed to reproduce the actions of the recorded data set to recreate the real IoT scenarios. For the experimental evaluation of this work, the simulator is configured to reproduce an environment with 8 or 100 E2EDs, the EDs simulator also includes an analytical model to reproduce realistic frame lost behavior in the free space environment. Simulated devices inject real frames into the system based on data recovered from a data set comprising 190,503 records in a vineyard located in Reggio Emilia[2].

The experiment setup for evaluating the E4L system involved a comprehensive hardware deployment that included three Intel NUCs, each equipped with an Intel i5-6260U CPU and 8GB of RAM, running Ubuntu 22.04.3 LTS. These machines were allocated for the EDs simulator, the NS, and the AS. Additionally, two Raspberry Pi 4 Model B units, powered by Broadcom BCM2711 SoCs and 4GB of RAM, operating on Debian GNU/Linux 11, were utilized as the E2GWs in the setup.

The data collected during the experiments conducted focused on key metrics such as total frame count, frame delivery by E2GWs, and computational resource utilization. These metrics were selected to offer a comprehensive view of the system's performance and suitability for practical implementation in real-world IoT scenarios.

[1]https://github.com/Edge2LoRa/e2l-device-simulation
[2]https://github.com/emanueleg/lora-rssi/blob/master/vineyard-2021_data/sensors_data.csv

8.4.1 Data reduction scenario

In the first scenario, the stream engine processes incoming LoRaWAN frames, aggregating them by device and computing the average of network parameters such as RSSI and SNR.

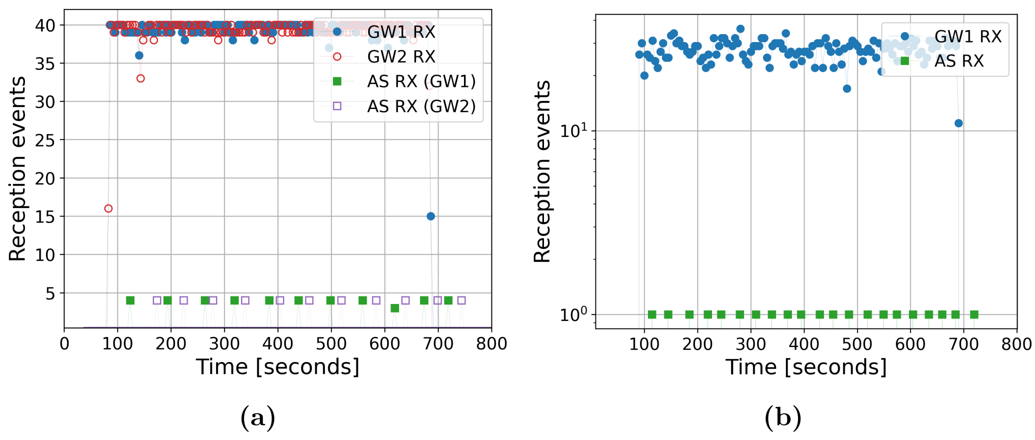


Figure 8.3. Reception events for each E2GW and AS within the context of the time-window aggregation application (a), and distributed Hampel filter (b).

Fig. 8.3 illustrates the performance metrics of the scenario with the E4L system activated. Here, the AS is configured to evenly distribute data processing responsibilities among each E2GWs. In Fig. 8.3a, the count of reception events by each E2GWs and the AS is depicted for the time-window aggregation application. There is a noticeable decrease in the number of frames arriving at the AS. For this experiment, the stream processing engine executes a time window aggregation, where at each 60-seconds interval, it aggregates data for each E2ED and computes the average of the RSSI, SNR.

For this experiment, 8 E2EDs have been taken into account. Each of these E2ED transmits at a rate of one frame per second, with each frame being configured to be 24 bytes in size. Statistics are collected every 5 seconds, which means 40 frames are acknowledged for each report as shown in Figure 8.3a. Aggregation is performed at 60-second intervals for each E2ED. This triggers a transmission to the AS every 60 seconds for each EED, and the processing is evenly distributed between the two E2GWs. As depicted in Fig. 8.3a, 4 frames are received at the AS from each E2GW. Compared to the legacy scenario, the events reaching the AS are significantly fewer, constituting 1 reception event every 60 seconds, which corresponds to an aggregation for each device comprising 480 frames.

For the second process application, a distributed Hampel filter application is applied to the LoRaWAN stream and subsequently detecting anomalies or outliers. In this configuration, only 1 gateway is utilized, and the Spark module performs anomaly detection using the hampel filter as the processing query. The application delineates a time-window aggregation, which produced a frame in which all anomaly detection events are transmitted to AS. Fig. 8.3b depicts the reception events at the E2GW, as well as the detection event received at the AS. This scenario also employs 8 E2EDs, where a frame loss model is implemented with a set loss rate of 25%. As shown in Fig. 8.3b, the number of received events at the E2GW is approximately 30, therefore a single frame is triggered from each aggregation, which contains the number of anomalies detected in the window.

Table 8.1. Comparison of anomalies detected from the execution of the distributed Hampel filter application versus its execution as a standalone application.

Frame Loss	Edge4Lora	Standalone	Anomaly Loss
0%	1159	1159	0%
10%	923	1159	20%
20%	750	1159	35%

Moreover, the experiment involves a performance evaluation intended to showcase the efficiency of distributed anomaly detection deployed by E4L. To this scope I compare the distributed execution of the application within E4L and its standalone execution, including the frame loss extracted from the formula described in [25]. Tab. 8.1 encapsulates the results obtained, which are detailed as follows: In the first test, without frame loss, 1159 anomalies were detected, validating 100% of the present data and affirming the effectiveness of the edge processing implemented. Therefore, I conducted two other tests that incrementally increased frame loss exactly 10%, and 20%. The second test identified only 923 anomalies, marking a decrease of 20.3%, attributable to system losses such as collisions or non-covered devices. Despite losing frames, the system continued to operate. In the final test, 750 anomalies were detected, reflecting a reduction of 35.3%. It becomes clear that increasing the number of E2GWs within the system leads to a decrease in frame loss and, consequently, enhances anomaly detection efficiency.

8.4.2 Auto-scaling scenario

To assess the scalability and load-balancing capabilities of the proposed architecture, an experiment was carried out in which the number of devices connected to the LoRaWAN network gradually (every 1 second) increased from 1 to 100. The objective was to evaluate how the system automatically recognizes the processing load at the E2GW level and activates an additional E2GWs when the load exceeds a predefined threshold. This dynamic load-balancing mechanism ensures efficient processing and prevents performance degradation due to overload. The E4L logic is introduced to monitor the behavior of the system as the device count increased and analyze the load distribution among the E2GWs.

The results of the evaluation are depicted in Fig. 8.4, which displays the transmission events E2ED and the count of frames processed by each E2GW. Initially, when the number of E2ED remains below 50, a single E2GW is tasked with processing the incoming data from the E2ED. However, as the number of E2ED escalates after 100 seconds, the processing load also expands, prompting the system to automatically engage a second E2GW to manage the additional incoming E2ED (both E2GWs are within the same coverage area). After the 150-second mark, both E2GWs are operational and frame processing is equally divided between them, with each E2GW processing exactly 50 frames every 5 seconds. This load balancing mechanism ensures that the processing load is evenly distributed across multiple E2GWs.

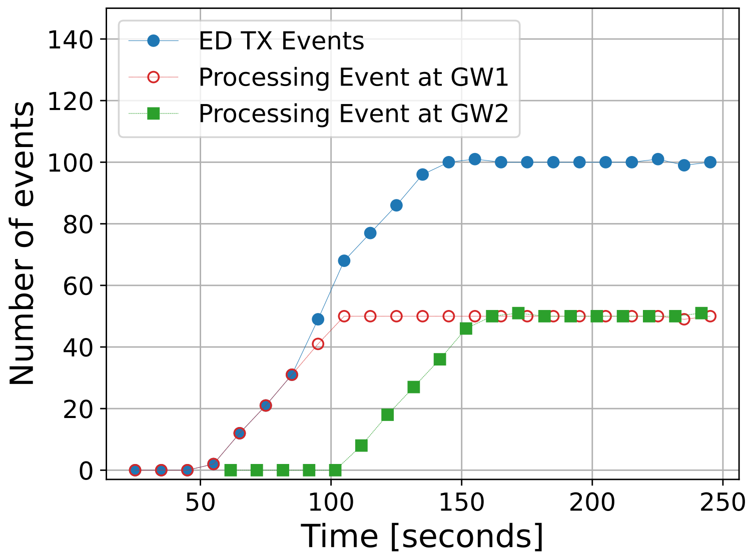


Figure 8.4. Display of system performance during the execution of the load balancing algorithm. This includes ED transmission events and processing at each E2GW as the number of E2ED progressively increases

8.4.3 Mobility scenario

To evaluate the system's capability to handle E2ED mobility within the LoRaWAN network, an experiment was conducted considering scenarios where E2ED moved from the coverage area of one E2GW to another. The objective was to assess how the system recognizes the movement of the E2EDs and dynamically assigns processing responsibilities to the appropriate E2GW. This dynamic processing assignment ensures efficient data processing while accommodating device mobility.

As shown in Fig. 8.5, the system's ability to recognize E2ED movement and dynamically assign processing responsibilities was evaluated. Initially, a group of 100 E2ED were within the coverage area of E2GW1, and processing for these E2ED was assigned to E2GW1. Frames were transmitted every 3 seconds, and measurements were collected every 5 seconds. The total average number of received events during the experiment, after the transitory period, was 167 every 5 seconds. The number of processing frames may vary as they can occur in different locations. As E2ED moved out of the coverage area of E2GW1 and entered the range of E2GW2, starting at 800 seconds in Fig. 8.5, the number of reception events at E2GW2 gradually increased. However, the processing for the device remained assigned to the original E2GW. A soft handover mechanism was implemented to facilitate a smooth handover. During a safe guard interval period, frames in the responsibility of E2GW1 were forwarded from E2GW2 to E2GW1 to avoid any ping pong phenomena. At the 1200-second mark, the system held the device's movement, and the load balancing mechanism took effect, progressively assigning the responsibility for processing to E2GW2. During this reconfiguration period, the system performed a new assignment, designating the E2GW within the E2ED new coverage area as the responsible E2GW for processing. Once the reconfiguration was complete, the frames from the E2ED were processed by the local E2GW, ensuring efficient data processing and

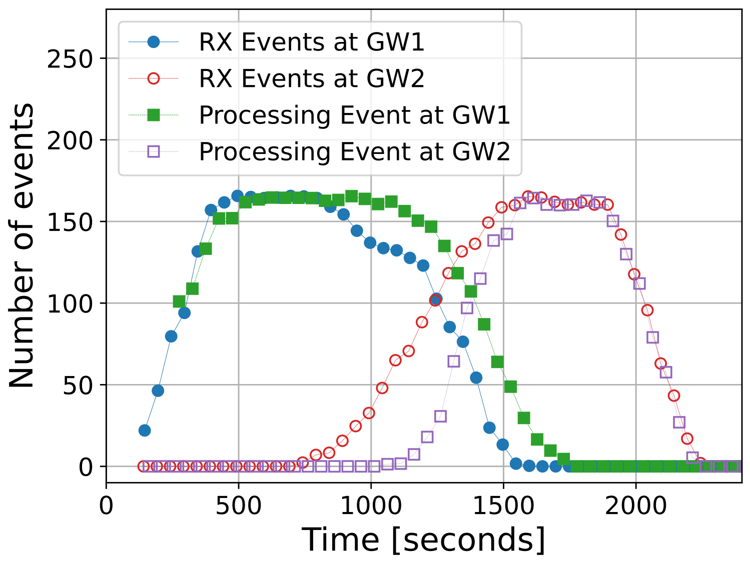


Figure 8.5. Display of system performance during the execution of the load balancing algorithm. This includes ED transmission events and processing at each E2GW when E2ED are subjected to mobility.

minimizing latency.

Chapter 9

Results, Conclusion & Future Works

The ever-growing resource needs of modern-day large-scale Internet of Things (IoT) deployments regarding guaranteed low latency and the massive data transfer rate are constantly pushing the boundaries of technologies and requiring a paradigm shift from the traditional producer/consumer model. At the same time, the new paradigm called Cloud Edge Computing Continuum (CECC) that evolves beyond the more traditional central cloud/DC with ultra-high-end processing powers and high-capacity networking infrastructure to extend their coverage all the way to the network edge, has not yet been fully extended to include edge and far-edge resources available in IoT deployments. This is mainly because Long-Range Wide-Area Network (LoRaWAN) adopts the design approach of using simple protocols to realise a centralised architecture that guarantees the security and confidentiality of the IoT generated data. Forcing the gateways to act as simple bridges, at one hand accommodates the rapid and low-cost deployment of unlicensed Low-Power Wide-Area Networks (LPWANs), on the other hand excludes them from potentially acting as trusted intermediate processing and storage elements in the CECC.

Recently some attempts have been made in modifying the specified operation of the LoRaWAN standard protocols, proposing alternative architectures to reduce the significant pressure imposed on the central cloud services in cases of massive IoT data streams or time-constrained consumption of IoT data. At the same time, introducing changes to the architecture and the specified operation of the protocols while maintaining backward compatibility is a challenging discourse. As a result, to the best of my knowledge, there is currently no proposal that incorporates LoRaWAN in the CECC both for network management and application data processing that maintains the practical philosophy of LoRaWAN and respects backward compatibility.

This thesis presents Edge2LoRa (E2L), an enhanced version of LoRaWAN that transforms LoRaWAN Gateways (GWs) from simple bridges to edge processing units in a secure and privacy preserving manner. By taking the advantage of edge/-fog computing, E2L offers substantial benefits to allow the execution of Big Data analytics across the CECC over LoRaWAN. The proposed design enables dynamic cloud-edge-far-edge coordination based on network conditions and application-level parameters. The resulting system ensures backward compatibility for seamless interoperability between legacy and new elements without disrupting network operation and providing Quality of Service (QoS) guarantees.

I present a comprehensive design of the key components and rational choices within the E2L. The description points several key features of E2L and how it differs from standard LoRaWAN. Specific implementation decisions are provided that result into an efficient, secure, and adaptable infrastructure for optimizing sensor data stream processing in LoRaWAN networks. E2L does not limit the network operator to this particular usage. Other design choices are also possible so that operation is optimized based on different network topologies, device characteristics and Big Data analytics.

Secure communication is ensured through shared session encryption keys within the E2L system, with Elliptic Curve Cryptography (ECC) proposed for creating these cryptographic keys. ECC offers high security with lower resource consumption, making it suitable for resource-constrained environments like LoRaWAN End Devices (EDs) with minimal energy consumption.

Moreover is presented the implementation and evaluation of an edge processing module within the architecture of a LoRaWAN network, with a focus on leveraging edge computing principles, called Edge4LoRa (E4L). The integration of a dedicated processing entity into the E2L architecture was identified as essential to ensure reliability and modularity. It enables the execution of processing algorithms at the network edge, leading to a significant reduction in data volume transmitted to the cloud. This reduction is achieved through the implementation of aggregation algorithms and near real-time analysis, enabling timely and proactive actions. Furthermore, the distributed execution of processing algorithms across multiple gateways demonstrates comparable efficiency to the execution of a standalone application.

Furthermore, the experiment highlighted the system's ability to handle device mobility while minimizing processing disruptions. The efficient forwarding of frame traffic during the reconfiguration period allowed uninterrupted data processing, ensuring a smooth transition between Edge2 Gateway (E2GW) and maintaining low latency.

A performance evaluation was conducted to assess the correctness, security, and network utilization of the new system. This evaluation approach considers the co-existence of legacy EDs and GWs with new ones, creating the possibility of unprocessed frames arriving at the LoRaWAN Network Server (NS)/LoRaWAN Application Server (AS) via legacy equipment, while simultaneously, some parts of the data arrive at the AS after being processed by an E2L GW.

Adopting E2L and E4L could lay the groundwork for more efficient, scalable, and secure LPWAN deployments, enabling IoT networks to effectively manage the increasing demands of real-world applications.

Future Works

At the time of writing, preparations are underway for a poster and a demo to be presented at the `21st International Conference on Embedded Wireless Systems and Networks (EWSN'24)`.

The poster presents the LoRa Mobility and Coverage (LoRaMC) Dataset, a large dataset for LoRaWAN, which presents devices with mobility patterns in a city-scale and which aims at providing researchers with a means to develop new algorithms and assess their performance on common data, thus allowing for fair comparisons. LoRaMC comprises mobility traces of 3300 taxis integrating network coverage information related to 50 LoRaWAN GWs deployed throughout the operation in the metropolitan city of Rome.

Starting from the CRAWDAD Rome/taxi dataset [13] that includes location data of 320 taxis moving around the city and a new dataset with 3300 taxis is produced including radio-level information, not present in CRAWDAD.

The entire CRAWDAD Rome/taxi dataset is condensed into a single day by segmenting the dataset into 30 different subsets, one for each day. This division allows to assign each taxi an identifier consisting of both its ID and the day, forming a new identifier pair (ID, day). Once the tracks are organized by this new identifier, all data is projected onto a single day. Each of the 3,300 tracks is further divided into 288 'snapshots'. These snapshots capture the state of each taxi within that 5-minute window. For each taxi, if there is at least one recorded position in that time window, the locations are aggregated into a single point, representing the snapshot. If no position data is available for a taxi in a given snapshot interval, that taxi is excluded from the snapshot, simulating inactivity or a scenario where the device is turned off and unable to send or receive data

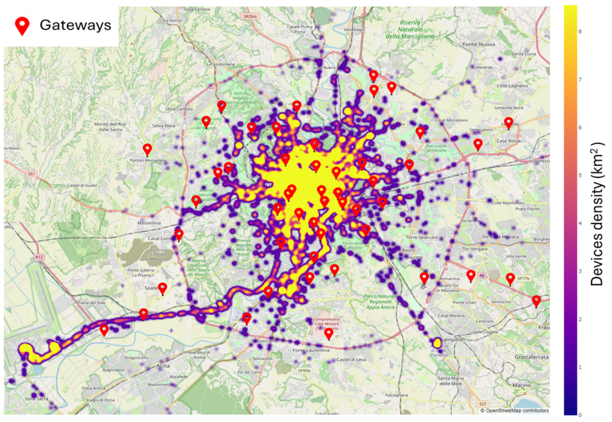


Figure 9.1. Dataset device density and coverage with 50 GWs over the entire period.

50 GWs are positioned in a way such that they reflect a realistic deployment strategy as it would be designed by a LoRaWAN operator. Fig. 9.1 presents the density of the devices and the position of the GWs throughout the city area. Thus, for each snapshot, a separate simulation was carried out in which for each device present in that snapshot a total of 10 LoRaWAN frames were transmitted. For each frame, the potential reception is monitored, including the number and positions of the GWs that received the frame. Additionally, the power at which each GW receives the frames is extracted, along with the data rate and the channel/frequency used for the frame transmission.

On the other hand, the demo introduces and demonstrates two new mechanisms for E2L that allows to seamlessly process data produced by Edge2 End Device (E2ED) by a E2GW even when the E2ED is not within the coverage of the E2GW due to mobility. The first mechanism forwards frames to the associated E2GW in the case when the E2ED is not within the coverage, while the second mechanism allows the handover of an E2ED from one E2GW to another based on the current network conditions and application-level parameters to optimize the available network, processing and storage resources. The latter mechanism introduces an automatic traffic flow management system. It is based on real-time monitoring of resources in terms of radio coverage and computational capacity, enabling the redirection of traffic flow from one GW to another without the need to pass through the central server.

To evaluate the scalability of the proposed extensions of E2L in large-scale urban scenarios, the LoRaMC Dataset is used, demonstrating how to perform edge processing enabling the E2GW to execute stream-based big data queries. Thus the traffic is automatically processed by different E2GW (edge nodes) as the E2ED change location within the city. The demo demonstrates the dynamic and adaptable nature of the system, thus, presenting the forwarding mechanism, and the load balancing reassignment, including the transition hand-over from one E2GW to another.

Publication & Awards

During my PhD program, I have made the following scientific contributions:

- Ivan Fardin, Stefano Milani, Francesca Cuomo, and Ioannis Chatzigiannakis. 2022. Enabling Edge Computing over LoRaWAN: A Device-Gateway Coordination Protocol. In Proceedings of the 12th ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications (DIVANet '22). Association for Computing Machinery, New York, NY, USA, 23–30. `https://doi.org/10.1145/3551662.3560926`

- Stefano Milani, Domenico Garlisi, Matteo Di Fraia, Patrizio Pisani, and Ioannis Chatzigiannakis. 2023. Enabling Edge processing on LoRaWAN architecture. In Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23). Association for Computing Machinery, New York, NY, USA, Article 103, 1–3. `https://doi.org/10.1145/3570361.3614074`

- Stefano Milani, Domenico Garlisi, Carlo Carugno, Christian Tedesco, Ioannis Chatzigiannakis. 2024. Edge2LoRa: Enabling edge computing on long-range wide-area Internet of Things. Internet of Things, Volume 27, 101266, ISSN 2542-6605. https://doi.org/10.1016/j.iot.2024.101266. **Awarded the Internet of Things Editor's Choice Awards - July 2024.** `https://www.sciencedirect.com/journal/internet-of-things/about/editors-choice#editor-s-choice-awards-july-2024`

- Domenico Garlisi, Stefano Milani, Christian Tedesco, Ioannis Chatzigiannakis. 2024. Achieving Processing Balance in LoRaWAN Using Multiple Edge Gateways. Springer Nature "Lecture Notes in Computer Science". **Awarded the ALGOCLOUD 2024 Best Paper Award**. *In course of publication.*

- Lorenzo Frangella, Stefano Milani, Domenico Garlisi, Ioannis Chatzigiannakis. 2024. Poster: LoRa Mobility and Coverage Dataset (LoRaMC). 2024.

21st International Conference on Embedded Wireless Systems and Networks (EWSN'24). *Accepted, to be presented at the conference.*

- Lorenzo Frangella, Stefano Milani, Domenico Garlisi, Ioannis Chatzigiannakis. 2024. Demo: Enhancing LoRaWAN Networks with Edge Computing: A Demonstration on a Large-Scale Scenario. 2024. 21st International Conference on Embedded Wireless Systems and Networks (EWSN'24). *Accepted, to be presented at the conference.*

Appendix

This Appendix provides a possible implementation of a E2GW selection algorithm within the application server. Alg. 5 dynamically select the optimal E2GW within the infrastructure, considering factors such as load balancing and proximity to the end E2ED. The experiment measured various performance metrics, including handover time, processing time, and frame forwarding latency, to evaluate the effectiveness of the dynamic processing assignment mechanism. In addition, the system monitored the handover process and the corresponding assignment changes in response to device mobility. Following the pseudo-code of the algorithm implemented along with comments explaining each step.

Algorithm explanation: the algorithm begins with the initialization of the system (Step 1), setting up the E2GWs and their processing capacities. It then proceeds to monitor the system resources (Step 2) and the number of received frames (Step 3).

To optimize GW assignments, the algorithm calculates the average processing time per frame (Step 4) based on historical data. The threshold values for system resources and the number of frames received are established (Step 5) to determine when dynamic E2GW assignment is needed.

If any of the system resources exceed their threshold values or the number of frames received exceeds the predefined threshold, the algorithm identifies E2GW with the lowest processing load (Step 6a). Calculate the processing load of the identified E2GW (Step 6b) and check if it has sufficient capacity to handle additional frames (Step 6c). If so, the responsibility for processing the new frames is assigned to the identified E2GW, and its processing load is updated accordingly.

If the identified E2GW is already overloaded and cannot handle additional frames, the algorithm selects a E2GW with a lower processing load as the responsible E2GW for the new frames (Step 6d). The processing load of the newly assigned E2GW is updated to reflect the additional frames it is processing.

The algorithm continues to monitor system resources and the number of received frames (Step 7). If the system resources or the number of received frames decrease below their respective threshold values, the E2GW assignments are reevaluated (Step 8). If necessary, E2GW responsibilities are reassigned to optimize the processing load distribution.

Algorithm 5 Dynamic GW Assignment based on System Resource Monitoring

```
// Step 1: Initialization
InitializeSystem()
// Step 2: Monitor system resources
MonitorSystemResources()
// Step 3: Monitor number of received frames
MonitorReceivedFrames()
// Step 4: Calculate average processing time per frame
CalculateAverageProcessingTime()
// Step 5: Set threshold values for system resources and received frames
SetThresholdValues()
// Step 6: Check if dynamic GW assignment is needed
if ResourcesExceededThreshold() OR FramesExceededThreshold() then
    // Step 6a: Identify GW with lowest processing load
    GW = IdentifyGWWithLowestLoad()
    // Step 6b: Calculate processing load of identified GW
    ProcessingLoad = CalculateProcessingLoad(GW)
    // Step 6c: Assign responsibility to GW if it has capacity
    if GWHasCapacity(ProcessingLoad) then
        AssignResponsibility(GW)
        UpdateProcessingLoad(GW)
    else
        // Step 6d: Select GW with lower load as responsible GW
        NewGW = SelectGWWithLowerLoad()
        UpdateProcessingLoad(NewGW)
    end if
end if
// Step 7: Continue monitoring system resources and received frames
ContinueMonitoring()
// Step 8: Reevaluate GW assignments if resources or frames decrease
if ResourcesBelowThreshold() OR FramesBelowThreshold() then
    ReevaluateAssignments()
    if ReassignmentNeeded() then
        ReassignGWResponsibilities()
    end if
end if
```

Bibliography

[1] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the limits of lorawan. *IEEE Communications magazine*, 55:34–40, 2017.

[2] Ram Ratan Ahirwal and Manoj Ahke. Elliptic curve diffie-hellman key exchange algorithm for securing hypertext information on wide area network. *International Journal of Computer Science and Information Technologies*, 4:363–368, 2013.

[3] Orestis Akrivopoulos, Na Zhu, Dimitrios Amaxilatis, Christos Tselios, Aris Anagnostopoulos, and Ioannis Chatzigiannakis. A fog computing-oriented, highly scalable iot framework for monitoring public educational buildings. In *2018 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2018.

[4] LoRa Alliance. Lorawan 1.1 specification. *technical specification*, 2017.

[5] LoRa Alliance. Lorawan 1.0.4 specification. `https://lora-alliance.org/wp-content/uploads/2020/11/LoRaWAN-1.0.4-Specification-Package\_0.zip`, 2020.

[6] Dimitrios Amaxilatis, Ioannis Chatzigiannakis, Christos Tselios, Nikolaos Tsironis, Nikos Niakas, and Simos Papadogeorgos. A smart water metering deployment based on the fog computing paradigm. *Applied Sciences*, 10(6), 2020.

[7] Apache flink. `https://flink.apache.org/`.

[8] Apache spark. `https://spark.apache.org/`.

[9] Apache storm. `https://storm.apache.org/`.

[10] Emekcan Aras, Gowri Sankar Ramachandran, Piers Lawrence, and Danny Hughes. Exploring the security vulnerabilities of lora. In *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*, pages 1–6, 2017.

[11] Roshan Bharath Das, Gabriele Di Bernardo, and Henri Bal. Large Scale Stream Analytics Using a Resource-Constrained Edge. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 135–139, July 2018.

[12] Laksh Bhatia, Po-Yu Chen, Michael Breza, Cong Zhao, and Julie A McCann. Ironwan: Increasing reliability of overlapping networks in lorawan. *IEEE Internet of Things Journal*, 2021.

[13] Lorenzo Bracciale, Marco Bonola, Pierpaolo Loreti, Giuseppe Bianchi, Raul Amici, and Antonello Rabuffi. Crawdad roma/taxi, 2022.

[14] Ismail Butun, Nuno Pereira, and Mikael Gidlund. Analysis of lorawan v1.1 security: research paper. In *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects*, SMARTOBJECTS '18, New York, NY, USA, 2018. Association for Computing Machinery.

[15] Fotios Chantzis, Evangel Deirme, Ioannis Stais, Paulino Calderon, and Beau Woods. *Practical IoT hacking the definitive guide to attacking the internet of things.* No Starch Press, Inc, 2020.

[16] Xiaoming Chen, Derrick Wing Kwan Ng, Wei Yu, Erik G. Larsson, Naofal Al-Dhahir, and Robert Schober. Massive access for 5g and beyond. *IEEE Journal on Selected Areas in Communications*, 39(3):615–637, 2021.

[17] Florian Laurentiu Coman, Krzysztof Mateusz Malarski, Martin Nordal Petersen, and Sarah Ruepp. Security issues in internet of things: Vulnerability analysis of lorawan, sigfox and nb-iot. In *2019 Global IoT Summit (GIoTS)*, pages 1–6, 2019.

[18] ETSI. Final draft ETSI EN 300 220-1 V2.4.1 (2012-01). Technical Report REN/ERM-TG28-434. `https://www.etsi.org/deliver/etsi\_tr/103500\_103599/103526/01.01.01\_60/tr\_103526v010101p.pdf`, 2012.

[19] ETSI. System reference document (srdoc); technical characteristics for low power wide area networks chirp spread spectrum (lpwan-css) operating in the uhf spectrum below 1 ghz. Technical Report 103 526, ETSI, April 2018. v1.1.1.

[20] Ivan Fardin, Stefano Milani, Francesca Cuomo, and Ioannis Chatzigiannakis. Enabling edge computing over lorawan: A device-gateway coordination protocol. In *Proceedings of the 12th ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, DIVANet '22, page 23–30, New York, NY, USA, 2022. Association for Computing Machinery.

[21] Paula Fraga-Lamas, Mikel Celaya-Echarri, Peio Lopez-Iturri, Luis Castedo, Leyre Azpilicueta, Erik Aguirre, Manuel Suárez-Albela, Francisco Falcone, and Tiago M. Fernández-Caramés. Design and experimental validation of a lorawan fog computing based architecture for iot enabled smart campus applications. *Sensors*, 19(15):3287, July 2019.

[22] Xinwei Fu, Talha Ghaffar, James C. Davis, and Dongyoon Lee. EdgeWise: A better stream processing engine for the edge. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 929–946, Renton, WA, July 2019. USENIX Association.

[23] Domenico Garlisi, Antonino Pagano, Fabrizio Giuliano, Daniele Croce, and Ilenia Tinnirello. A coexistence study of low-power wide-area networks based on lorawan and sigfox. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7, 2023.

[24] Domenico Garlisi, Gabriele Restuccia, Ilenia Tinnirello, Francesca Cuomo, and Ioannis Chatzigiannakis. Leakage detection via edge processing in lorawan-based smart water distribution networks. In *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, pages 223–230. IEEE, December 2022.

[25] Domenico Garlisi, Ilenia Tinnirello, Giuseppe Bianchi, and Francesca Cuomo. Capture aware sequential waterfilling for lorawan adaptive data rate. *IEEE Transactions on Wireless Communications*, 20(3):2019–2033, 2021.

[26] Rémi Géraud-Stewart, Marius Lombard-Platet, and David Naccache. Approaching optimal duplicate detection in a sliding window. In *Computing and Combinatorics: 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29–31, 2020, Proceedings 26*, pages 64–84. Springer, 2020.

[27] Cenk Gündoğan, Christian Amsüss, Thomas C Schmidt, and Matthias Wählisch. Iot content object security with oscore and ndn: a first experimental comparison. In *2020 IFIP Networking Conference (Networking)*, pages 19–27. IEEE, 2020.

[28] Hampel filter. `https://medium.com/\@miguel.otero.pedrido.1993/hampel-filter-with-python-17db1d265375`.

[29] Jialuo Han and Jidong Wang. An enhanced key management scheme for lorawan. *Cryptography*, 2:34, 2018.

[30] Jetmir Haxhibeqiri, Eli De Poorter, Ingrid Moerman, and Jeroen Hoebeke. A survey of lorawan for iot: From technology to application. *Sensors*, 18(11), 2018.

[31] Lu Hou, Kan Zheng, Zhiming Liu, Xiaojun Xu, and Tao Wu. Design and prototype implementation of a blockchain-enabled lora system with edge computing. *IEEE Internet of Things Journal*, 8(4):2419–2430, 2021.

[32] Thomas Janssen, Noori BniLam, Michiel Aernouts, Rafael Berkvens, and Maarten Weyn. Lora 2.4 ghz communication link and range. *Sensors*, 20(16), 2020.

[33] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud. A survey on scalable lorawan for massive iot: Recent advances, potentials, and challenges. *IEEE Communications Surveys & Tutorials*, 25(3):1841–1876, 2023.

[34] kmackay. micro-ecc. https://github.com/kmackay/micro-ecc, 2020.

[35] Taiwo Kolajo, Olawande Daramola, and Ayodele Adebiyi. Big data stream analysis: a systematic literature review. *Journal of Big Data*, 6(1):47, 2019.

[36] K. Lauter. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless Communications*, 11:62–67, 2 2004.

[37] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, October 2017.

[38] Zhiming Liu, Qihao Zhou, Lu Hou, Rongtao Xu, and Kan Zheng. Design and implementation on a lora system with edge computing. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2020.

[39] Jaco M. Marais, Reza Malekian, and Adnan M. Abu-Mahfouz. Lora and lorawan testbeds: A review. In *2017 IEEE AFRICON*, pages 1496–1501, 2017.

[40] David McGrew, Kevin Igoe, and Margaret Salter. Fundamental elliptic curve cryptography algorithms. *Internet Engineering Task Force RFC*, 6090:1–34, 2011.

[41] Stefano Milani and Ioannis Chatzigiannakis. Design, analysis, and experimental evaluation of a new secure rejoin mechanism for lorawan using elliptic-curve cryptography. *Journal of Sensor and Actuator Networks*, 10:36, 6 2021.

[42] Stefano Milani, Domenico Garlisi, Carlo Carugno, Christian Tedesco, and Ioannis Chatzigiannakis. Edge2lora: Enabling edge computing on long-range wide-area internet of things. *Internet of Things*, 27:101266, 2024.

[43] Stefano Milani, Domenico Garlisi, Matteo Di Fraia, Patrizio Pisani, and Ioannis Chatzigiannakis. Enabling edge processing on lorawan architecture. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, page 1–3, Madrid Spain, October 2023. ACM.

[44] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 20:416–464, 1 2018.

[45] Kiyoshy Nakamura, Pietro Manzoni, Alessandro Redondi, Edoardo Longo, Marco Zennaro, Juan-Carlos Cano, and Carlos T. Calafate. A lora-based protocol for connecting iot edge computing nodes to provide small-data-based services. *Digital Communications and Networks*, 8(3):257–266, 2022.

[46] Jorge Navarro-Ortiz, Sandra Sendra, Pablo Ameigeiras, and Juan M. Lopez-Soler. Integration of lorawan and 4g/5g for the industrial internet of things. *IEEE Communications Magazine*, 56(2):60–67, 2018.

[47] Nebulastream. `https://nebula.stream/`.

[48] Moises Nunez Ochoa, Luiz Suraty, Mickael Maman, and Andrzej Duda. Large scale lora networks: From homogeneous to heterogeneous deployments. *International Conference on Wireless and Mobile Computing, Networking and Communications*, 2018-October:192–199, 12 2018.

[49] Antonino Pagano, Daniele Croce, Ilenia Tinnirello, and Gianpaolo Vitale. A survey on lora for smart agriculture: Current trends and future perspectives. *IEEE Internet of Things Journal*, 10(4):3664–3679, 2023.

[50] Wajid Rafique, Lianyong Qi, Ibrar Yaqoob, Muhammad Imran, Raihan Ur Rasool, and Wanchun Dou. Complementing IoT Services Through Software Defined Networking and Edge Computing: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*, 22(3):1761–1804, 2020.

[51] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials*, 19(2):855–873, 2017.

[52] David Reinsel, John Gantz, and John Rydning. Data age 2025: the digitization of the world from edge to core. *Seagate*, 16, 2018.

[53] Eduard Gibert Renart, Javier Diaz-Montes, and Manish Parashar. Data-Driven Stream Processing at the Edge. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 31–40, May 2017.

[54] RIOT-OS. Riot os crypto module. `https://api.riot-os.org/group\_\_sys\_\_crypto.html`.

[55] RIOT-OS. Riot-os. `https://github.com/RIOT-OS/RIOT`, 2020.

[56] Michel Rottleuthner, Thomas C Schmidt, and Matthias Wählisch. Eco: A hardware-software co-design for in situ power measurement on low-end iot systems. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, pages 22–28, 2019.

[57] Ramon Sanchez-Iborra, Jesús Sánchez-Gómez, Salvador Pérez, Pedro J Fernández, José Santa, José L Hernández-Ramos, and Antonio F Skarmeta. Enhancing lorawan security through a lightweight and authenticated key management approach. *Sensors*, 18:1833, 2018.

[58] V. K. Sarker, J. Peña Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund. A Survey on LoRa for IoT: Integrating Edge Computing. In *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 295–300, June 2019.

[59] Yuya Sasaki. We do not have systems for analysing iot big-data. In *CIDR*, 2020.

[60] Yuya Sasaki. A survey on iot big data analytic systems: Current and future. *IEEE Internet of Things Journal*, 9(2):1024–1036, 2022.

[61] Barak Shani. On the bit security of elliptic curve diffie-hellman. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10174 LNCS:361–387, 2017.

[62] Zehua Sun, Huanqi Yang, Kai Liu, Zhimeng Yin, Zhenjiang Li, and Weitao Xu. Recent advances in lora: A comprehensive survey. *ACM Trans. Sen. Netw.*, 18(4), nov 2022.

[63] Maria Xekalaki, Juan Fumero, Athanasios Stratikopoulos, Katerina Doka, Christos Katsakioris, Constantinos Bitsakos, Nectarios Koziris, and Christos Kotselidis. Enabling transparent acceleration of big data frameworks using heterogeneous hardware. *Proc. VLDB Endow.*, 15(13):3869–3882, sep 2022.

[64] Xueying Yang. Lorawan: vulnerability analysis and practical exploitation. *Delft University of Technology. Master of Science*, 2017.

[65] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA, 2010. USENIX Association.

[66] Steffen Zeuch, Ankit Chaudhary, Bonaventura Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. The nebulastream platform: Data and application management for the internet of things. In *Conference on Innovative Data Systems Research (CIDR)*, 2020.

[67] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 5 2010.

[68] Qihao Zhou, Kan Zheng, Lu Hou, Jinyu Xing, and Rongtao Xu. Design and implementation of open lora for iot. *IEEE Access*, 7:100649–100657, 2019.

[69] Michele Zorzi, Alexander Gluhak, Sebastian Lange, and Alessandro Bassi. From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view. *IEEE Wireless Communications*, 17(6):44–51, December 2010.