Seamless Remote Roaming Activation in LoRaWAN via an API-driven Gateway Bridge Service

Luca D'Agati[a,c], Laura García[b], Rafael Asorey-Cacheda[b], Marco Garofalo[a,c], Francesco Longo[a,c], Antonio-Javier Garcia-Sanchez[b], Joan Garcia-Haro[b], Antonio Puliafito[a,c], Giovanni Merlino[a,c]

[a] *Department of Engineering, University of Messina, Contrada di Dio, 98158, Messina, Italy*
[b] *Department of Information and Communication Technologies, Universidad Politécnica de Cartagena, Plaza Cronista Isidro Valverde, 30202, Cartagena, Spain*
[c] *CINI: National Interuniversity Consortium for Informatics, Rome, Italy*

Abstract

The rapid expansion of the Internet of Things (IoT) landscape, notably in smart cities, smart metering, environmental monitoring, and smart agriculture, highlights the critical need for seamless and efficient device roaming within Long Range Wide Area Network (LoRaWAN) infrastructures. Although LoRaWAN shows great potential, its deployment encounters significant challenges, especially under the constraints of version 1.0 specifications, in enabling efficient device mobility. This study introduces a comprehensive approach to enhance LoRaWAN-supported mobility, leveraging three

Email addresses: luca.dagati@studenti.unime.it (Luca D'Agati), laura.garcia@upct.es (Laura García), rafael.asorey@upct.es (Rafael Asorey-Cacheda), marco.garofalo@studenti.unime.it (Marco Garofalo), flongo@unime.it (Francesco Longo), antoniojavier.garcia@upct.es (Antonio-Javier Garcia-Sanchez), joang.haro@upct.es (Joan Garcia-Haro), apuliafito@unime.it (Antonio Puliafito), gmerlino@unime.it (Giovanni Merlino)

URL: http://orcid.org/0000-0002-9992-4179 (Luca D'Agati), http://orcid.org/0000-0003-2902-5757 (Laura García), http://orcid.org/0000-0003-0722-4181 (Rafael Asorey-Cacheda), https://orcid.org/0000-0001-5580-1081 (Marco Garofalo), http://orcid.org/0000-0001-6299-140X (Francesco Longo), http://orcid.org/0000-0001-5095-3035 (Antonio-Javier Garcia-Sanchez), http://orcid.org/0000-0003-0741-7530 (Joan Garcia-Haro), http://orcid.org/0000-0003-0385-2711 (Antonio Puliafito), http://orcid.org/0000-0002-1469-7860 (Giovanni Merlino)

Preprint submitted to Journal of Network and Computer Applications *June 9, 2024*

key advancements: the decoupling of Gateways (GWs) from core network infrastructures to facilitate flexible deployments, the adoption of an adaptive GW-to-network connection protocol responsive to real-time traffic demands, and the enhancement of the GW interoperability to provide dynamic and efficient network configurations. A notable innovation of this research is developing a mechanism that enables packet forwarding through unidentified GW, eliminating the need for pre-established network agreements. The paper's empirical evaluations detail the architecture's effectiveness in IoT applications, showcasing significant enhancements in LoRaWAN's roaming capabilities. This contribution addresses pivotal challenges within the current LoRaWAN networks and sets the background for future advancements in IoT network technologies.

Keywords: Roaming, LoRaWAN, server-independent, API-driven, Internet of Things

1. Introduction

In the fast-evolving domain of Internet of Things (IoT) technology, Long Range Wide Area Network (LoRaWAN) emerges as a central actor, highlighted by its ability to enable transformative impact across various domains such as smart cities, smart metering, environment monitoring, and smart agriculture [1]. Despite its promising capabilities, LoRaWAN faces a significant challenge in enabling seamless and efficient device roaming [2], an essential feature for its competitive edge against 5G networks and its independence from traditional telecommunications infrastructure. This challenge is heightened in deployments constrained by budgets, such as grassroots initiatives and phased projects, where the need for robust roaming solutions becomes even more critical [3, 4].

The evolution of LoRaWAN is marked by introducing two specification versions: 1.0 [5] and 1.1 [6]. The latter version was designed to mitigate some of the limitations of its predecessor, notably by offering enhanced security features and introducing preliminary support for roaming [7]. However, in its initial iterations, LoRaWAN version 1.0 did not define or give robust support to roaming functionalities. This was subsequently addressed in version 1.1, facilitating seamless communication across different network regions, enhancing the mobility of LoRaWAN devices. Despite these advancements, version 1.1 has encountered design limitations that had impeded its certification by

the LoRa Alliance until 2023 [8], thus limiting its widespread adoption.

The prevailing market ecosystem compounds this issue, with many devices supporting only version 1.0 [9]. This compatibility mismatch requires that networks default to version 1.0 specifications, even if a single device in the network lacks support for the newer version. Consequently, this restricts the progress in LoRaWAN deployments to the capabilities of version 1.0, despite the advancements intended by the newer specifications.

To address these challenges, this study proposes a three-way strategy to substantially enhance LoRaWAN-supported mobility within the version 1.0 framework. Initially, the research advocates for the conceptual and operational decoupling of LoRaWAN Gateways (GWs) from the core network infrastructure, thereby introducing greater flexibility in deployment and operational strategies. Subsequently, an adaptive GW-to-Network connection protocol is proposed, predicated on real-time network traffic dynamics, to augment efficiency, particularly in device mobility scenarios. Lastly, the study introduces a dynamic scheme that facilitates roaming capabilities, thereby preventing the conventional requirement for pre-established agreements, often intended as a limitation to agile deployment and scalable network growth.

An innovation introduced by this research is the strategic modification of the GW Bridge service, enabling the activation and management of roaming capabilities remotely through API calls. This approach allows for dynamic adaptation to changing network conditions and user needs, effectively bypassing traditional constraints associated with device roaming in LoRaWAN networks. By leveraging API calls for the precise control and activation of roaming functionalities, this revised service framework enhances the operational flexibility and scalability of LoRaWAN deployments, fostering a more robust and user-centric network ecosystem.

This research has conducted a thorough analysis of the ChirpStack architecture components: the GW Bridge, Network Server (NS), and Application Server (AS). Our exploration began with examining the system's role in facilitating communications between physical GWs and the NS, highlighting the operational challenges encountered. Subsequently, we focused on designing and implementing an API-enabled roaming service, removing the need for a specific preset configuration that allows agreements between NSs involved in the roaming process.

A pivotal contribution of this work, with a designated architecture, depicted in Figure 2, is the seamless integration of the proposed roaming solution within the ChirpStack GW Bridge package. This integration extends the

research proposed in [10] and the GW Bridge's standard functionalities by allowing external interaction capabilities through APIs, thus enabling data roaming from unrecognized GW devices to their respective NS. Additionally, packet decoding functionalities have been implemented to discern the Device Address (DevAddr) and Network Identifier (NetID) of each device, ensuring precise routing of packets to the appropriate NS following API-driven Activation.

Moreover, the deployment of a local database, storing necessary information to keep alive the roaming process, increases the system reliability and ensures uninterrupted communication. This development is instrumental in preserving the integrity and efficacy of the roaming service.

The real-world deployment of the system validates its operational effectiveness and functionality. Furthermore, the system's scalability was extensively tested in an environment densely populated with simulated devices, demonstrating our solution's capacity to extend the functionalities of the existing GW Bridge. This innovative roaming approach mitigates the need for redundant network installations and promotes an efficient roaming model. Extensive testing in simulated environments has confirmed the solution's efficiency and scalability, highlighting its potential for widespread use.

The organization of this paper is structured as follows. After the introduction, Section 2 delves into the existing literature on LoRaWAN roaming. Section 3 offers an in-depth analysis of conventional LoRaWAN roaming methodologies, establishing the basis for the innovative proposal delineated in Section 4. Ensuing sections (Sections 5, 6, 7, and 8) assess the empirical performance of the proposed architecture, both, by using experimental validation for a limited number of devices and simulation to demonstrate device scalability. Finally, our work culminates in a synthesis of findings, implications, and prospective research directions in Section 9. A summary of acronyms used throughout the paper is provided in Table 1, enhancing readability and serving as a quick reference for the reader.

2. Related Work

The exploration and deployment of roaming functionalities within LoRaWAN are central in the progression of low-power, wide-area networks. While roaming in LoRaWAN is emerging as research subject, it is worth noting that the real-world implementation of these functionalities faces multifaceted challenges. Among these challenges is the constraint of centralized

Table 1: List of Acronyms.

Acronyms Used in This Paper	
Abbreviation	Definition
ACL	Access Control List
AS	Application Server
DevAddr	Device Address
DNS	Domain Name System
ED	End Device
fNS	Forwarding Network Server
fSvcProvider	Forwarding Service Provider
fBroker	Forwarding Broker
fBridge	Forwarding Bridge
sBroker	Serving Broker
GW	Gateway
ABP	Activation By Personalization
IoT	Internet of Things
JS	Join Server
LPWAN	Low Power Wide Area Networks
MIC	Message Integrity Code
NwkID	Network ID
NS	Network Server
SLA	Service Level Agreement
sNS	Serving Network Server
sSvcProvider	Serving Service Provider

architectures, as exemplified by conventional solutions like The Things Network (TTN), which primarily favor large organizations at the expense of smaller or experimental networks. This bias arises because Network IDs (NwkIDs), critical for Handover Roaming, are generally accessible only to entities with significant financial backing. Therefore, smaller networks often find themselves relegated to using generic NwkIDs, thus depriving them of the benefits of full-fledged roaming functionalities.

Within the academic community, LoRaWAN roaming has garnered escalating interest since around 2018, peaking in research output in 2020. Many approaches have been provided to tackle the inherent challenges of LoRaWAN roaming, many of which are innovative yet require further evaluation for commercial scalability.

One such inventive approach is the IoTRoam framework developed by Sandoche Balakrichenan et al. [11]. This framework proposes a federated identifier system akin to Electronic Product Code (EPC) identifiers. IoTRoam aims to amplify the scalability of LoRaWAN networks and simplify the roaming process by integrating these identifiers. Although the feasibility of IoTRoam has been proven through proof-of-concept tests, the transition to commercial deployments remains intricate [12]. Further, the same group of researchers extended their work by embedding PKI security into a LoRaWAN roaming architecture, additionally incorporating Domain Name System (DNS) functionalities [12]. The enhancement offered minimal latency overhead, thereby retaining network efficiency.

Building upon the IoTRoam framework, Arnol Lemogue et al. [13] made additional contributions by adding a private DNS over HTTP (DoH) server, termed DNS Broker, into the architecture [14]. However, performance evaluations indicated that the introduction of HTTP-based solutions compromised performance metrics, possibly due to the resource-intensive nature of HTTP. The exploration into DNS over Constrained Application Protocol (CoAP) could be a promising avenue for future research.

Simultaneously, there have been attempts in providing alternative perspectives. Although limited in its reach, the reduced number of available literature regarding roaming in LoRaWAN makes worth noting works such as that of Tadas Lisauskas et al. [15], who veered away from federated identifiers and DNS solutions, introducing a Distribution Server (DS) model that exploits NwkIDs to route messages across various NSs intelligently. This approach shows promising results for national scaling and performed well in simulated DS RAM and CPU utilization tests. In parallel, Stephane Delbruel et al. [16] initiated a framework named FLIP, specifically targeting the efficient management of LoRaWAN GWs, and proved its efficacy across multiple cities.

Moreover, Mohammed Hamnache et al. engaged in an in-depth analysis of existing Handover Roaming specifications and proposed architectural changes aimed at improving network subscription and mechanisms for context migration and DNS resolution [17] [18]. They implemented DNS-based extensions for Chirpstack and evaluated their solutions in actual and simulated environments, observing minimal impact on latency.

There is also an evident inclination toward adopting diverse techniques such as Static Context Header Compression (SCHC) [19], Blockchain-based solutions [20], and Artificial Intelligence (AI) [21]. For instance, Wael Ayoub

et al. suggested using SCHC and its mobile extension (MSCHC) to achieve reduced latency and improved bandwidth [22]. Arnaud Durand presented Blockchain-supported solutions to manage JoinEUI keys autonomously, providing an additional layer of security and enabling implicit roaming agreements [20]. Finally, AI and Machine Learning (ML) techniques have been considered for deployment in vehicle-based shipment tracking solutions [21], albeit still in a preliminary phase.

The domain of LoRaWAN roaming is characterized by different approaches, each with unique advantages and limitations. Particularly, most of the related works have unusual designs for their scenarios and do not present a feasible and reliable solution that can be adopted by the currently deployed LoRaWAN networks. As this paper seeks to extend the discussion, it is essential to acknowledge these myriad works for setting the stage for further innovations in the LoRaWAN roaming landscape. Our work builds upon these foundational studies, incorporating simulation-based and experimental validation [10].

3. A Comprehensive Investigation of Roaming Techniques in LoRaWAN

As a fundamental aspect of IoT communications, LoRaWAN roaming is vital for ensuring smooth device mobility across different network coverages. However, existing roaming protocols pose challenges and have limitations, as acknowledged by the LoRa Alliance [23]. This section assesses passive and handover roaming critically—the two dominant roaming mechanisms—within the overarching structure of LoRaWAN.

3.1. Passive Roaming

Passive roaming relies on a dual-layer control architecture that involves two specialized types of NSs: the Serving NS (sNS) and the Forwarding Network Server (fNS). End-Devices (EDs) primarily communicate with an sNS, even as their data packets may travel via a GW governed by an fNS. This fork creates a rich yet intricate operational landscape.

3.1.1. The Duality of Stateful and Stateless Passive Roaming

The passive roaming framework is subdivided into two key forms:

1. **Stateful Roaming**: In this scheme, the fNS retains a dynamic context for each ED, streamlining packet processing for repeated interactions from that device.
2. **Stateless Roaming**: Here, each packet is handled individually, devoid of any retained context. This minimizes computational burden but sacrifices functionalities like packet sequencing or data recovery.

The LoRa Alliance has remained vague about the protocols governing fNS discovery by roaming partners, leaving room for proprietary mechanisms that could limit universal roaming solutions.

3.2. Handover Roaming

Handover roaming allows for the seamless MAC layer control shift between different NSs. Here, the sNS assumes comprehensive authority over the control and data planes, often in collaboration with a Join Server (JS) and an AS. Despite its holistic control model, its practical deployment has been locked by limitations in the LoRaWAN 1.1 specifications and the absence of formal LoRa Alliance certification.

3.3. LoRaWAN's Roaming Limitations

Originally designed for expansive, first-generation public networks, LoRaWAN's roaming protocols now face the complexities of a landscape marked by public, private, and hybrid networks, among others. The following limitations are particularly noteworthy in this diverse context:

3.3.1. Security and Trust Gaps

Roaming in LoRaWAN inherently assumes a level of trust in participating partners and hubs, opening vulnerabilities to security threats like data leaks and middle-man attacks. Moreover, more than the encryption mechanisms specific to LoRaWAN is required to mitigate these risks appropriately.

3.3.2. Administrative Complexities

The web of contractual arrangements among network operators poses significant administrative challenges, often slowing down the rapid deployment and scalability of LoRaWAN networks in a dynamic technological environment.

3.3.3. Data Handling Limitations

Current LoRaWAN protocols do not adequately support the segregation of payload and metadata, thereby constraining the network's ability to finely control or tailor data handling, which in turn affects the variety of applications that can be effectively supported.

3.3.4. Resource Constraints

The need for each network participant to manage its own NS can be prohibitive for smaller entities, financially and in terms of required technical expertise.

3.3.5. Centralization Risks

A growing trend towards centralized roaming hubs introduces potential single points of failure, compromising the network's resilience, particularly in large-scale failures or targeted attacks.

3.3.6. Standardization and Compatibility Issues

The absence of a universally accepted roaming protocol raises compatibility issues, leading to a fragmented ecosystem and complicating the interconnection of different networks.

3.3.7. Scalability and Performance

Current roaming protocols may not scale efficiently to meet the demands of expanding and increasingly complex networks, leading to potential performance degradation and elevated latency.

3.3.8. Quality of Service (QoS) Deficiencies

LoRaWAN's roaming protocols currently lack robust QoS mechanisms, leaving no guaranteed data reliability or timeliness, which are critical for specific applications like emergency response or real-time monitoring.

3.4. The Evolution LoRaWAN Roaming

The current LoRaWAN landscape predominantly features two roaming strategies: passive and handover roaming. While passive roaming remains the more practical and widely implemented choice due to its compatibility with LoRaWAN 1.0 and 1.1 standards, handover roaming represents a theoretical yet compelling alternative. As LoRaWAN matures, we can expect significant improvements in its roaming capabilities, designed to overcome

existing challenges. These advancements will likely yield more robust and versatile roaming options better suited for various applications, from smart cities to industrial IoT, smart agriculture, and beyond. It is within this context that our research seeks to contribute. We address some limitations and challenges currently impeding LoRaWAN roaming by introducing innovative methodologies and frameworks. Through empirical studies and theoretical modeling, we strive to advance the current state of the art, offering more effective solutions for seamless device mobility across varied network infrastructures.

3.5. Roaming Interactions

Our proposed solution addresses the limitations of existing LoRaWAN roaming strategies, mainly passive roaming. Unlike passive roaming, our approach eliminates the need for complex interactions between different NS. The focus here is on GW-level services, denoted by GW, responsible for the modification ("mangling") of incoming LoRa packets and their appropriate forwarding to the designated sNS.

Upon the arrival of a LoRa packet at a foreign forwarding Bridge (fBridge), the packet undergoes decoding to extract its DevAddr. This address uniquely identifies the originating ED and facilitates the extraction of the NwkID to resolve NetID related to the packet. These identifiers are critical for routing the packet accurately. To obtain the IP address of the sNS for further communication, a Domain Name System (DNS) query is initiated. This query uses the resolved NetID value as its key parameter.

Once the sNS IP address is identified and cached, the GW moves to the packet transmission phase. This, however, is subject to the existence of an active service level agreement (SLA) between the GW and the serving Broker (sBroker). If such an agreement does not exist or is invalid, an SLA Modification Request (ACLmodReq) is generated. This request updates the permissions and allows communication with the sBroker. In the implementation of our architecture proposal, this is provided via an exposed API, which enables the roaming process as shown in Figure 1. Once the ACLmodReq receives acknowledgment (ACLmodAck), the fBridge subscribes to the sBroker, empowering the Forwarding Broker (fBroker) to forward the manipulated MQTT packets to the sNS.

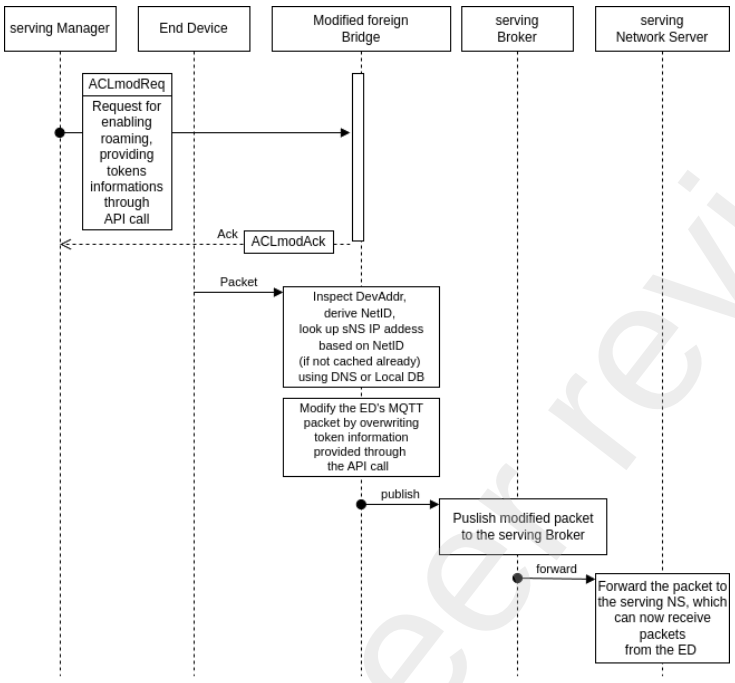

Figure 1: Proposed ABP(Activation By Personalization)-based data packet roaming.

4. LoRaWAN Server-Independent Roaming Framework

This section presents the framework of the proposed solution. Our implementation approach to enable server-independent roaming is detailed in Algorithm 1, which allows LoRaWAN devices to transmit through third-party GWs. This process is crucial for maintaining communication when devices move out of the range of their home network's GWs and encounter GWs from another network.

LoRaWAN version 1.0 specifications introduce limitations that prevent devices from moving between LoRaWAN networks while ensuring seamless communication. Our server-independent roaming approach enables LoRaWAN devices to transmit through third-party GWs as long as they know the required information from the serving network. As shown in Figure 2, the EDs are firstly activated with their respective LoRaWAN Networks through ABP, in this case. This process can be performed manually through the Chirpstack AS's user interface or AS APIs. Once, the EDs are activated, they communicate with the GW or GWs that belong to their network. The GWs include a GW bridge service that manages the incoming messages and

sends them to the MQTT broker. The Chirpstack NS is subscribed to the MQTT topics for each of the GWs in their network and processes the message, publishing it to the corresponding MQTT topic with the ID of each ED, or discarding messages from devices that do not belong to the network. The AS reads the messages processed by the NS and displays the information through its interface. However, if an ED moves, it may fall out of the coverage range of the GWs from its network and may be required to send the data through another network's GW. This roaming functionality is enabled by the GW Bridge, which has been modified to redirect LoRaWAN messages to the MQTT broker of the network they belong to. This process is completely transparent to the Chirpstack Application and Network servers, which ensures complete compatibility with already deployed LoRaWAN 1.0 networks. The GW Bridge detects the GatewayID from the LoRaWAN message and replaces it with the GatewayID corresponding to one of the GWs in the ED's network. Once the message reaches its respective MQTT broker, the NS processes the messages as if it was sent through a GW belonging to the network, and the AS can receive them and display the information on its user interface, not knowing the message had not followed the expected route.

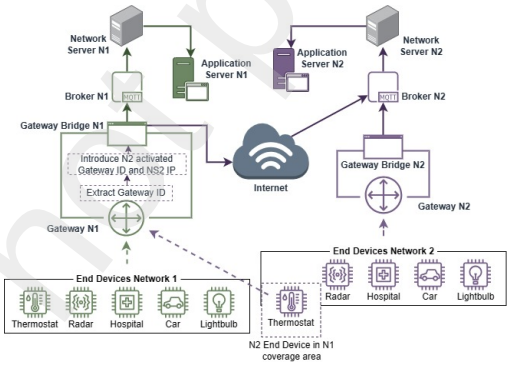

Figure 2: Framework of the LoRaWAN server-independent roaming approach.

5. Validation of Conceptual Framework Through Experimental Testing

This section delineates the specific hardware configurations and network topology employed to ascertain the viability of the proposed LoRaWAN networking model in real-world scenarios.

Algorithm 1 Enhanced MQTT Broker Integration for Roaming

```
Define global variables for MQTT client, server IPs, topics, ports, and local database connection.
procedure LaunchAPI
    Start async listener on a specific port to handle HTTP requests.
    Initialize local database connection for parameter storage.
end procedure
procedure InitializeMQTTClient(brokerAddress)
    if brokerAddress is not specified then
        Log error and exit.
    else
        Setup MQTT client with auto-reconnect and session persistence.
        Attempt to connect to the specified MQTT broker.
    end if
end procedure
procedure HandleHTTPRequest(HTTP request)
    if Request method is POST and request URL is "/configure" then
        Decode request body to extract broker and GW details.
        Save the extracted details into the local database.
        Update global configurations with details from the local database.
        Acknowledge the configuration setup via HTTP response.
        Use details to initiate MQTT subscriptions on topics related to device messages.
    else
        Respond with HTTP 400 error for unsupported methods or URLs.
    end if
end procedure
procedure OnMessageReceived(MQTT message)
    Decode and inspect the message payload for valid JSON format.
    if Payload is valid and contains device roaming data then
        Invoke ModifyPayload with payload and new GatewayID from local DB.
        Publish the modified payload to a specific topic for the NS.
    end if
end procedure
procedure SubscribeToTopic(topic)
    Subscribe MQTT client to a topic with a message handler for incoming messages.
    Log any errors and unsuccessful subscription attempts.
end procedure
function ModifyPayload(payload, newGatewayID)
    Extract DevAddr from the payload.
    Update the payload with the new GatewayID.
    Return the modified payload.
end function
```

5.1. Experimental Infrastructure

Two separate LoRaWAN networks were operationalized to facilitate the tests. Network 1 incorporated devices configured for roaming capabilities, while Network 2 was confined to home-based devices. Both networks employed ChirpStack [24] as the underlying server infrastructure to instantiate the Network and ASss.

5.1.1. Network 1 Server and Gateway Configuration

Network 1 was instantiated on a Raspberry Pi 4 model B [25] equipped with 8 GB of RAM and running a 64-bit Desktop Raspberry Pi OS. Chirpstackv3 Application and NSs as well as the MQTT broker for Network 1 were installed as required. This Raspberry Pi also worked as the LoRaWAN network GW and was augmented with a RAK 5146 module [26] and a Pi-Hat adapter. Therefore, the script and Golang version of the GW Bridge for enabling roaming was deployed in this device.

5.1.2. Network 2 Server Configuration

The Network 2 server was deployed on a virtual machine running Ubuntu 22.04.1, hosted on an Asus TUF laptop endowed with 32 GB of RAM and an i7-11800H processor. This virtual machine also had its respective Chripstackv3 AS, NS, and MQTT broker installed.

5.1.3. Database and End-Device Specifications

To manage the information regarding the roaming devices, a TinyDB [27] NoSQL database was utilized. This database was hosted in the raspberry pi. All the end-devices involved were implemented using Arduino MKR WAN 1310 boards [28], each equipped with an 868 MHz antenna. They were programmed to send LoRaWAN messages at a fixed interval.

5.1.4. Additional Hardware

The Wi-Fi router employed to enable the communication between the raspberry pi and the laptop was a TP-Link AX 3000 model.

5.2. Network Topology and Data Flow

The test network topology deployed for testing the roaming scenarios is outlined in Figure 3. As depicted, every component in the network communicates wirelessly, being LoRa the chosen technology for the IoT devices

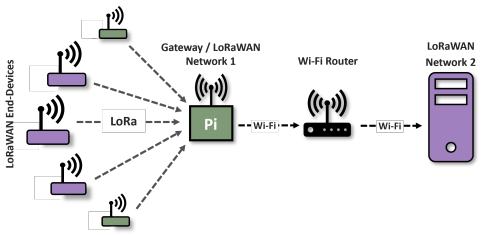

Figure 3: Topology of the Test Network.

and WiFi the technology that enables the communication between both LoRaWAN NSs. Both roaming and non-roaming LoRaWAN devices transmitted their data to the LoRaWAN GW. This GW performs the tasks of both the GW-Bridge and the Chirpstack server for Network 1. Depending on the tests, the script version or the Golang version of the proposed roaming solution is being executed at the GW in a completely transparent manner and without modifying nor disrupting the operation of the Chirpstack GW-Bridge or Server software, which follow LoRaWAN 1.0 specifications. This way, the incoming messages are filtered to determine if they belong to another network, and sent to the network they belong to, which corresponds to Network 2 of our testbed. The roaming messages are then received at Network 2 and handled by the Chirpstack server following its usual operation, in other words, the roaming process is fully transparent to the Chirpstack servers of both networks.

5.3. Parameters for Experimental Tests

The experimental tests were conducted at the laboratory with a variable number of home (non-roaming) and roaming devices (See Figure 4). The tests evaluate in an experimental setup i) if messages from foreign LoRaWAN networks could be redirected to their home network successfully; and ii) if the presence of non-roaming devices hindered the roaming process. The technical specifications of the materials used for the experimental tests are provided in Table 2. The duration for all the experiments was 1 hour. Each LoRa device transmitted messages with 15 Bytes of data using a configuration of SF (Spreading Factor) 7 and code rate 4/5. All tests were performed using message transmission intervals of 30 s. This interval is the shortest delay the LoRa devices could manage between consecutive transmissions. This configuration increases the demands for message handling as much as possible to determine if the proposed roaming solution could perform satisfactorily

in high-demand environments. Common transmission intervals from LoRa networks intended for IoT applications, such as telemetry, range from minutes to hours. Therefore, if our proposal meets the performance needs for the high-demand conditions devised for this testbed, it is expected to perform well in deployments with fewer data transmission requirements.

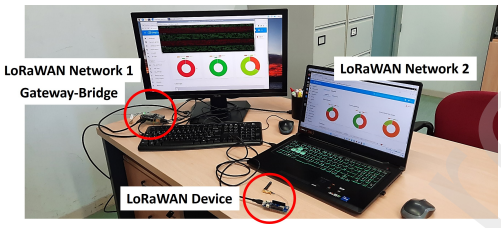

Figure 4: Experimental Setup.

Table 2: Hardware Specifications

Device	Processor	Clock Speed	RAM
Arduino MKR 1310	Arm® Cortex®-M0 32-bit SAMD21	32.768 kHz	SAMD21G18A (32KB SRAM) and Murata CMWX1ZZABZ (20KB RAM)
Raspberry Pi 4 B	64-bit quad-core Cortex-A72	1.5 GHz	8 GB
PC	I7-11800H	2.3 GHz	32 GB

6. Experimental Results

Experimental tests were performed with two versions of the implementation of our proposal. The first version was the script implementation. The script was developed in Python programming language and runs at the Raspberry Pi alongside the official Chripstack GW Bridge software. The second version corresponds to the Golang implementation, where the proposed solution was developed as an add on to the code of the Chirpstack GW Bridge. This way, our proposal is fully integrated and compatible with Chirpstack LoRaWAN networks without needing to manually execute complementary scripts.

6.1. Script Version

The results of the tests performed utilizing the script version of the proposed roaming solution are provided in this subsection.

6.1.1. Test 1: One Roaming Device, Zero Home Network Devices

The first experiment was implemented with a single roaming device, while the home network has no active devices. The latency incurred at the Chirpstack server is illustrated in Figure 5. As it can be seen, the latency for this experiment remained between the values of 271.98 ms and 411.54 ms. Furthermore, the RSSI and SNR of the received messages averaged -19.52 dBm and 12.47 dB respectively (The graphical representations for the RSSI and SNR data gathered throughout all tests are provided in Apendix Appendix A). All messages reached their corresponding NS without packet losses.

6.1.2. Test 2: One Roaming Device, One Home Network Device

The second test added a home network device to increase the messages handled by our roaming solution and determine if any interference was introduced. The latency metrics are displayed in Figure 6. During this test, the latency results varied between 317.06 ms and 527.44 ms, with average RSSI and SNR values of the received messages of -39.27 dBm and 12.84 dB respectively. Therefore, the addition of another device impacted the resulting latency for the roaming device, specifically in one isolated occasion at the end of the test. No packet losses were detected in this test.

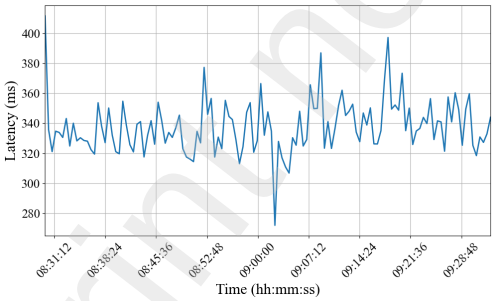

Figure 5: Latency at the Chirpstack Server for the script version of Test 1.

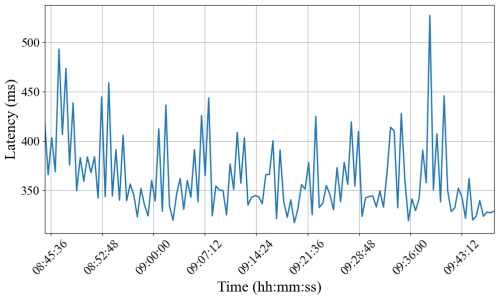

Figure 6: Latency at the Chirpstack Server for the script version of Test 2.

6.1.3. Test 3: One Roaming Device, Two Home Network Devices

In this scenario, the experiment was conducted with one roaming device and two devices present in the home network, increasing the number of handled messages and possible interference. The latency at the Chirpstack server is represented in Figure 7, with a minimum value of 316.05 ms and a maximum value of 1055.16 ms. Although most latency readings remained within the range of previous tests, one message experimented a significant increase in delay nearing the middle time of the test. The average RSSI and SNR resulting from the received messages were -34 dBm and 12.4 dB respectively, not being hindered by the home devices. Furthermore, this test did not present any packet losses.

6.1.4. Test 4: Two Roaming Devices, Two Home Network Devices

Another roaming device was added to the network to asses the correct management of the roaming process with different devices. The latency figure for both roaming devices is plotted in Figure 8. The first device ranged from 264.77 ms to 1503.16 ms, whereas the latency for the second device oscillated between 314.13 ms and 1387.13 ms. These transmissions resulted in an average RSSI of -26.17 dBm and -29.2 dBm for devices 1 and 2, respectively. The average SNR was 12.5 dB for device 1 and 12.51 dB for device 2. The introduction of a second roaming device has not altered the overall latency achieved in previous tests. However, the sporadic cases of messages with substantially higher latency that occurred with one roaming device has also been experienced by the second roaming device; notwithstanding, all messages were correctly delivered.

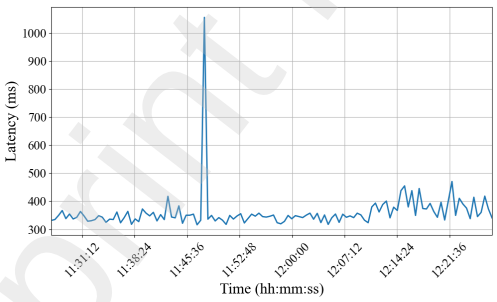

Figure 7: Latency at the Chirpstack Server for the script version of Test 3.

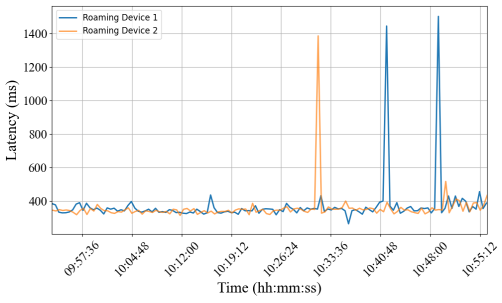

Figure 8: Latency at the Chirpstack Server for the two roaming devices in the script version of Test 4.

6.1.5. Test 5: Three Roaming Devices, Two Home Network Devices

The final test added another roaming device, leading to a total of five LoRaWAN devices being handled by the script. The latency results for the all roaming devices are displayed in Figure 9. It ranged between 284.70 ms and 3610.03 ms for device 1, 326.19 ms and 659.16 ms for device 2, and 295.96 ms and 506.64 ms for device 3. Furthermore, the RSSI and SNR captured from the received messages of all roaming devices averaged -28.78 dBm, -29.2 dBm, and -46.6 dBm regarding the RSSI, and 12.68 dB, 13.01 dB, and 12.64 dB regarding the SNR for devices 1, 2, and 3, respectively. In this test only one roaming device experienced the sporadic peaks of high latency, noting one message with a more severe increase. There also was one lost packet on arrival at the server. That is, the packet reached the GW successfully but was not correctly handled by the server. The rest of the tests presented a similar performance to the previous tests with fewer devices. Therefore, the roaming solution presented in this paper shows signs of good scalability.

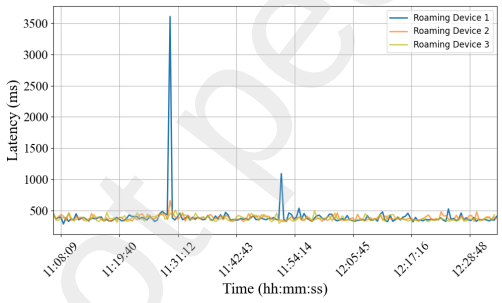

Figure 9: Latency at the Chirpstack Server for the three roaming devices in the script version of Test 5.

6.1.6. Discussion

The series of tests conducted for the script version of the roaming implementation for LoRaWAN elucidated that the proposed architecture facilitates a highly efficient roaming process. Specifically, the latency figures from the end-device to the final LoRaWAN server exhibit an average latency range of 337 to 384 ms (See Table 3), with limited message transmissions with higher latency but successful delivery. Considering the requirements of LoRaWAN networks for common IoT long-range telemetry applications, where there aren't strict real-time constraints, we can conclude that our roaming proposal satisfies the needs of this type of deployment, and thus, its commercial

implementation is feasible.

Table 3: Summary of Experimental Results for the Script Version

Test Number	Roaming Devices	Home Network Devices	Uplink Frequency	Avg. Latency at Server	Lost Packets at GW	Lost Packets at Server
1	1	0	30 s	337.68 ms	0	0
2	1	1	30 s	363.15 ms	0	0
3	1	2	30 s	360.38 ms	0	0
4	2	2	30 s	363.99 ms	0	0
5	3	2	30 s	384.70 ms	0	1

6.2. Golang Version

Mirroring the previous tests, this section presents the performance results for the Golang implementation of our LoRaWAN roaming proposal to ascertain any differences in operation. This version was developed following the format of the current implementation of the Chirpstack software for both GWs and Servers.

6.2.1. Test 1: Single Roaming Device, No Home Network Devices

Test one is performed with exclusively one roaming device. Figure 10 reveals the latency as captured at the Chirpstack server, with values oscillating between 320.71 ms and 476.05 ms. The RSSI and SNR values associated with the reception of these messages resulted in mean values of -43.09 dBm and 12.51 dB respectively. No significant differences in performance were observed compared to the script implementation, except for the baseline latency being moderately higher.

6.2.2. Test 2: Single Roaming Device, One Home Network Device

This test introduced one home device to the network. Figure 11 presents the latency metrics recorded at the Chirpstack server. The values ranged from 276.48 ms to 600.76 ms. In addition, the average RSSI and SNR results derived from this test were -40.96 dBm for the former and 12.81 dB for the latter. As in test 2 of the script version, the introduction of another device has created one sporadic message that experienced higher latency compared to the rest of the transmissions. There were no other observations and all packets were successfully delivered.

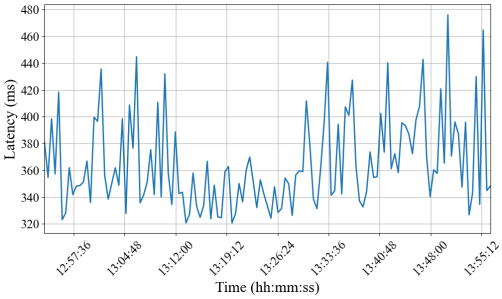

Figure 10: Latency at the Chirpstack Server for a single roaming device with no home network devices in Test 1.

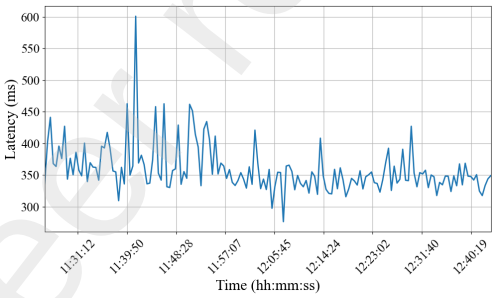

Figure 11: Latency at the Chirpstack Server for a single roaming and a single home network device in Test 2.

6.2.3. Test 3: Single Roaming Device, Two Home Network Devices

The addition of a second home device did not result in noteworthy changes. Figure 12 details the latency as recorded at the Chirpstack server, which ranged from values of 290.19 ms to values of 498.14 ms. Although the latency began to rise at the end of the test, there were no messages with high latency peaks, as opposed to the script version. This fact begins to elucidate that the Golang version is better at handling increasing numbers of messages. The mean RSSI and SNR results were -34.46 dBm and 12.5 dB respectively, showing no impact on link quality from the home devices.

6.2.4. Test 4: Dual Roaming Devices, Two Home Network Devices

This test added one more roaming device to the network. Figure 13 shows the latencies for each roaming device at the Chirpstack server, with average delays reported at 391.36 ms and 389.86 ms for the first and second devices, respectively. The maximum recorded latency for the first roaming device was

498.14 ms, and the lowest value was 322.66 ms. For the case of the second device, the maximum value was 506.25 ms and the minimum was 288.96 ms. The RSSI and SNR results obtained for this test averaged -36.45 dBm and 12.72 dB respectively for device 1, and -29.68 dBm and 12.62 dB respectively for device 2. Although the average latency for the roaming devices in this test is higher than those of the script versions, it is noteworthy that no isolated peaks exceed 1000 ms. Furthermore, there were no lost packets. This shows that the Golang implementation is able to manage different roaming devices successfully as well, without the occurrence of unexpected events.

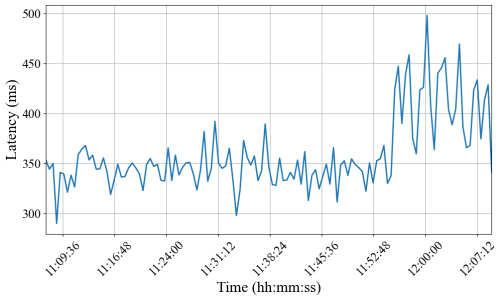

Figure 12: Latency at the Chirpstack Server for a single roaming and two home network devices in Test 3.

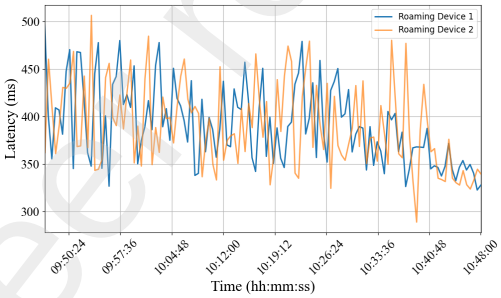

Figure 13: Latency at the Chirpstack Server for the two roaming devices in Test 4.

6.2.5. Test 5: Triple Roaming Devices, Dual Home Network Devices

This final test includes the last roaming device addition. Figure 14 displays the latency of all roaming devices. The corresponding average delays were 351.40 ms, 350.80 ms, and 347.34 ms, respectively. Moreover, the latency ranged from 315.14 ms to 412.88 ms for device 1, 309.83 ms to 471.64 ms for device 2, and 308.65 ms to 419.64 ms for device 3. These last latency readings corroborate the successful implementation of our roaming solution without unexpected delay increases for specific messages. The average latency remains within similar levels compared to one-device network transmissions, demonstrating the capacity of this implementation to handle multiple messages seamlessly. There were however 2 lost packets at the server, but the error was identified between the NS and the AS, and thus, attributed to the Chirpstackv3 server code. Lastly, the mean RSSI and SNR results for the three roaming devices were -31.07 dBm and 12.69 dB for device 1, -22.73 dBm and 12.4 dB for device 2, and -43.09 dBm and 12.64 dB

for device 3, not showing disturbances caused by the increased number of devices within the coverage area of the GW.

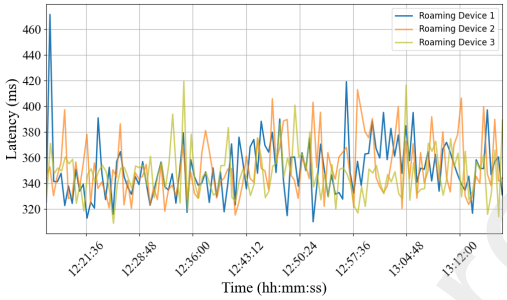

Figure 14: Latency at the Chirpstack Server for the three roaming devices in Test 5.

6.2.6. Discussion

The summary of the tests performed with the Golang implementations, recorded in Table 4, shows the stability of this implementation. The average latency for all devices ranged from 349.85 to 390.61 ms. Although some of the tests presented higher average values than those for the script tests, the observed differences are negligible considering the absence of sporadic latency increases and the usual requirements in transmission delays for common LoRaWAN IoT applications. Thus, we conclude that the Golang version of our proposal does comply with the needs in terms of latency of LoRaWAN networks, and its deployment in existing networks would satisfy current roaming needs that have not been addressed by leading manufacturers.

7. Scalability and Performance Analysis Using LWN-Simulator

The experimental tests for both implementations showed promising results for an extended use of our roaming solutions. However, the limitations in the number of available physical devices restricted the scope of the tests. For this reason, this section presents the simulation setup to analyze the performance of denser networks, including a higher number of roaming and home devices.

Table 4: Summary of Results for Golang Version

Test Number	Roaming Devices	Home Network Devices	Uplink Frequency	Avg. Latency at Server	Lost Packets at GW	Lost Packets at Server
1	1	0	30 s	364.88 ms	0	0
2	1	1	30 s	359.17 ms	0	0
3	1	2	30 s	359.30 ms	0	0
4	2	2	30 s	390.61 ms	0	0
5	3	2	30 s	349.85 ms	0	2

7.1. Simulation Setup

To evaluate the scalability and performance of our proposed LoRaWAN roaming architecture, we employed the *LWN-Simulator*[1], a highly adaptable simulation tool acclaimed for its precise modeling of LoRaWAN network components, specifically EDs and GWs. Our experimental testbed consisted of three Virtual Machines (VMs) operating on Ubuntu 20.04 LTS, each tasked with simulating one of the network components integral to our roaming architecture: *NetworkServer1* and *NetworkServer2*, functioning as home and foreign servers, respectively, and a *GW Bridge*, which was augmented with our GW bridge service enhancements to support efficient roaming operations. Each VM was provisioned with 4 CPUs and 4096 MB of RAM, ensuring robust performance for the simulations.

To mirror a real-world wired network environment within our simulations, we employed the `tc-netem` tool to introduce and manage a network delay, specifically simulating a consistent 50ms delay in network communication. This methodology was necessary for assessing the performance of our roaming solution under realistic network latency conditions. Moreover, the VMs

[1] `https://github.com/UniCT-ARSLab/LWN-Simulator`

were synchronized with an NTP server to maintain accurate timing across the system and be able to evaluate the results. This configuration, emulating practical LoRaWAN deployments with VMs configured with the ChirpStack NS and the LWN-Simulator, along with a third VM running the enhanced GW Bridge, facilitated the examination of our roaming architecture. This methodology allowed for the assessment of the network's reliability and scalability during the adoption of the modified GW bridge service, providing valuable insights into its performance, in the simulating phase of real-world LoRaWAN network scenarios. All tests were performed using demanding message transmission intervals of 30 s.

8. Simulation Outcomes and Detailed Analysis

This section presents an exhaustive analysis of the data derived from our extensive simulations, assessing the network's performance with both roaming and stationary devices. We address the average packet latency between these two device states and the distribution of latencies over time.

8.1. Average Packet Latency Analysis

We examined average packet latency as a function of frame count numbers (*FCnt*) for roaming and stationary states, employing these counts to compute each packet transmitted by the EDs. In the LoRaWAN protocol, Frame Count Numbers (*FCnt*) play a crucial role in maintaining the integrity and security of data transmissions. Each packet sent from an ED is assigned a unique *FCnt* value, which increments with every transmission. This sequential incrementing mechanism is instrumental in thwarting replay attacks, in which an attacker attempts to re-send previously intercepted packets. Figure 15 illustrates the impact of roaming and stationary states on latency (one stationary device and one in roaming), with roaming devices showing an anticipated slight increase in latency due to the additional processing requirements inherent in roaming. Furthermore, this latency increase remains within acceptable parameters for network applications, with an average latency compared to the stationary case of approximately 0.16 s, highlighting, in this specific simulation test, the efficiency of the proposed roaming architecture.

Additionally, Figure 16 explores the distribution of packet latencies over time. This analysis reveals stable latency for roaming devices and consis-

tently lower latency for non-roaming devices, suggesting the network adeptly handles roaming's additional demands.

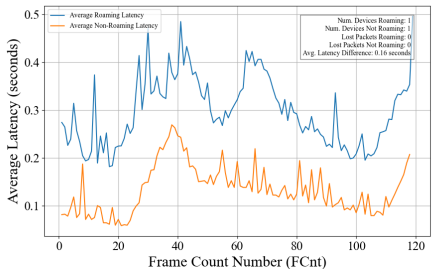

Figure 15: Comparative analysis of packet latency across frame counts for roaming and non-roaming devices, accentuating the added latency induced by roaming yet confirming its suitability for application demands.

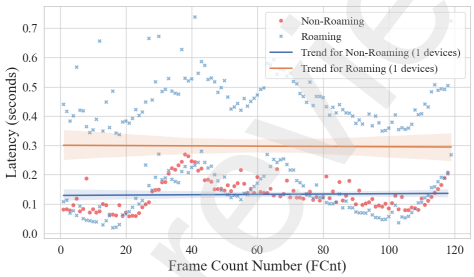

Figure 16: Distribution of packet latencies over time, contrasting the relative consistency in non-roaming devices against the marginally higher latency trajectory observed in roaming devices.

8.2. Roaming Dynamics Analysis

Expanding our analysis, we assessed scenarios with increased roaming activity involving two roaming devices and one stationary device. As depicted in Figure 17, we note a marginal rise in average latency with the introduction of extra roaming devices (compared to the stationary), yet the network demonstrates its capability to maintain service quality even under increasing roaming conditions. Furthermore, the network exhibits improved performance compared to the previous simulation, with maximum latency peaks of 0.45 s (lower than the previous case). Additionally, the average packet latency for roaming devices compared to stationary ones, in terms of packet transmission and reception times, is reduced to 0.12 s.

Figure 18 further confirms the network's proficiency in managing latency, indicating a resilient system that copes with the intricacies of multiple roaming sessions without compromising the quality of service for stationary devices.

8.3. Balanced Device Mobility Analysis

In balanced scenarios with an equal number of roaming and non-roaming devices (two for each), Figure 19 illustrates similar latency patterns for both

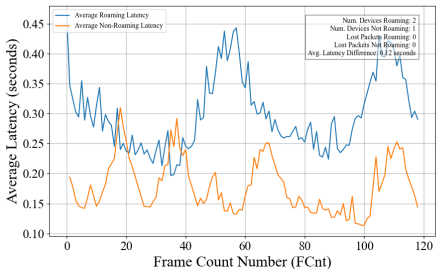

Figure 17: Network performance under concurrent roaming sessions, evidencing minimal impact on service quality despite the increase in roaming devices.

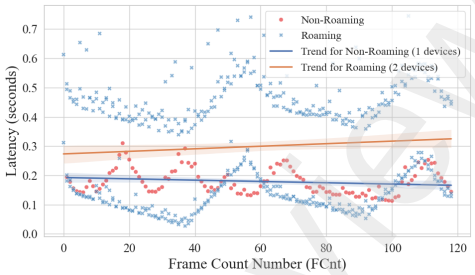

Figure 18: Latency analysis in multi-roaming scenarios, underscoring the network's ability to sustain balance and performance amidst the increased operational demands.

device states, affirming the network's efficiency and service quality. Notably, the maximum latency peaks for roaming devices remain within the same range as the previous case, indicating consistent performance. Furthermore, in this situation, the difference in average packet latency between roaming and non-roaming devices settles at approximately 0.17 s. Despite variations, the network's behavior remains stable as the number of received packets increases.

Figure 20 gives us a finer view of latency management, showcasing the network's capability to maintain service quality even with a balanced distribution of device mobility.

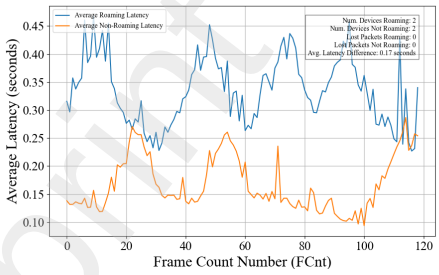

Figure 19: Analysis of network efficiency with an equivalent distribution of roaming and non-roaming devices.

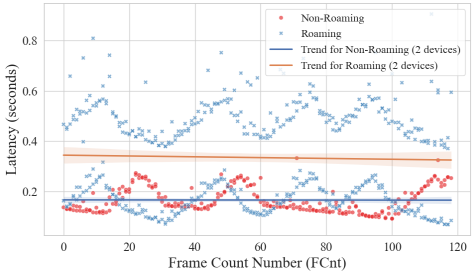

Figure 20: The latency distribution for an equal mix of roaming and non-roaming devices, underscores the network's adeptness at ensuring latency remains within the desirable thresholds for all devices.

In environments where roaming devices predominate, the network's capacity to mitigate heightened latency pressures is evident, as depicted in Figures 21 and 22, ensuring consistent service quality. Specifically, in this scenario with three roaming devices and two stationary devices, we observe a negligible increase in maximum latency, approximately 0.5 s. However, concurrently, there is a slight reduction in the difference in average latency between roaming and stationary packets, approximately 0.15 s. Once again, we observe the network's stability despite variations in the number of devices involved.

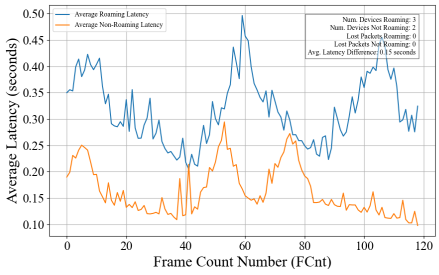

Figure 21: Network latency trends in scenarios dominated by roaming devices, demonstrating the system's strong performance and effective management of increased roaming activities.

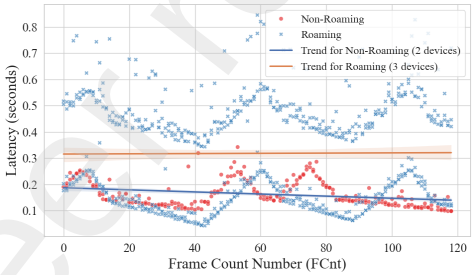

Figure 22: In-depth latency distribution analysis in a network scenario with a majority of roaming devices, highlighting the network's capacity to ensure service continuity and operational stability.

8.4. Analysis of a Balanced Environment

In order to mirror conditions that more accurately reflect real-world operational scenarios, our simulations were expanded to include an environment consisting of an equal distribution of roaming and stationary devices, specifically ten of each category. This approach facilitates a comprehensive evaluation of the network's performance when subjected to a uniform device mobility distribution scenario. The outcome of this simulation is depicted in Figures 23 and 24, which collectively assess the impact of such balanced conditions on network latency.

The simulation results reveal that, under these balanced conditions, the maximum latency observed stabilizes at an approximate value of 0.55 s. Concurrently, an interesting observation is the narrowing of the latency gap, with the differential between the average latencies of packets emanating from

roaming versus stationary devices diminishing by roughly 0.10 s. This reduction is indicative of a comparative decrease in latency disparities observed in prior simulations. It is imperative to highlight that, as in previous simulations, this scenario also resulted in no packet losses. Moreover, the latency trends between roaming and non-roaming devices began to display intermittent convergence, suggesting a homogenization of network performance across device mobilities.

Figure 24 presents a detailed visualization of the latency distribution across devices with varying mobility patterns. The scatter plot distinctly differentiates between the packet latencies of roaming (blue) and non-roaming (red) devices as a function of the Frame Count Number (FCnt). The data points for roaming devices exhibit a broader distribution of latencies, which is expected due to the computation processes introduced. In contrast, the non-roaming devices maintain a more tightly clustered latency range, indicative of the stable connectivity associated with stationary environments.

Trend lines traced through the respective data sets reveal the central tendencies for each device type. Notably, the trend for roaming devices appears relatively flat, suggesting that their latencies are not significantly impacted by the FCnt within the observed range. On the other hand, the non-roaming devices exhibit a slight upward trend, implying a subtle increase in latency as the FCnt progresses.

The overlap of data points between roaming and non-roaming devices, particularly in the lower latency range, indicates that the network is capable of providing comparable service levels to both mobile and stationary users. The convergence of the trend lines towards the higher end of the FCnt spectrum further supports this observation, signifying the network's efficacy in latency management despite the inherent challenges presented by device mobility.

8.5. Scalability Under High Device Load

In the realm of network performance, scalability is a pivotal factor that directly influences the robustness and efficiency of service delivery. Figures 25 and 26 illustrate the network's behavior under two distinct load conditions. The first scenario, depicted in Figure 25, encompasses an environment with fifty roaming and fifty stationary devices. Initially, the roaming devices experienced an escalation in transmission latency, peaking at a maximum latency of 3 s. This peak persisted until the cumulative transmission of approximately 38 packets per device, after which a marked improvement in roaming

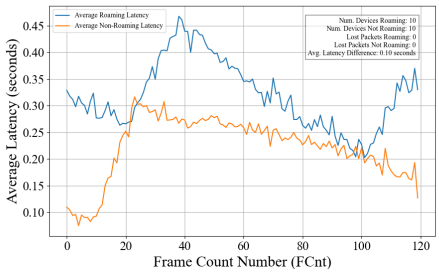

Figure 23: Comparative analysis of average packet latency between ten roaming and ten stationary devices, showcasing the network's capability to sustain stable performance under conditions of balanced device mobility.

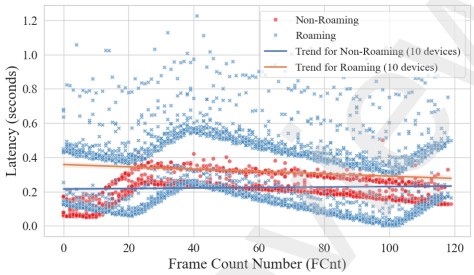

Figure 24: Latency distribution in a balanced environment, elucidating the network's proficiency in managing a mix of mobile and stationary devices, thereby ensuring consistent latency performance.

performance was observed, converging with the average latency trend of the stationary devices. Notably, despite this early increased latency, the overall simulation spanning an hour with an aggregate of 120 packets per device resulted in an average latency deviation of 0.5 s in comparison to stationary devices. Packet loss was observed to be symmetrical, with two packets lost in each category.

In the second scenario, as illustrated in Figure 26, the network was subjected to a more substantial load with one hundred roaming and one hundred stationary devices. Positive finding, in this case, the maximal latency decreased to 1.2 s. Roaming network performance began to align with that of the non-roaming network after the dissemination of approximately 25 packets per device, which led to the average latency difference settling at 0.16 s. This demonstrates that the network tends to optimize performance with increasing device count, thereby exhibiting scalability. However, a disparity was observed in packet loss, with the roaming devices losing 25 packets in contrast to 4 lost by the non-roaming devices when considering a global count of approximately 120 packets per device.

These findings resonate with the research conducted by Mishra et al. [29], where the Mosquitto broker was analyzed under high message throughput conditions. Their investigation into the broker's performance under stress utilized metrics such as CPU utilization, latency, and message throughput. Our system's latency management efficiency under a significant device load reflects their insights, corroborating the notion that the Mosquitto broker's

scalability is, to some extent, attributable to its adept resource management and load handling capabilities, particularly in high-demand scenarios [29].

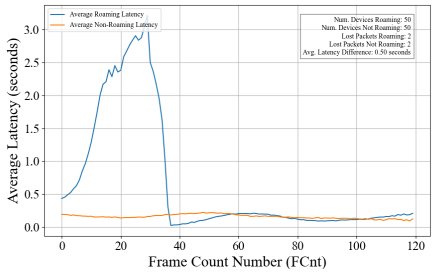

Figure 25: Network performance under heavy load with fifty roaming and fifty non-roaming devices, illustrating maintained low latency and high service quality.

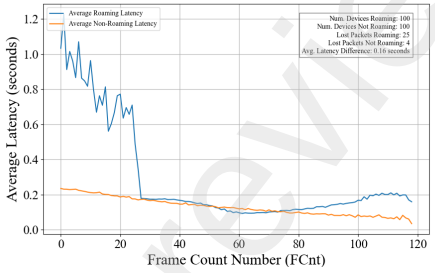

Figure 26: Evaluating network scalability and efficiency in a massive-scale environment, affirming the architecture's capability to support extensive device mobility without service degradation.

Figures 27 and 28 depict the scatter plots of packet latency as a function of the FCnt for both roaming and non-roaming devices. The data points for roaming devices generally present a higher latency, dispersed widely in the initial frames, suggesting variability in the latency likely due to the introduced computation load. As the FCnt increases, the latency values for roaming devices converge towards a trend that stabilizes, which is indicated by the orange trend line, due to the auto-adapting process of the network where the MQTT broker is involved.

Conversely, non-roaming devices exhibit a more concentrated cluster of latency measurements with a noticeably lower variance, as indicated by the blue data points. The trend line for non-roaming devices, shown in red, demonstrates a slight upward trajectory but remains significantly below that of roaming devices, indicating more stable and lower latency readings. Additionally, the overall density of points decreases with increasing FCnt, which may reflect a network (roaming) that effectively handles congestion through efficient packet management and the packets mangling process, thus maintaining service quality even as the demand for network resources grows.

The simulation data validates the network's ability to uniformly manage service quality and latency across varying device roaming and operational scales. Despite the challenges posed by high device loads and diverse roaming activities, the network shows a trend to low-latency communication and

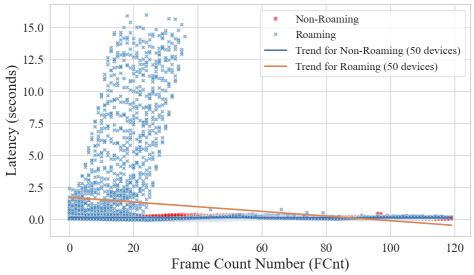

Figure 27: Detailed latency analysis in a high population scenario, demonstrating network reliability and consistent service levels across varied device mobility scenarios.

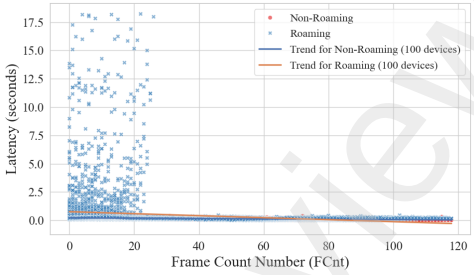

Figure 28: Comprehensive latency distribution analysis in a highly congested environment, highlighting the network's reliability and consistent service levels across a broad spectrum of device mobility scenarios.

exhibits the necessary scalability. The outcomes demonstrate that the proposed LoRaWAN roaming architecture can sustain consistent service levels, affirming its potential for wide-ranging LoRaWAN applications.

9. Conclusion

Our proof-of-concept tests verify that a LoRaWAN roaming solution is feasible and performs adequately in terms of end-to-end delay and packet loss (considering application in the LoRa networks domain). The experimental and simulation results have established the robustness and efficiency of our proposed architecture. The solution's ability to handle high-demand scenarios, maintain low latency, and contain packet loss, even in extensive network setups suggests that the solution is feasible for real-world applications.

In future works, the focus of research will be on enhancing the adaptability of this solution. This includes optimizing the architecture to handle a greater number of devices and varied network topologies. Emphasis will also be placed on improving the QoS features, ensuring that the solution can support a wider range of IoT applications, considering those requiring real-time data transmission too.

Moreover, the integration of advanced technologies such as AI and blockchain could be examined to further refine the efficiency and security of the roaming process. AI algorithms could be employed for predictive maintenance and network optimization, while blockchain technology could enhance the security and transparency of SLA negotiations and agreements.

A future direction that we want to investigate is incorporating SLA between service providers to enhance the operational robustness and reliability of the LoRaWAN architecture. This can be envisioned as a dynamic and automated negotiation framework facilitated by Smart Contracts technology, where SLAs could play a crucial role in defining clear service condition terms between the serving and forwarding Service Providers. This approach might cover aspects such as signaling traffic management, data volume pricing tiers, and maximum allowable packet loss rates, to ensure the terms of service provision are clearly defined and attached.

Our research contributes to the advancement of LoRaWAN technologies, particularly in the context of roaming functionalities. The positive results from our proof-of-concept tests and simulations stand as a solid basis for future developments in this field, to fully explore the potential of LoRaWAN for a wide array of IoT applications. The continuous evolution and growing demand for efficient, reliable, and scalable IoT networks highlight the importance of ongoing research and innovation in this domain.

Appendix A. RSSI and SNR results for the experimental tests

This appendix includes the RSSI and SNR results of each device for all the different tests that were conducted. This is corresponded to tests 1 to 5 of the script version for Figures A.29. And the results from tests 1 to 5 of the Golang version of the implementation of the roaming solution for Figures A.30.

Acknowledgment

This work was supported by the grants PID2020-116329GB-C22, TED2021-129336B-I00, and FJC2021-047073-I, funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. This work was also funded by Fundación Séneca (22236/PDC/23). This work was also a result of the ThinkInAzul and AgroAlNext programmes, supported by MICIU with funding from European Union NextGenerationEU (PRTR-C17.I1) and by Fundación Séneca with funding from Comunidad Autónoma Región de Murcia (CARM).

This work is partially supported by the project "AGREED – Agriculture, Green & Digital" - cod. ARS01_00254 - notice n. 2954 of 27/02/2020 and the European Union (NextGeneration EU) through the MUR-PNRR project SAMOTHRACE (ECS00000022).

References

[1] A. Osorio, M. Calle, J. D. Soto, J. E. Candelo-Becerra, Routing in lorawan: Overview and challenges, IEEE Communications Magazine 58 (6) (2020) 72–76.

[2] S. Sravan, S. Mandal, P. Alphonse, Sec-roam: Secure and efficient roaming in lorawan v1. 1, in: 2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, 2024, pp. 904–912.

[3] I. Butun, N. Pereira, M. Gidlund, Security risk analysis of lorawan and future directions, Future Internet 11 (1) (2018) 3.

[4] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, T. Watteyne, Understanding the limits of lorawan, IEEE Communications magazine 55 (9) (2017) 34–40.

[5] Lorawan® specification v1.0.
URL https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-0

[6] Lorawan® specification v1.1.
URL https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1

[7] S. Loukil, L. C. Fourati, A. Nayyar, K.-W.-A. Chee, Analysis of lorawan 1.0 and 1.1 protocols security mechanisms, Sensors 22 (10) (2022) 3717.

[8] Ts009-1.1.0 lorawan® certification protocol.
URL https://resources.lora-alliance.org/technical-specifications/ts009-1-1-0-certification-protocol

[9] J. R. Cotrim, J. H. Kleinschmidt, Lorawan mesh networks: A review and classification of multihop communication, Sensors 20 (15) (2020) 4273.

[10] G. Merlino, R. Asorey-Cacheda, L. D'Agati, F. Longo, A.-J. Garcia-Sanchez, J. Garcia-Haro, A. Puliafito, Infrastructure-centric, networkserver-agnostic lorawan roaming, in: 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), Vol. 21, IEEE, 2022, pp. 149–156.

[11] S. Balakrichenan, A. Bernard, M. Marot, B. Ampeau, Iotroam: design and implementation of a federated iot roaming infrastructure using lorawan, Tech. rep. (2021).

[12] M. M. Sandoche Balakrichenan, Antoine Bernard, B. Ampeau, Iotroam-design and implementation of an open lorawan roaming architecture, in: 2021 IEEE Global Communications Conference (GLOBECOM), IEEE, 2021, pp. 01–07.

[13] L. T. Arnol Lemogue, Ivan Martinez, A. Bouabdallah, Federated iot roaming using private dns resolutions, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2022, pp. 1–6.

[14] A. Lemogue, I. Martinez, L. Toutain, A. Bouabdallah, Federated iot roaming using private dns resolutions, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, 2022, pp. 1–6. doi:10.1109/NOMS54207.2022.9789852.

[15] J. Lamberg-Liszkay, T. Lisauskas, An alternative roaming model inlorawan, B.s. thesis, Linnaeus University, Faculty of Technology, Department of computer science and media technology (CM) (2018).

[16] S. Delbruel, N. Small, E. Aras, J. Oostvogels, D. Hughes, Tackling contention through cooperation: A distributed federation in lorawan space, in: Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks, Junction Publishing, 2020, pp. 13–24.

[17] M. Hamnache, R. Kacimi, A.-L. Beylot, Enabling an inter-operator roaming capability in lorawan networks, Ad Hoc Networks 139 (2023) 103025. doi:https://doi.org/10.1016/j.adhoc.2022.103025.
URL https://www.sciencedirect.com/science/article/pii/S1570870522001974

[18] R. K. Mohamed Hamnache, A.-L. Beylot, Unifying lorawan networks by enabling the roaming capability, in: 2021 IEEE 46th Conference on Local Computer Networks (LCN), IEEE, 2021, pp. 371–374.

[19] W. Ayoub, M. Mroue, A. E. Samhat, F. Nouvel, J.-C. Prévotet, Schc-based solution for roaming in lorawan, in: Advances on Broad-Band

Wireless Computing, Communication and Applications: Proceedings of the 14th International Conference on Broad-Band Wireless Computing, Communication and Applications (BWCCA-2019) 14, Springer, 2020, pp. 162–172.

[20] A. Durand, P. Gremaud, J. Pasquier, Decentralized LPWAN infrastructure using blockchain and digital signatures, Concurrency and Computation: Practice and Experience 32 (12) (2020) e5352.

[21] F. Flammini, A. Gaglione, D. Tokody, D. Dohrilovic, Lora wan roaming for intelligent shipment tracking, in: 2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT), 2020. doi:10.1109/GCAIoT51063.2020.9345843.

[22] W. Ayoub, F. Nouvel, A. E. Samhat, M. Mroue, J.-C. Prévotet, Mobility management with session continuity during handover in lpwan, IEEE internet of things journal 7 (8) (2020) 6686–6703.

[23] TS2-1.1.0 LoRaWAN® Backend Interfaces (Jan 2021).
URL https://lora-alliance.org/resource_hub/ts002-110-lorawan-backend-interfaces/

[24] Chirpstack, open-source lorawan® network server stack.
URL https://www.chirpstack.io/

[25] Raspberry Pi (Trading) Ltd., Raspberry pi 4 model b datasheet.
URL https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[26] R. D. Center, Rak5146 wislink lpwan concentrator, accessed: 2023-09-03 (2023).
URL https://docs.rakwireless.com/Product-Categories/WisLink/RAK5146/

[27] Python Software Foundation, Tiny db.
URL https://pypi.org/project/tinydb/

[28] Arduino, Mkr wan 1310 documentation.
URL https://docs.arduino.cc/hardware/mkr-wan-1310/

[29] B. Mishra, B. Mishra, A. Kertesz, Stress-testing mqtt brokers: A comparative analysis of performance measurements, Energies 14 (18) (2021) 5817.

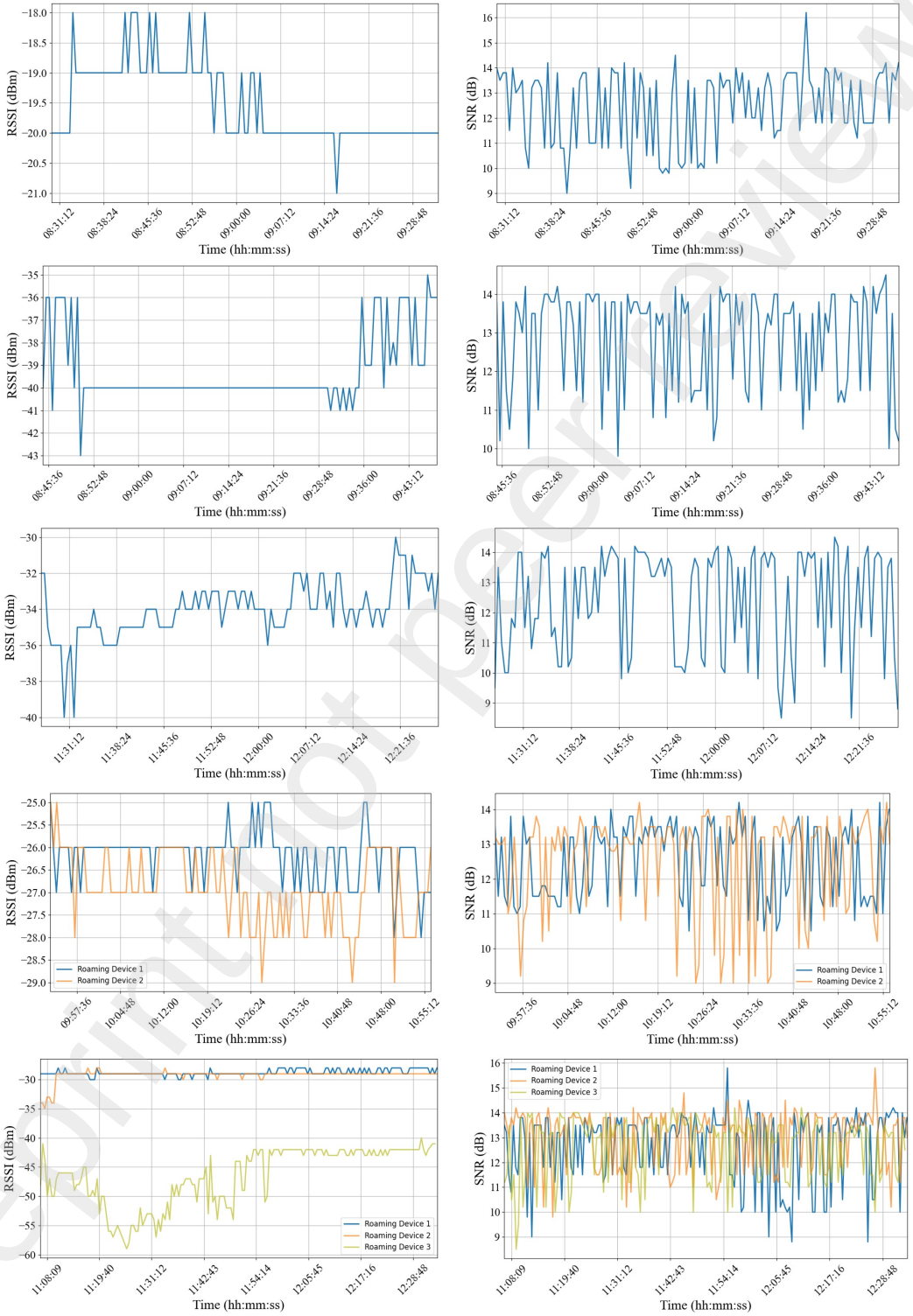

Figure A.29: RSSI and SNR at the Chirpstack Server for the tests performed with the script version. From top to bottom results for Test 1, Test 2, Test 3, Test 4, and Test 5.

Figure A.30: RSSI and SNR at the Chirpstack Server for the tests performed with the Golang version. From top to bottom results for Test 1, Test 2, Test 3, Test 4, and Test 5.