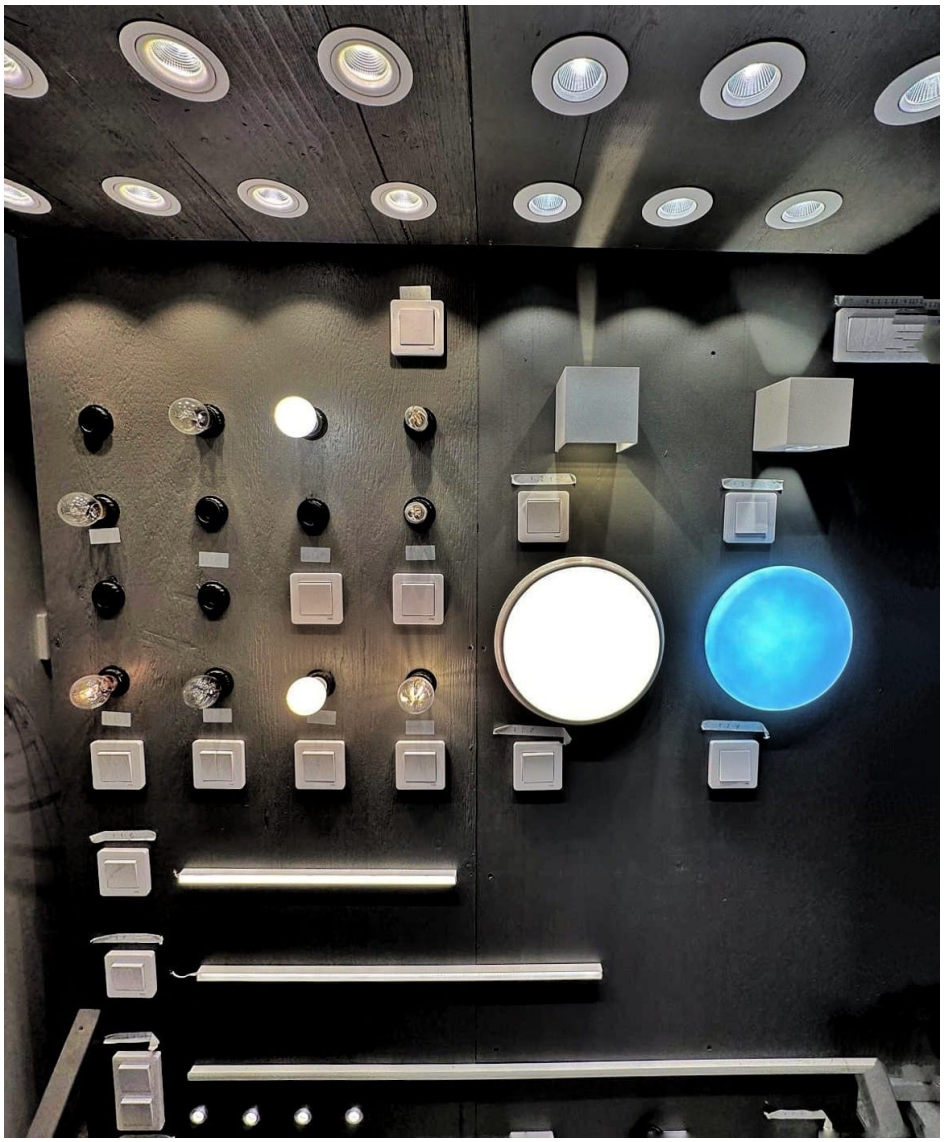
CHALMERS
UNIVERSITY OF TECHNOLOGY



Live Analysis of Mesh Networks

Master's thesis in Master Program Information and Communication Technology

ASHUTOSH GATTELU

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Live Analysis of Mesh Network

From Data to Insight - Analyzing Mesh network for Enhanced Performance and Reliability

ASHUTOSH GATTELU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Live Analysis of Mesh Network
From Data to Insight - Analyzing Mesh network for Enhanced Performance and Reliability

ASHUTOSH GATTELU

Supervisor: Robin Säfstorm, Plejd AB
Examiner: Tommy Svensson, Department of Electrical Engineering
Chalmers Advisor: Javad Aliakbari, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Communications Systems group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Wind visualization constructed in Matlab showing a surface of constant wind speed along with streamlines of the flow.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Live Analysis of Mesh Network
From Data to Insight - Analyzing Mesh network for Enhanced Performance and Reliability
ASHUTOSH GATTELU
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In the realm of the Internet of Things (IoT), the complexity of mesh architectures, particularly those employing rebroadcasting(RBC) mesh, presents formidable challenges. These challenges, ranging from flooding of packets due to rebroadcast, congestion, packet loss, demand innovative solutions to ensure the seamless operation of interconnected devices. This study embarks on a journey to unravel the intricacies of IoT mesh networks, employing a multidimensional approach that blends cutting-edge visualization techniques with sophisticated monitoring tool.

This study presents a Python-based application that has been painstakingly designed to make real-time IoT mesh network monitoring and analysis easier. The study provides thorough time series visualization by utilizing the powerful features of InfluxDB and Grafana, illuminating the dynamic interactions between devices in the network.

Through an elaborate test setup simulating diverse household scenarios with over 200 interconnected devices, the research meticulously examines the efficacy of live analysis in unraveling the complexities of mesh networks. Parameters such as packet count, system time synchronization, version cache integrity, level validation, and scene validations are meticulously scrutinized in real-time, providing profound insights into the network's behavior.

The findings underscore the superiority of live monitoring in detecting nuanced outcomes and ensuring real-time validation of network performance. This thesis not only contributes to advancing smart home technology evaluation but also offers practical insights into the implementation and analysis of IoT mesh networks. Through this work, we aim to enhance the reliability and efficiency of Plejd's smart home solutions, ultimately contributing to the broader field of IoT network research.

Keywords: Internet of Things(IoT), Mesh, InfluxDB, Grafana, Live Analysis, Time-Series data, Visualization

Acknowledgement

I am profoundly grateful to my thesis supervisor, Robin Säfstorm, for entrusting me with the opportunity to undertake my MSc thesis at Plejd AB. His invaluable guidance, insightful suggestions, and dedication to refining this report have been instrumental in this thesis development. I extend my sincere appreciation to Marwan Ghamlouch for his unwavering support and invaluable guidance throughout the thesis process. His profound expertise and innovative ideas have greatly enriched this project. I am equally indebted to Michael Marne for generously sharing his programming knowledge and skills, which significantly contributed to the success of this endeavor.

I express my heartfelt gratitude to my Examiner, Prof. Tommy Svensson, for his approval of this thesis. I am also thankful to Javad Aliakbari for his valuable suggestions and prompt responses to my queries, which greatly enhanced the quality of this report.

Furthermore, I am deeply appreciative of the unwavering support and love of my friends and family. Their encouragement and understanding have been my pillars of strength throughout this journey. Without their unwavering support, this project would not have been possible.

I am sincerely thankful to each and every individual who has played a role, no matter how small, in shaping this thesis and enriching my academic journey.

Ashutosh Mahesh Gattelu, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CDC	Communication Device Class
D3.js	Javascript Library
GAP	Generic Access Profile
GATT	General Attribute
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
IIoT	Industrial Internet of Things
LDAP	Lightweight Directory Access Protocol
LRU	Least-Recently-Used
MQTT	Message Queuing Telemetry Transport
NTP	Network Time Protocol
RBC	Rebroadcasting
REST	Representational State Transfer
SoC	System on Chip
SQL	Structured Query Language
TCP	Transmission Control Protocol
TSDB	Time - Series Database
UDP	User Datagram Protocol
UART	Universal Asynchronous Receiver-Transmitter

Contents

List of Figures

1
Introduction

IoT, or the Internet of Things, is a network of connected devices equipped with sensors, software, and other technologies that allow them to collect and exchange information. Its crucial components include sensors to gather information, actuators to perform actions, connectivity enabling communication between devices and servers, data processing to analyze information, and user interfaces for interaction. In everyday life, IoT applications such as smart homes, wearable health gadgets, and connected vehicles improve efficiency and convenience by streamlining processes and making better decisions [3].
IoT uses a variety of wireless communication topologies, such as mesh networks, star topologies, and hybrid configurations. Mesh networks provide stability and scalability by enabling devices to communicate directly with one another, resulting in a self-healing network [4]. Star topologies consolidate communication around a hub, which simplifies management but may result in single point failure [5]. Hybrid configurations incorporate components of both to improve network performance based on individual application requirements [6]. These wireless communication infrastructures play an important role in enabling smooth connectivity and data sharing among IoT devices, thus facilitating the realization of its benefits in everyday life [7].
With this foundational understanding of IoT and its communication topologies, we can now transition to exploring the specific context of a IoT company in the field. The following section provides a background on Plejd, a company that epitomizes the practical application of these technologies.

1.1 Company's Background

Plejd is a growing Swedish technology company that specializes in smart lighting and home automation solutions. Plejd, known for its unique approach, designs, develops, and manufactures finely crafted products in Sweden for seamless integration into both commercial and residential environments. Plejd provides a wide range of lighting control options, allowing users to easily enhance their spaces using cutting-edge wireless technology. Plejd's sturdy mesh network allows for scalable solutions that can be customized to changing needs, whether you're beginning small or going huge. Plejd is committed to quality and variety, ensuring that every home and company has access to the ease and efficiency of modern smart lighting and automation [8].
Understanding Plejd's background sets the stage for a deeper dive into the broader

context of IoT and its transformative impact on modern communication systems. The next section will further elucidate the importance of mesh networks in this interconnected ecosystem.

1.2 Background

In today's interconnected world, IoT has emerged as a transformative force, changing the way things communicate and interact. Mesh networks are key to this transition, as they enable strong device-to-device communication. Mesh networks, unlike traditional communication topologies, provide redundancy and resilience by establishing several communication channels, ensuring consistent connectivity even in the event of network disruptions or node failures[9].
However, the intrinsic complexity of mesh networks creates considerable obstacles. The interconnected nodes form a complex web of communication channels, potentially causing bottlenecks, duplicated messages, increased network traffic, and latency. Furthermore, the distributed architecture of mesh networks poses weaknesses, as outdated messages can remain in the system, increasing the likelihood of failure[10].
Understanding and managing the complicated nature of mesh network design is critical, especially in light of Plejd's IoT ecosystem. To maintain smooth network operation, developers and engineers entrusted with improving Plejd's smart device performance must manage the complexities of real-time monitoring and analysis.
To address these problems, Plejd has created a comprehensive test setup that includes its product line, as well as specialized testing equipment that has been precisely constructed to simulate common household conditions. This configuration allows for rigorous examination of Plejd's mesh network under a variety of scenarios, including both machine-generated and human-induced procedural cases, as well as planned test scenarios. Plejd plans to use live analysis tools to identify potential bottlenecks, improve overall efficiency, and obtain a better understanding of the behavior of its mesh network.
With this detailed background in mind, we now transition to the specific aims of this thesis. The following section outlines the goals and objectives that drive this research, providing a clear roadmap for the subsequent analysis.

1.3 Aim

The aim of this thesis is to perform a thorough examination of the complexities of Plejd's mesh network, with an emphasis on its role in enabling effective functioning and connectivity across their portfolio of smart home devices. The goal of delving into the complexities of IoT mesh networks and the issues they provide is to gain a better knowledge of Plejd's mesh capabilities in real-world scenarios.
This project intends to compare the effectiveness of live analysis to typical sequential testing procedures by creating a bespoke solution for real-time monitoring and analysis. The ultimate goal is to help progress smart home technology evaluation methods by setting rigorous testing standards, developing a useful tool for building

and analyzing mesh networks, and presenting the findings in a clear and understandable manner.
Following the establishment of our aims, we proceed to define the specific scope of this thesis. The next section will detail the parameters and constraints within which this research operates.

1.4 Scope

The scope of this thesis comprises an evaluation of Plejd's mesh network using a test setup that simulates a household environment with 200 Plejd products and specific equipment for monitoring mesh messages and electrical events. Various scenarios will be carried out, with the effects thoroughly observed. The emphasis is on comparing the effectiveness of live analysis and sequential testing approaches, particularly in terms of discovering subtle outcomes. The evaluation will include the effectiveness of live monitoring in both fixed and procedural test cases generated by both human and computer inputs, with the goal of offering insights into live analysis' superiority over traditional testing methodologies.
With a clear understanding of the scope, we can now outline the structure of the thesis. The following section provides an overview of the chapters and their respective focuses, guiding the reader through the research process.

1.5 Report Structure

The thesis report is divided into five chapters that outline the steps taken during the research and analysis. The Introduction section gives a brief overview of the thesis work; the subsequent theory chapter delves into the fundamentals of RBC mesh, as well as an overview Mesh System architecture, mesh commands, InfluxDB and Grafana. Moving on to the Methods section, the study explains how rule sets for test case execution have been created along with tools for live analysis, and finishing with a summary of the scenarios addressed. The Results chapter presents the findings derived from the analysis conducted throughout the thesis. Finally, the Conclusions and Future Work section concludes the report by summarizing major findings and identifying areas for further study and improvement.

1.6 Related Work

In the area of real-time data analysis, the research by [11] on streaming analytics for IoT networks stands out. They implemented a system that continuously monitored network data streams to detect anomalies and trigger alerts. The primary advantage was the immediate feedback and actionability of the insights derived from the data. However, the system's performance was heavily dependent on the network's bandwidth and computational resources, which could limit its scalability .
Comparatively, the current thesis builds upon these foundations by integrating real-time analysis tools specifically designed for IoT mesh networks, such as the Plejd

mesh. By focusing on live data analysis and visualization through tools like Grafana and InfluxDB, this research aims to address some of the limitations noted in previous studies. For instance, unlike the redundancy-focused approach of [12], this thesis emphasizes efficient real-time monitoring without significantly increasing network complexity.
The current work also leverages time-series data visualization to enhance the interpretability of network status and faults. Additionally, by employing a live monitoring tool that can be readily adapted to various network scales and conditions, this research seeks to overcome the scalability issues highlighted by [11].

2

Theory

This chapter provides a detailed exploration of the foundational components that underpin Internet of Things (IoT) ecosystems. We begin by dissecting the essential elements of IoT systems, from sensor nodes to gateway devices, highlighting their roles in facilitating seamless communication. Moving forward, we delve into the intricate layers of mesh system architecture, uncovering the mechanisms governing communication protocols and network topology. The discussion then shifts to the hardware landscape, exploring supported platforms and their implications for IoT deployments. Subsequently, we explore mesh commands, decoding the language that drives device configuration and management. Following this, we delve into database solutions tailored for IoT analytics, focusing on time-series databases like InfluxDB. Lastly, we examine visualization tools such as Grafana, highlighting their role in transforming raw data into actionable insights. Through this exploration, readers gain a comprehensive understanding of the foundational elements driving connectivity and data analytics in IoT environments.

2.1 Internet of Things (IoT)

The Internet of Things (IoT) is a network of physical devices integrated with sensors, actuators, and connectivity that can gather, exchange, and act on data to fulfill specified goals or tasks. The Internet of Things has emerged as a disruptive technology, transforming a variety of industries by increasing efficiency, production, and decision-making skills as stated by [13].

2.1.1 Components of IoT

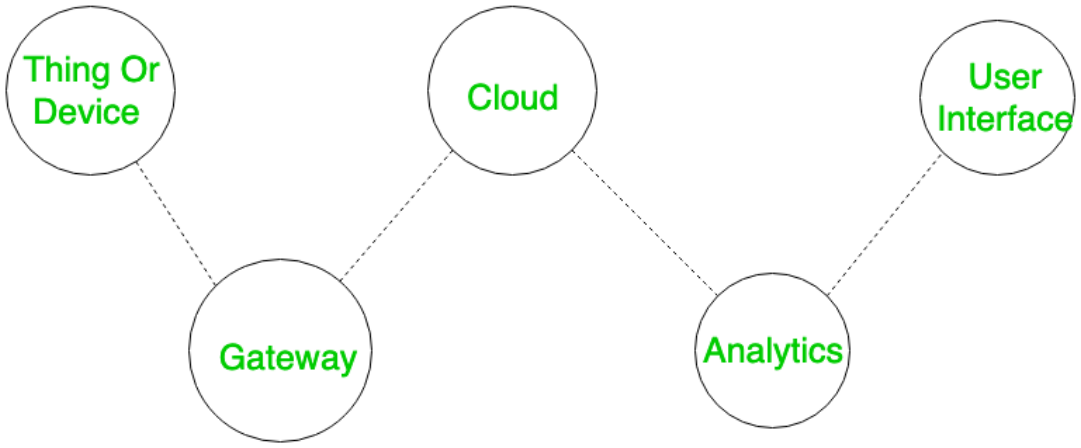


Figure 2.1: Components of IoT

The figure 2.1 illustrates the key components of the Internet of Things (IoT) ecosystem as mentioned in [14]. These components include:

- **Things or Devices**
  These components, depicted in the diagram 2.1 as the physical devices in the IoT network, collect data from their surroundings (sensors) and enable devices to take actions based on that data (actuators).
- **Gateway and Cloud**
  The IoT ecosystem relies on various connectivity options, including wired or wireless networks, to enable communication between devices and external systems. This connectivity is represented in the diagram by the gateway and cloud components.
- **Analytics**
  Once data is collected by sensors, it undergoes processing, analysis, and interpretation to derive meaningful insights. This processing component, represented as analytics in the diagram, plays a crucial role in generating actionable intelligence from IoT data.
- **User Interfaces**
  Interfaces such as mobile applications or web portals provide users with the ability to interact with and control IoT devices. These user interfaces facilitate seamless communication and management of IoT systems.

By integrating these components, the IoT ecosystem enables the seamless exchange of data and the execution of intelligent actions, driving efficiency, automation, and innovation across various domains.

2.1.2 Communication Topologies in IoT

The choice of communication topology in an IoT network significantly impacts its performance and scalability. Three common topologies include:

- **Star Typology**
  Devices are connected to a central hub or gateway, facilitating communication between devices and centralized control.
- **Mesh Topology**
  The devices are interconnected, forming a network where each node can communicate directly with neighboring nodes, enhancing reliability and scalability.
- **Hybrid Topology**
  Combines aspects of mesh and star topologies to maximize network performance according to specific needs [15].

2.1.3 Communication Protocols in IoT

Several communication protocols are utilized in IoT networks to facilitate efficient data exchange. These include:

- **MQTT**
  Simple, lightweight publish-subscribe protocol that makes effective communication across erratic networks possible for devices with limited resources.
- **CoAP**
  Committed to low-power networks and device constraints, CoAP facilitates light-weight, straightforward communication based on RESTful principles [16].
- **BLE**
  Short-range, low-power wireless communication protocol that is frequently utilized in wearable technology and smart home applications.

2.1.4 Key use cases of IoT

IoT finds applications across various domains, including:

- **Smart Home Automation**
  IoT devices enable remote monitoring and control of home appliances, lighting, security systems, and energy management.
- **IIoT**
  Industry-specific predictive maintenance, asset tracking, and real-time monitoring are made possible by IoT, which boosts productivity and decreases downtime.
- **Healthcare**
  Wearable medical devices, smart medical equipment, and remote patient monitoring are examples of IoT applications that enhance patient outcomes and healthcare delivery.
- **Smart Cities**
  IoT enables efficient resource management, traffic monitoring, waste management, and infrastructure optimization, enhancing urban sustainability and livability.

2.1.5 Impact of IoT on Global Transformation

The widespread adoption of IoT is changing the economy, industry and society in many ways. IoT improves ease, productivity, and efficiency in a variety of fields by

facilitating real-time data collecting, analysis, and decision-making. IoT is changing how we interact with the physical world, spurring innovation and opening up new possibilities for the future. Examples include optimizing supply chains, increasing healthcare results, and enabling sustainable urban development [17].

2.2 Understanding Communication in IoT

In the Theory section, we established the fundamental concepts of communication topologies and protocols in IoT networks. Mesh topology emerged as a critical architecture due to its ability to provide robust connectivity and scalability, making it well-suited for diverse IoT applications. However, the effective implementation of mesh networks presents unique challenges, such as ensuring reliable communication and efficient data synchronization across a large number of interconnected nodes.

2.2.1 Exploring RBC Mesh: Enhancing Mesh Communication

Following our discussion on the importance of mesh topology in IoT networks, we now delve into the details of the Rebroadcasting Mesh framework (RBC Mesh) to address specific challenges within this framework. RBC Mesh represents a sophisticated solution to the complexities of mesh communication, offering innovative approaches to wireless infrastructure. By leveraging rebroadcasting mechanisms and dynamic flood control algorithms, RBC Mesh ensures seamless communication and data synchronization among BLE-capable nodes. Through a comprehensive examination of its architecture and functionalities, we illuminate the pivotal role of RBC Mesh in enhancing the reliability and efficiency of mesh networks in IoT deployments.

2.2.2 RBC Mesh

The Rebroadcasting Mesh framework [18], commonly referred to as "the framework" or RBC mesh, functions as a Wireless infrastructure that aims to synchronize states among a network of nodes equipped with Bluetooth Low Energy (BLE). In this framework, communication is facilitated through rebroadcasting, where each node in the mesh receives broadcast messages from neighboring nodes and rebroadcasts them to further propagate the information. This mechanism allows wireless devices to communicate even when they are not within direct radio range, as intermediary nodes relay messages throughout the network.

Using the nRF51 SoftDevice (A SoftDevice is a wireless protocol stack that complements an nRF5 Series SoC) [19], which offers fundamental Bluetooth capabilities, is at the center of the framework. The SoftDevice uses the Timeslot API to intelligently distribute radio time so that nodes can efficiently receive and transmit messages. Every node in the mesh network has access to the same set of indexed data slots, each of which contains data that has been propagated using version numbers. The version number is increased by a node when it modifies a value, guaranteeing

data synchronization between all nodes.

The propagation of values is governed by the implementation of the IETF RFC6206 "Trickle" algorithm [20], a flood control mechanism designed for lossy, low-power networks. Message consistency and update frequency are used by the Trickle algorithm to dynamically define the intervals at which nodes broadcast values. The framework can adapt to changes in node density and value update rates because of this adaptive methodology.

Every handle-value pair in the mesh corresponds to a Trickle instance, and the values are maintained using a LRU caching system. Nodes stop retransmitting values as they leave the cache, which guarantees effective resource use. Furthermore, the framework offers a GATT [21] feature in the SoftDevice as a means via which standard BLE devices can communicate with the mesh.

To sum up, the Rebroadcasting Mesh architecture provides a reliable and effective framework for coordinating states across BLE-capable nodes. The framework is designed to enable dependable communication and data synchronization in a variety of Internet of Things applications, from industrial IoT deployments to smart home systems, by utilizing rebroadcasting and dynamic flood control mechanisms.

2.3 Mesh System Architecture

We are examining the Mesh System Architecture at this stage to provide a comprehensive understanding of the underlying framework that supports RBC Mesh communication. By delving into the intricacies of the architecture, we aim to elucidate the hierarchical structure and functional layers that enable seamless communication within the mesh network. This exploration is crucial for understanding the design principles, message handling mechanisms, and data transmission processes that underpin the operation of RBC Mesh. Moreover, by analyzing the system architecture, we can identify the key components and their roles in facilitating efficient communication, data synchronization, and application interaction within the IoT ecosystem. Therefore, our focus on the mesh system architecture serves as a foundational step towards comprehending the intricacies of RBC Mesh and its implications for IoT deployments.

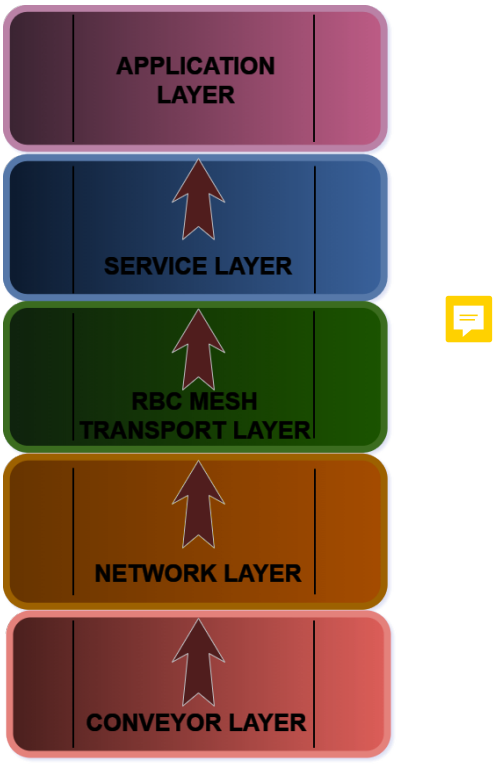


Figure 2.2: Mesh System Architecture

In exploring the Mesh System Architecture 2.2, we delve into its layered structure and functional components, providing insight into how it supports RBC Mesh communication.

2.3.1 Overview

The layers that make up the Mesh System Architecture are assigned specific roles and are arranged in relation to the Bluetooth Low Energy (BLE) core architecture. The Bluetooth LE layer is the base of the Bluetooth LE stack, which is necessary for wireless communication.Further information on Bluetooth mesh networking can be found in [22, 23]. This layer functions as the support structure, giving the mesh architecture above it access to essential communication capabilities. It is important to remember that the mesh system requires a functional Bluetooth LE stack. We will now examine each layer of the mesh design, working our way up from the bottom.

2.3.2 Conveyor Layer

The Mesh infrastructure Architecture's Conveyor layer acts as a conduit between mesh messages and the underlying communications infrastructure that transmits and receives them. The protocols used by the communications system to manage mesh packets are specified in this tier. The GATT conveyor and the advertising conveyor are the two conveyors that are currently defined. The Advertising conveyor transmits and receives mesh packets by utilizing Bluetooth LE's GAP advertising and scanning features. A BLE peripheral can only be linked to one central device

at a time due to the exclusive nature of GATT connections. A peripheral that is connected to a central device ceases to advertise itself, and until the link is broken, other devices are unable to notice it or connect to it. Once the connection with the central device is established, other devices in the network communicate using the RBC mesh.

2.3.3 Network Layer

The Network layer of the Mesh System Architecture establishes different message address types and defines a network message format. This format enables the transmission of RBC Mesh transport layer packets via the Conveyor layer. By facilitating the transport of mesh packets, the Network layer plays a crucial role in ensuring efficient communication within the mesh network.

2.3.4 RBC Mesh Transport Layer

Rebroadcasting, in which each mesh node receives broadcast messages from nearby nodes and rebroadcasts them to spread the information further, is one method of communication facilitated by the RBC Mesh Transport layer. Due to intermediary nodes' ability to transport messages throughout the network, wireless devices can communicate even when they are not in close radio range. Moreover, the Transport layer guarantees reliable and safe communication within the mesh network by handling the encryption, decryption, and authentication of application data traveling to and from the Service layer.

2.3.5 Service Layer

The Service layer defines how applications utilize the transport layer, encompassing tasks such as defining the format of application data and handling Mesh version numbers. It controls the encryption and decryption process performed in the transport layer and ensures data integrity by verifying that received data is intended for the correct network and application before forwarding it up the stack.

2.3.6 Application Layer

The Application layer sits on top of the Service layer and is in charge of putting certain features customized for end-user apps into action. In order to add application-specific logic and features like device control and data visualization, it interfaces with the Service layer to access and manipulate data shared inside the mesh network. It guarantees smooth communication and interaction with mesh-enabled devices while preserving compatibility and interoperability by utilizing capabilities from lower layers. To further improve usability and user experience, the Application layer offers a user-friendly interface for controlling mesh network features [22, 23].

2.4 Supported Hardware

The RBC mesh supports the nRF5x [24] series of microcontrollers manufactured by Nordic Semiconductors. Additionally, the platform offers compatibility with 2.4GHz Bluetooth Low Energy (BLE) radios, as well as 2.4GHz dual-mode BLE and IEEE 802.15.4 radios. This hardware versatility ensures flexibility and compatibility for a wide range of IoT applications and deployments.

Following the exploration of supported hardware, the focus now shifts to the operational aspects of the mesh network. Understanding how devices within the mesh network are configured, monitored, and managed is crucial for effectively deploying and maintaining such networks. Hence, the subsequent section delves into the intricate world of mesh commands, elucidating their significance in facilitating seamless communication and control within the mesh ecosystem.

2.5 Mesh Command

Through a serial port connection, the serial interface of the mesh stack acts as a communication bridge, allowing external devices to communicate with the devices running the mesh stack. This interface, which offers a method of configuration, monitoring and management, is essential for managing and controlling different parts of the mesh network from external systems.
Mesh commands are given to the nRF5, the device running the mesh stack, from the host controller (such as a PC via a nRF52840 dongle [25]). These instructions can be used to set configuration parameters, inquire about device status, start firmware updates, or request information from the mesh stack.
Commands are generally classified into two types: set commands and get commands. Set commands are used to configure or modify parameters within the mesh stack, such as setting device parameters, updating network configurations, or initiating actions like provisioning new devices. On the other hand, get commands are used to retrieve information from the mesh stack, such as querying device status, retrieving network topology information, or obtaining diagnostic data.
Mesh commands are categorized into different classes according to their use and purpose. Device commands deal with operations and configurations specific to a device; Application commands control behaviors at the application level; Segmentation and Reassembly commands handle large data packets; Configuration commands manage network configuration settings; Provisioning commands handle device provisioning processes; Direct Firmware Upgrade commands initiate firmware updates; and Model Specific commands are specific to a particular model or functionality of a device. Moreover, contingent on the particular implementation, time-based, RF parameter, and calibration instructions might be available.
A user-friendly interface for communicating with devices running the mesh stack and serial interface is provided by the Interactive Python sniffer console. By enabling COM port opening, it makes command-line communication with the mesh stack

possible. In addition, the console can be used to view mesh packets, which helps in debugging, analyzing, and troubleshooting networks.

2.6 Database Solutions for IoT

In the rapidly evolving landscape of Internet of Things (IoT) applications, managing and analyzing the vast streams of data generated by connected devices is of paramount importance. This section explores various database solutions tailored to address the unique challenges posed by IoT data, providing insights into their architecture, functionalities, and applicability in IoT ecosystems.

2.6.1 Database Requirements for IoT

The significant volumes of data created by applications in the Internet of Things (IoT) pose problems in terms of scalability and data input speed, making database requirements critical. Effective storage requires efficient management of heterogeneous data types and dynamic datasets. Sharding and replication approaches limit scalability, while hardware, system design, consistency requirements, data model complexity, and index utilization affect overall throughput. In order to satisfy IoT requirements without sacrificing performance, a database that is specifically designed for these factors must be created.

2.6.2 Time-Series Database

Time-series data, such as server statistics, system performance, sensor readings, and market trades, are best stored in a Time Series Database (TSDB). A TSDB, which is specifically made to manage time-series metrics, is a tool for tracking changes over time. It has features including effective data scans, summarization, and information life-cycle management.

2.6.3 InfluxDB: Empowering IoT Data Analytics

Among the array of database solutions, InfluxDB stands out as a robust time-series database tailored for IoT applications. This subsection provides an in-depth exploration of InfluxDB's architecture, features, and capabilities. From its distributed architecture to seamless integration with data visualization tools like Grafana, InfluxDB emerges as a pivotal component in IoT data analytics pipelines.

Figure 2.3: InfluxDB Logo [1]

The distributed time series database InfluxDB, which was developed in 2013 by InfluxData, is open-source and uses LevelDB for key-value storage. It is written in Go. It provides smooth interaction for front-end apps with client libraries and an HTTP API. Perfect for time series data, InfluxDB excels at aggregating values in real-time. It may be used with programs such as Grafana and allows SQL searches with key-value pairs and timestamps for measurements, series, and points. Floating points, strings, and integers are examples of values. The protocols used by InfluxDB for data storage are TCP, HTTP, and UDP.

2.6.3.1 Architecture of InfluxDB

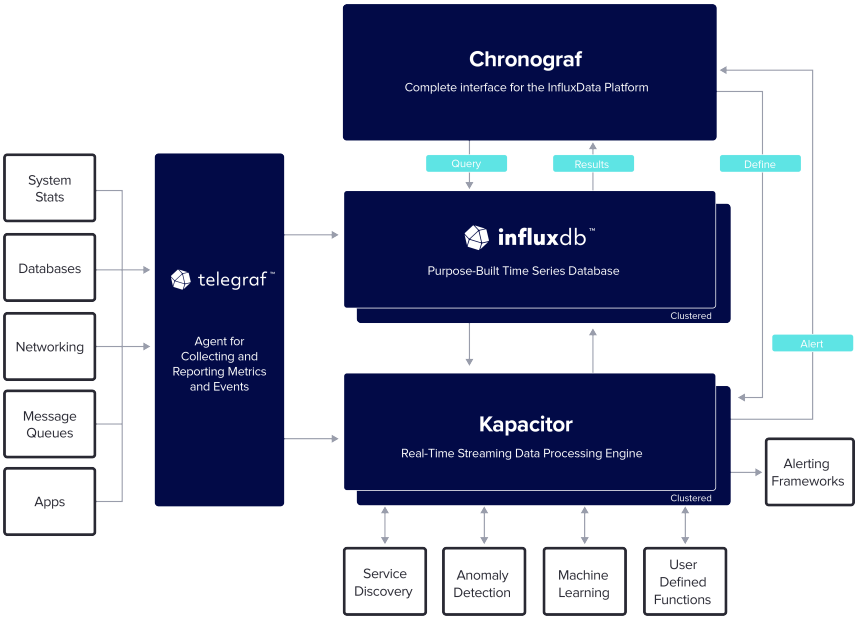


Figure 2.4: Architecture of InfluxDB [1]

The general architecture of InfluxDB is illustrated in Figure 2.4.Four layers make up the InfluxDB architecture: Telegraf, InfluxDB, Chronograf, and Kapacitor. The time series database designed to handle high write and query loads is called InfluxDB, and Telegraf is the server-driven open-source agent for collecting and reporting metrics. The web application for managing alerts, tracking infrastructure,

and visualizing data is called Chronograf. As the framework for handling information, Kapacitor makes alert creation, ETL chores, and anomaly detection easier. Within the InfluxData ecosystem, these layers work together to create a strong architecture that facilitates the effective administration and analysis of time series data.

2.7 Grafana: Visualizing IoT Insights

Transitioning from database solutions, we now delve into Grafana, a powerful tool for visualizing data insights in IoT applications. This section offers a comprehensive exploration of Grafana's features, highlighting its role in transforming raw data from databases like InfluxDB into intuitive visual representations. From real-time dashboards to advanced analytics, Grafana seamlessly integrates with InfluxDB, forming a dynamic duo in driving actionable insights and facilitating informed decision-making in IoT environments.

Figure 2.5: Grafana Logo [2]

Grafana, an open-source platform, facilitates data visualization and monitoring by enabling users to create and share dashboards showcasing real-time data from diverse sources like databases, servers, and cloud services. Supporting a wide range of data sources including InfluxDB, Prometheus, and Graphite, Grafana offers features such as alerting, annotations, and plugins for enhanced visualization and integration. Widely utilized in system monitoring, IoT, and network monitoring fields, Grafana simplifies the visualization and comprehension of large datasets, aiding in trend identification and anomaly detection. Its alerting functionality enables users to set up notifications based on predefined conditions, ensuring timely updates on system status [2].
Visualization tools like Grafana are vital in IoT due to the immense volume of data generated. They provide intuitive visual representations, enabling stakeholders to monitor IoT systems in real-time. Grafana facilitates data interpretation, helping to timely decision making and problem solving. It helps identify trends, anomalies, and patterns in IoT data, informing optimization strategies. Through clear visualizations, stakeholders gain insight into system performance and device behavior. Grafana enhances collaboration by offering a common platform for data analysis. It enables predictive maintenance, resource allocation, and optimization of IoT sys-

tems. By improving situational awareness, Grafana improves operational efficiency in IoT deployments. Its intuitive interface makes it accessible to users with varying technical expertise. Overall, Grafana plays a crucial role in maximizing the value of IoT investments by transforming raw data into actionable insights.

2.7.1 System Design of Grafana

Grafana's system design includes a number of essential parts that function as a unit. Raw data for Grafana dashboards comes from several data sources, such as time-series databases and cloud services. Plugins connect to various sources, query the data, and format them so that Grafana can use them more easily. Grafana uses the D3.js package to visualize the data after they have been collected, filtered, and aggregated as necessary. To present visualizations, including historical and real-time data, users build customized dashboards. Alerting functions guarantee timely system status updates by informing users of specific conditions. Access is managed via mechanisms for authorization and authentication that enable a number of techniques, including LDAP and basic authentication.
Grafana's scalability makes it possible to cluster and scale horizontally for higher load and high availability. Its adaptable architecture supports a wide range of display choices and data sources. Grafana offers a complete solution for data gathering, processing, visualization, and monitoring by combining these elements. Because of its strong system architecture, Grafana is a well-liked option for businesses in a variety of sectors. It also increases scalability, dependability, and usability.

3
Methods

The methodology chapter describes the methodical approach that ~~will be~~ used to achieve the goals of this thesis, including data gathering techniques, experimental designs, procedures, and it is depicted in the cross- Functional flowchart Fig. 3.1. It is designed to give enough specifics so that anybody with a basic grasp of the subject can follow the procedure and get results that are similar. The client company's ability to assess whether the stated goals may be achieved depends on the approach that was selected. It also clarifies why the chosen methodology is expected to produce a consistent result.

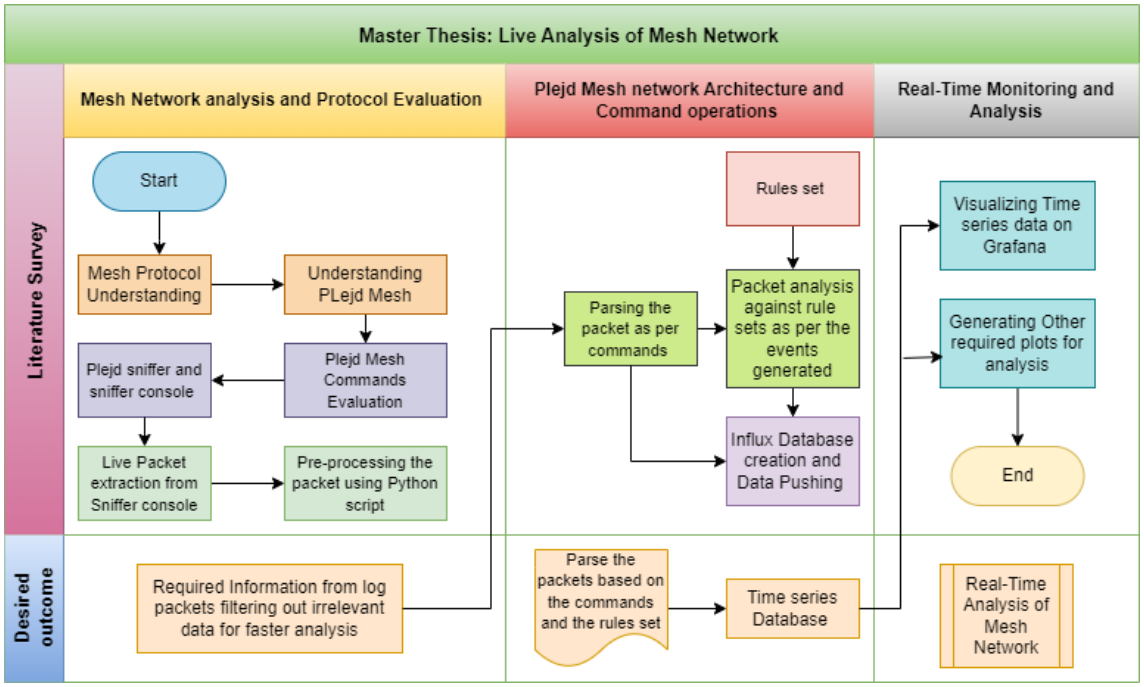


Figure 3.1: Cross - Functional Flow of the Thesis

To ensure informed decision-making throughout the study process, the methodology for this thesis entails a thorough literature assessment of previous studies on IoT mesh networks and protocols. The Plejd Mesh Network is examined in great detail, including its architecture, command protocols, and operational dynamics. To provide effective monitoring and analysis, it is mandatory to become familiar with the Plejd Mesh commands. In real-world experiments, mesh packets and associated electrical events are recorded in virtual homes using the Plejd Sniffer. In order to extract relevant data for additional analysis, captured packets are pre-processed and

analyzed using a Python tool. In order to create rule sets that correspond to command features and enable organized assessment, parsed packet analysis is essential. A database is created using InfluxDB to hold parsed data and processed packet data, guaranteeing easily accessible and structured data management. Grafana helps with real-time monitoring and key information visualization by enabling time-series live analysis of packet data from the InfluxDB database. Grafana also creates a variety of graphs for further analysis that shed light on the effectiveness of the test setting. The approach ensures ongoing improvement and validation of the testing procedure by enabling iterative refinement based on insights from preliminary analysis.

3.1 Implementation of Real-Time Mesh Packet Analysis Tool

The Real-Time Mesh Packet Analysis Tool created for this thesis is a complex Python-based tool that has been methodically designed to allow for comprehensive real-time analysis of mesh network packets. It is a critical component that connects sniffer hardware to the Grafana visualization platform, allowing for seamless data flow and visualization for better monitoring and troubleshooting.

At its core, the tool takes advantage of Python's powerful capabilities to perform data collecting, preprocessing, parsing, and visualization tasks with exceptional quickness and precision. Using InfluxDB client APIs, the tool creates a seamless interface with the Influx database, enabling for the construction of structured queries for efficient data storage and retrieval.

The tool is architected to handle the complexities of mesh networks, accommodating up to 256 devices in a single network. While theoretical limits allow for this scale, practical considerations cap the number of devices at around 80 to maintain optimal performance and low-latency communication.

Exception handling mechanisms are intricately woven into the tool's design to ensure uninterrupted operation, even in the face of hardware discontinuities or unforeseen events. Real-time processing capabilities are a hallmark feature, with the tool meticulously engineered to adhere to strict time constraints for timely data visualization.

The tool provides comprehensive metadata analysis capabilities, including packet counts, validation statuses (successful, unsuccessful, pending), and latency measurements. Notably, latency calculations are performed to gauge packet processing efficiency, excluding radio and propagation parameters to provide accurate insights.

The detailed steps and logic of the Real-Time Mesh Packet Analysis Tool are outlined in the algorithm stated in 1. This algorithm highlights the initialization, data capture, packet processing, and data pushing procedures critical to the tool's operation. It serves as a blueprint for the real-time analysis process, ensuring a structured and efficient approach to managing mesh network packets.

Algorithm 1 Real-Time Mesh Packet Analysis Tool

Require: Sniffer hardware connected, InfluxDB connection

```
Initialize Real-Time Mesh Packet Analysis Tool
function InitializeTool()
    sniffer.connect() {Connect to Sniffer Hardware}
    influx_client = InfluxDBClient(host="xx.xx.xx", port=xxxx, username='xxxx', password='xxxx')
    influx_client.switch_database('xxxx') {Establish connection to InfluxDB}
    return influx_client
end function
Main Function
function Main()
    mesh = Mesh(mesh_id, mesh_crypto, log_all=True, port="xxxx")
    mesh.snifferThreadCallback = partial(re_auth, mesh)
    mesh.start()
    last_data_push_time = latest_time
    f = open("./console_log.txt", 'w')
while True do
  packets = mesh.get_all_mesh_data() {Capturing raw mesh data}
  if len(packets) != 0 then
    version_filtered_data = versionfilter(packets)
    for packet in version_filtered_data do
      Write packet details to file for data logging and create a sniffer console on the terminal
      version_handling_rule(packets)
      Update metadata counts
      if System time package received then
        System_time_verification(packet)
      else if Output announce package received then
        Handle output announce
      else if Other command received then
        Process other commands
      end if
    end for
  end if
  Execute actions and handle validations
  Handle pending level comparisons
  Handle pending scene checks
  if 10 seconds passed since last data push then
    Push visualization data to InfluxDB
    Clear visualization list
    Update last data push time
  end if
end while
end function
```

Figure 3.2: Capture and Processing of Mesh packet Flow

3.2 Capturing and Processing Mesh Packages

Mesh packages are captured through a number of crucial steps that make use of the nRF52840 chip's capabilities as well as a number of software tools to effectively gather, process, and evaluate mesh messages. First, mesh messages are wirelessly collected by the nRF52840 device from the network's nodes. Efficient serial communication is ensured by converting these recorded messages into a byte structure compatible with UART. The USB CDC, a composite universal serial bus device class that enables communication between the nRF52840 chip and a host system, usually a computer, receives the UART-compatible data. On the host system, a Python script running above this stack utilizes the PySerial library to extract the UART bytes from the CDC. The bytes are rearranged to match the original mesh package data after they have been retrieved. The last stage is to parse the reor-

ganized data and decode the individual commands and the data payloads they are associated with by considering the mesh command and payload information. The process is illustrated in the fig 3.2.

3.3 Payload Caches for Mesh Data Optimization

One essential element for effectively storing and handling payload data in a mesh network is a payload cache. The payload cache is made to hold payload data of different sizes, which can change depending on the particular mesh instructions and the kinds of operations needed. In order to store payload data for each node linked to the network, this cache is essential to the mesh's ability to support up to 256 devices.Depending on the particular command types and operational requirements, the payload data in the cache includes pertinent details including node state, level, output announcements, and other relevant data. The payload cache makes it easier to quickly access and retrieve this data locally as needed for a variety of mesh operations and data analysis tasks.
The capacity to reduce needless network traffic resulting from the repeated rebroadcasting of identical mesh packages containing different version numbers and timestamps is a key benefit of using a payload cache. This is a common occurrence in mesh networks and can result in inefficient data processing and analysis. Incoming mesh packages can be compared to the payload data that has been stored for each individual node using the payload cache, which serves as a filtering mechanism. Through this comparison, the system may determine which packages are redundant and can be removed, optimizing network traffic and improving the overall efficiency of data analysis.

3.4 Data Processing with Version Filtering and Version Cache

The performance of data processing in mesh networks is improved by version filtering mesh packages and using a version cache. A version cache is a kind of repository where version numbers connected to certain mesh network indices are kept. The version numbers of newly arrived mesh packages are compared to those of the previously downloaded packages for the same index that are kept in the cache. The package is considered significant for additional processing if it has a higher version number, which signals updated or modified information.
The version filtering function reads the global version cache after receiving a list of mesh packets as input. Every packet's version number is compared to the version number stored in the cache for the associated index. Higher version number packages are kept and added to the filtered packages list, whereas lower version number packages are removed. This filtering system makes sure that only the most current and pertinent data is taken into account for further examination and action.
Mesh networks can maximize resource usage and data processing by utilizing version caching and version filtering. By filtering away redundant or out-of-date information,

needless processing overhead and network traffic are decreased. Using the most recent data ensures that system decisions are made using the most recent information available, which improves the mesh network's overall responsiveness and efficiency.

3.5 Parsing Intensity Levels from Payload Data

Level parsing is the process of taking the payload data from a mesh package and extracting the intensity level as an integer. Intensity level data supplied over mesh networks is frequently interpreted and used using this functionality. The level parsing function accepts a mesh packet as input, extracts and parses the intensity level data from the payload, and outputs the parsed intensity level as an integer number.

In the context of mesh data processing, where intensity level information is included within the payload of mesh packages, the level parsing function is commonly called. The level parsing function receives a mesh packet, extracts the pertinent intensity level data from the payload, and formats it into an integer format that may be used.

3.6 Removal of Overridden Commands for Level Comparison

The removal of overridden commands function plays a crucial role in ensuring the accuracy and efficiency of level comparison within mesh-based systems. Before adding a new level comparison from a control package, this function is employed to identify and remove any old command packages that would be overwritten by the new package. By considering only the latest and most relevant package comparisons, this process helps validate intensity levels accurately and discard outdated control messages.

When a new control package containing intensity level information is received, it may override previous commands sent to the same device. The removal of overridden commands function ensures that only the most recent package comparison, representing the latest intensity level setting, is retained for validation. This is particularly useful in scenarios where intensity adjustments are made rapidly, such as toggling a slider in a mobile app or pressing buttons to vary intensity quickly. By focusing on the latest comparison, unnecessary validations are reduced, and system efficiency is improved.

Taking a mesh packet as input, the function determines which commands need to be eliminated to create room for the new package by utilizing the level comparison list that already exists. Only the most recent and pertinent comparisons are kept when overridden instructions are detected and eliminated from the list. After that, the system returns the modified level comparison list which is now devoid of obsolete command for additional processing.

3.7 Version Numbering for Mesh Packets

In an Internet of Things mesh network, version numbers play a crucial role in uniquely identifying and organizing packets as they traverse the network. These version numbers are indispensable for maintaining a reliable and orderly network connection.
Primarily, version numbers facilitate the identification and elimination of duplicate packets. By comparing the version numbers of incoming packets with those previously received, nodes can effectively detect and discard duplicate packets. This process reduces unnecessary processing and conserves valuable network resources.
Moreover, version numbers streamline the distribution of packets in sequential order. They enable nodes to arrange packets correctly as they travel through the network, ensuring that data is delivered to applications or higher layers in the intended sequence. This maintains the consistency and integrity of the network's information flow, allowing data to be processed and acted upon accurately.
Additionally, version numbers aid in the detection of packet loss and retransmission. Nodes monitor the version numbers of the incoming packets to identify any breaks or gaps in the sequence, indicating potential packet loss. Armed with this information, nodes can initiate recovery or retransmission mechanisms to ensure dependable data transfer [26].

3.7.1 Version Cache Implementation for Anomaly Detection in Mesh Networks

In the implementation of our application, we have devised a version cache mechanism to facilitate anomaly detection within the IoT mesh network. The version cache is essentially an array with 256 indexes, where each index corresponds to a specific mesh index, and the value at each index represents the version number associated with that mesh index.The flow of this version cache implementation is outlined in Figure 3.3.
The primary rule governing our version cache is that every index should increase by 1, which applies universally to all packets within the network. Before overwriting the previous version in the version cache, we rigorously check that the new packet's version is exactly 1 larger than the version stored in the cache. Any deviation from this rule indicates the presence of an anomaly, which is promptly reported for further investigation.
Python coding was employed to dynamically extract and parse packets within our implementation. Through efficient Python scripting, we iterated through the packet stream, processing each packet in real-time to extract relevant information. This approach facilitated dynamic management of the version cache and enabled timely anomaly detection within the IoT mesh network.
The implementation process involves the following steps:

- **Iteration through Packet Stream**
  We iterate through the stream of packets received from the IoT mesh network, processing each packet individually to extract relevant information.
- **Updating the Version Cache**

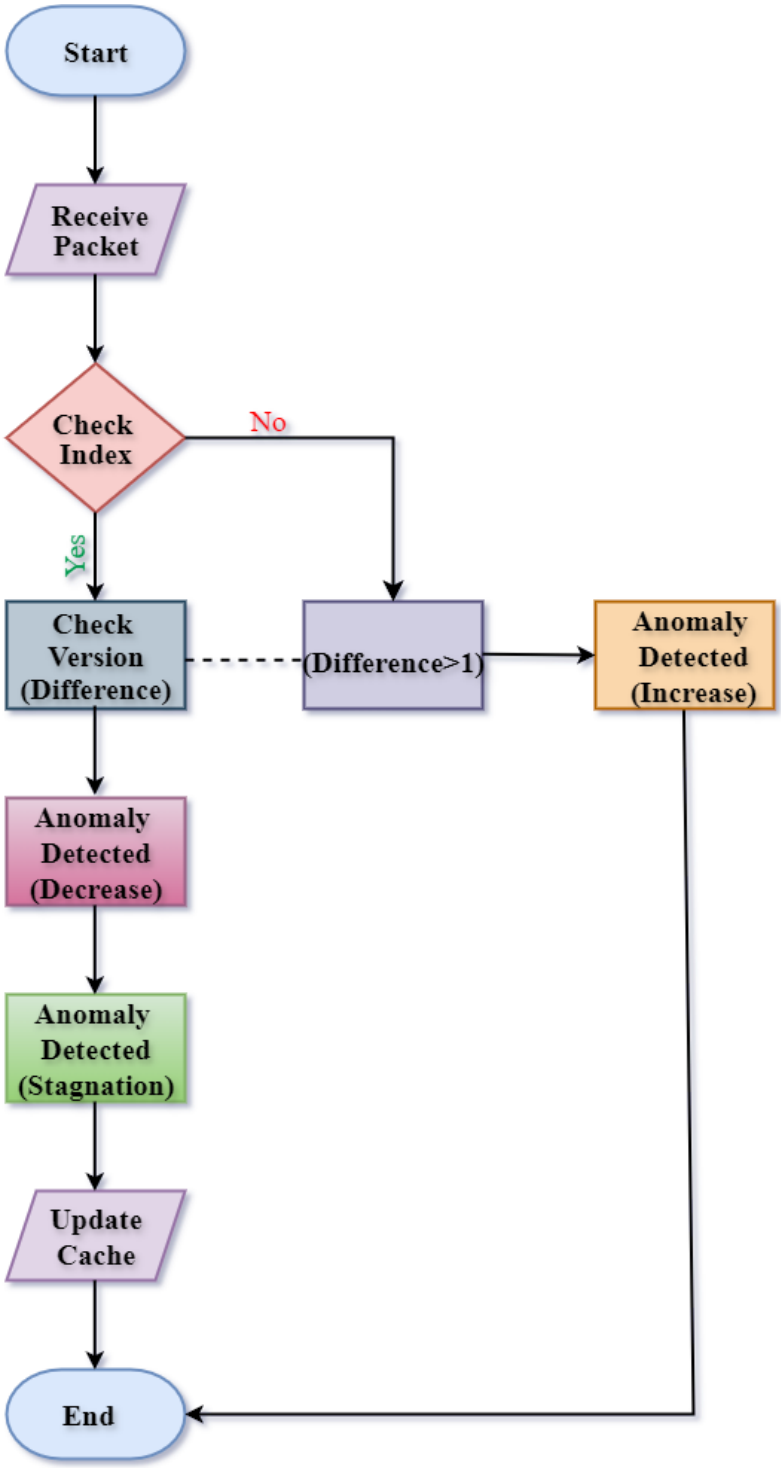


Figure 3.3: Version Rule Set Flow

For each packet, we verify if the corresponding index is already present in the version cache. If not, we add the index to the cache along with its associated version.

- **Anomaly Detection**
  Upon encountering a packet with a known index, we compare its version with

the version stored in the cache to identify any discrepancies. Anomalies are flagged if the version increment exceeds the expected value or if the version decrements unexpectedly.

- **Updating the Cache**
  Regardless of anomaly detection, we update the version and payload information in the cache for the current index, ensuring that the cache reflects the latest state of the network.

This version cache implementation serves as a critical component of our anomaly detection system, enabling real-time monitoring and identification of irregularities within the IoT mesh network.

3.8 Enhancing Reliability and Efficiency in Mesh Networks

A node light table is a type of data structure used in Internet of Things mesh networks to hold data regarding the status and configuration of individual nodes, especially lamps. In addition to variables like index, status, level, and time, this table usually has an expiration time parameter defined to guarantee data currency. On the other hand, the level comparison of packages refers to the process of comparing the state and level information received from different nodes or packages within the mesh network. This comparison is vital for ensuring consistency and reliability in the network operation.
In this context, an output announce is a particular command or packet that a light node sends in response to a controller order. This command, which is essential for preserving network synchronization and coordination, usually includes information about the lamp's current condition and level.
These parts are essential to an IoT mesh network because they make precise synchronization, monitoring, and troubleshooting possible. Efficient management and control are made possible by the node light table, which offers a decentralized store for node information. In the meantime, level comparison checks received data against expected values to guarantee consistency and integrity within the network. In light of the nodes' current state, output announces provide controllers with real-time status updates that let them decide what to do and how to do it.

3.8.1 Dynamic Lamp State and Level Comparison Rule Set

The implemented rule set is designed to monitor and compare the state and level of the lamps in an IoT mesh network. The process, illustrated in Figure 3.4, begins with receiving a command group output state and level from the lamp, which includes information about the current state and level. If the length of the payload is 4, indicating an output announce, the state, level, and time are extracted and added to a level comparison data frame with an expiration time of 5 seconds.
Subsequently, a node light table is created to store the lamp's state and level. If the command is 0x0098 and a particular lamp index, the state and level are extracted from the payload and added to the node light table. If the length of the level

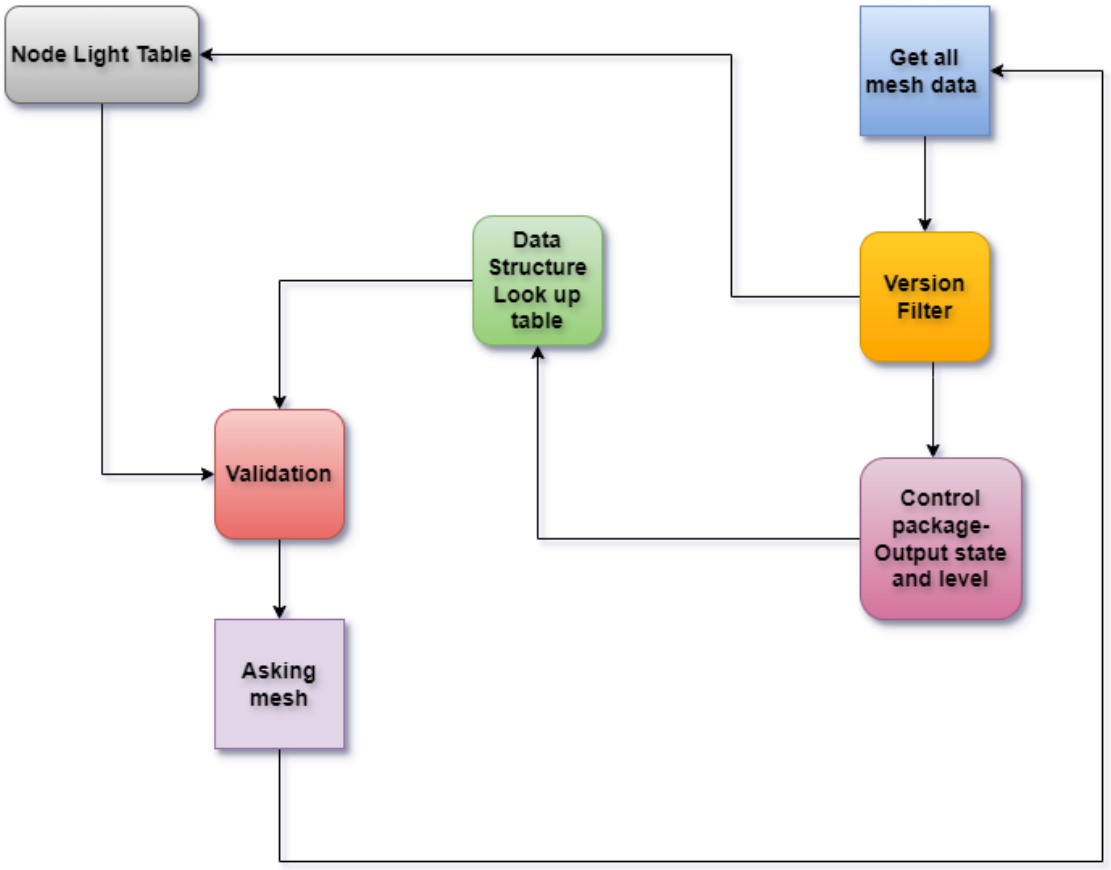


Figure 3.4: Dynamic lamp state and Level Rule Set Flow

comparison list is less than one, a control package is manually provided through the control terminals(i.e. App, console, switch, etc.) to obtain the lamp's state and level, which are then added to the level comparison list.

Validation checks are performed by comparing the current time with the expiration time in the level comparison list. If the expiration time is exceeded, the index is marked for removal and the validation fails. Additionally, if the level of the lamp matches the level in the level comparison data frame, the level validation is successful, and the index is added to the removal list. Otherwise, the level mismatch is visualized.

To simulate the scenario where the output announce packet is missed, the get command is used to manually retrieve the payload information for indexes. If the retrieved level matches the level in the comparison data frame, it confirms that the output announce was not received but matches the manually checked level.

Finally, the indexes marked for removal are iterated over, and the corresponding comparisons are removed from the level comparison list. This iterative process ensures the accuracy and efficiency of the lamp state and level comparison mechanism.

3.9 System Time Synchronization in IoT Mesh Networks

In IoT mesh networks, the System Time announce is a fundamental method for synchronizing network node clocks. This announcement sends the current system time from a certain node to others, ensuring that timekeeping is constant throughout the network. The importance of system time synchronization in mesh networks cannot be emphasized because it enables exact coordination and orchestration of network activity. Synchronized clocks allow nodes to carry out activities like data transmission and event triggering in a coordinated manner. Furthermore, system time synchronization is critical to the reliability and efficiency of network protocols and algorithms. Many mesh networking technologies rely on exact timing information to determine topology, route packets, and manage resources. Synchronized system time boosts network performance, lowers latency, and increases reliability.

Accurate time-stamping of data packets and events is also required for troubleshooting and forensic analysis in mesh networks. Synchronized time allows network managers to reliably correlate events across nodes, which aids root cause investigation and problem resolution. Overall, the System Time announce and synchronization mechanism is critical for ensuring network integrity and performance. They enable the installation of advanced network operations and applications, hence ensuring the smooth operation of IoT mesh networks.

3.9.1 System Time Rule Set for Time Synchronization

Our System Time Rule Set governs time synchronization in IoT mesh networks. As depicted in Figure 3.5, it dictates that a System Time announce occurs every 5 minutes, with a random spread factor of approximately 100 seconds. When a node detects a time delta of less than 8 seconds from its own time upon hearing a Time announce, it reschedules its own announce for another period. If a node does not have its own time, it accepts the time from the announce. However, if the node already has its own time and the time delta is greater than 8 seconds, it evaluates its internal time certainty. If the internal time is inexact and the announce time is exact, the node accepts the announce time. The time exactness is observed with the help of the system time exactness flag. Conversely, if the internal time is exact or has the same certainty as the announce, and the announce time delta is greater than 15 seconds, the node accepts the new time.

This rule set ensures accurate time synchronization across the mesh network, promoting reliable network operation and efficient coordination of network activities. It plays a vital role in ensuring the reliability, efficiency, and security of IoT mesh networks by facilitating accurate time synchronization among network nodes.

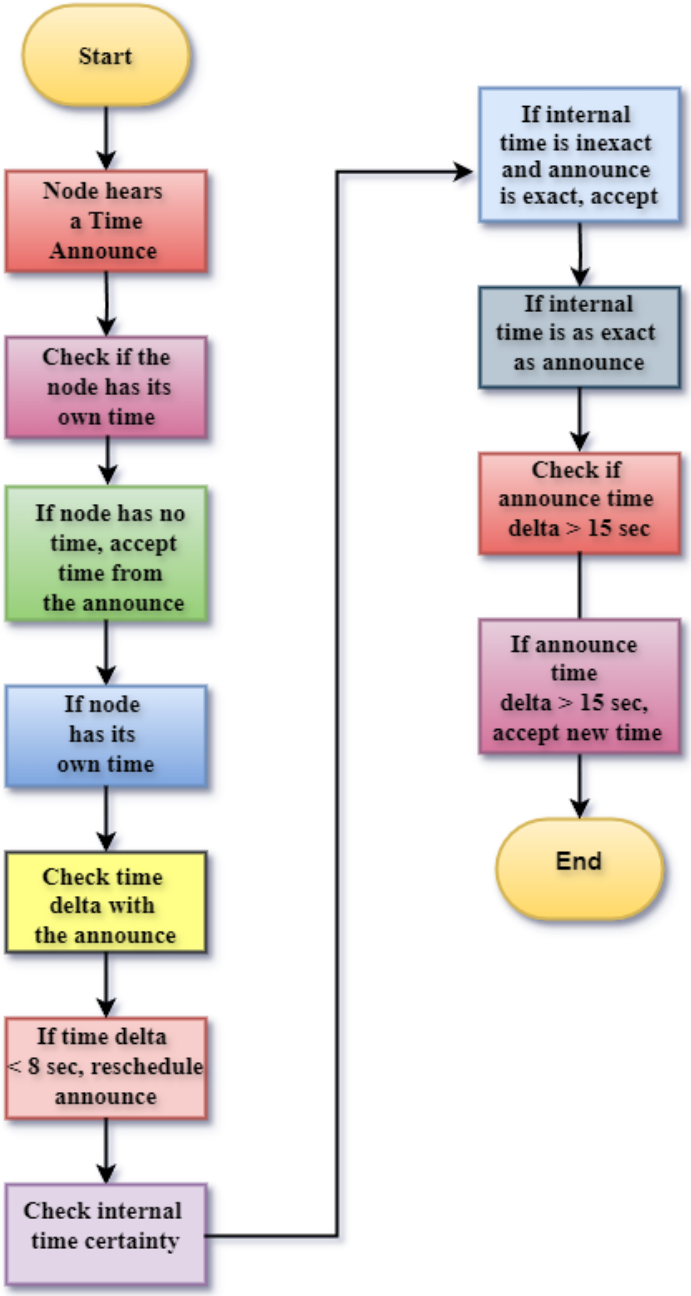


Figure 3.5: System Time Announce Flow

3.10 Enhancing Device Management Efficiency with Scenes in IoT Mesh Networks

In an IoT mesh network, a scene is a predetermined configuration or state that can be deployed to several devices with a single command. These scenes make device control easier, especially when multiple devices must adjust to specific configurations at the same time. For example, in a smart house, a "Movie Night" scenario may dim lights, change the thermostat, and engage entertainment equipment with a single

command. Scenes make user interactions easier, minimizing the need for manual modifications each time. This improves the user experience while also increasing network management efficiency.
Scenes also help save energy by aligning setups with individual usage patterns or preferences. For example, a "Good Morning" scene may activate gadgets at the start of the day, and a "Goodnight" scene could remove superfluous equipment at night. Scenes boost user pleasure by making device management easier. Furthermore, scenes enable quick responses to changing needs or activities, increasing the adaptability of IoT ecosystems. They also allow users to make bespoke setups based on their lifestyle and interests. Overall, scenes ease device administration, improve user experience, and optimize energy utilization in IoT mesh networks.

3.10.1 Scene Validation Rule Set for IoT Mesh Networks

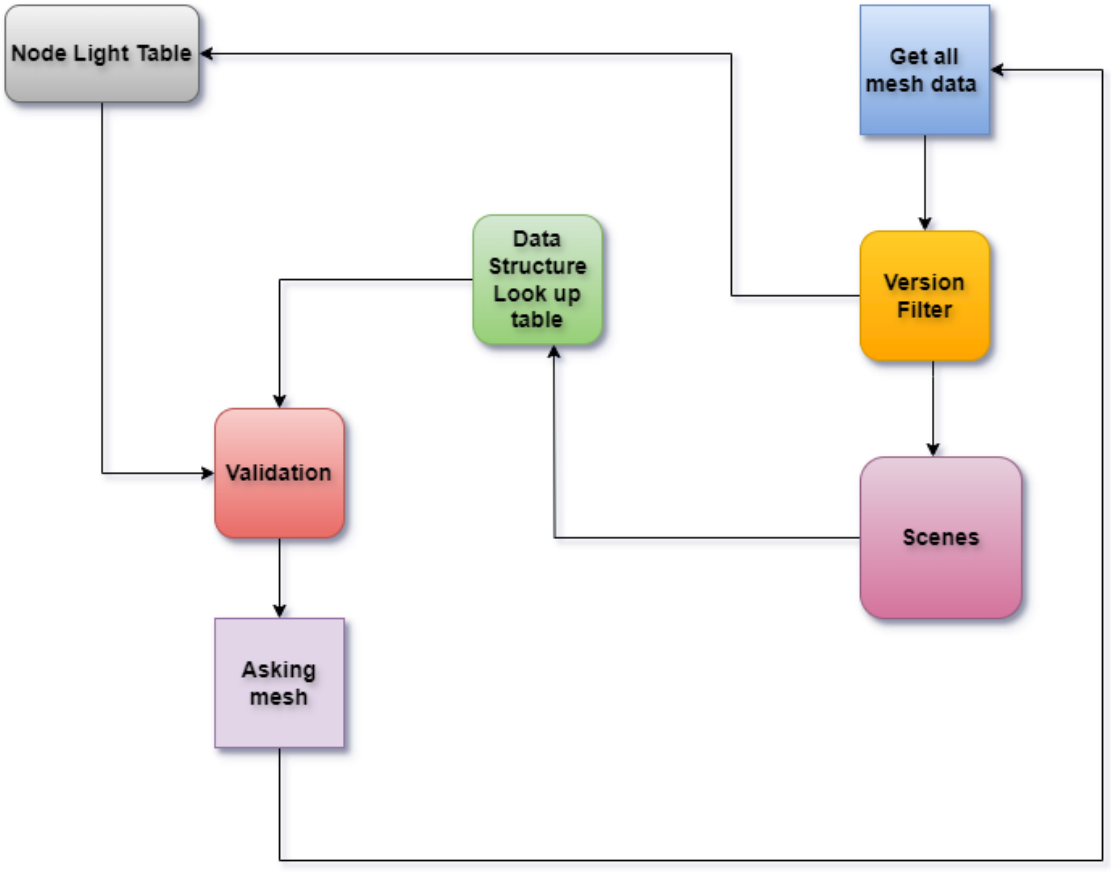


Figure 3.6: Scene Rule Set Flow

The Scene Validation Rule Set for IoT Mesh Networks facilitates the validation of scenes configured in the Plejd ecosystem. This rule set comprises several critical components that ensure correct and reliable scene execution. As illustrated in Figure 3.6, the process begins with storing all configured scenes in a Scene Data Structure Lookup Table. Each scene has its own set of indices, along with associated states and levels.

When a scene is triggered, the rule set compares the scene package with the Scene Data Structure Lookup Table. The purpose of this comparison is to ensure consistency in the statuses and levels of the corresponding indexes. If the state and level of the indexes match those in the Scene Lookup Table, the scene validation is deemed successful. This confirmation indicates that the configured scene has been correctly executed over the specified indexes. The validation process provides valuable insights into the responsiveness and accuracy of scene execution within the mesh network, allowing users to identify and rectify any discrepancies or irregularities in scene performance as necessary.
Furthermore, the rule set ensures that astronomical events in scenes are accurately implemented based on the specified time instances. This feature enhances the precision and reliability of scene-based automation in IoT mesh networks.

4
Results

In presenting the results, we maintain a systematic approach aligned with the methodologies outlined in Chapter 3. Each test scenario is meticulously documented to ensure clarity and consistency. The results are presented in a structured manner, with careful consideration given to the order of presentation.
The test conditions, including duration and key parameters, are clearly outlined for each scenario to provide context for the analysis. By leveraging the Plejd-Home test setup, we conduct a comprehensive evaluation of the performance of the rule sets under various conditions.
Furthermore, we intentionally introduce manual conditions to simulate real-world scenarios, allowing for a thorough assessment of system behavior and performance. Throughout the results section, the dynamic nature of Grafana visualizations enhances the interpretability of the findings, providing valuable insights into the adherence of the system to specifications and the identification of anomalies.
Overall, the results are presented in a structured manner to facilitate understanding and interpretation, enabling readers to derive meaningful conclusions from the data presented.

4.1 Plejd-Home Test setup

Figure 4.1: Plejd-Home Test Setup

A test setup named Plejd-Home has been meticulously developed at the Plejd office, showcasing a diverse array of lights sourced from both in-house development and various manufacturers. The setup as depicted in fig. 4.1 boasts a combination of strip LED lights, normal light bulbs, ceiling lights, outdoor lights, spotlights, colored temperature lights, and more, providing a holistic representation of real-world lighting scenarios.
Comprising over 200 devices, all interconnected within a single mesh site, Plejd-Home serves as a robust platform for testing both hardware and firmware aspects of Plejd devices. Despite the typical limitation of around 80 devices for optimal operation, Plejd-Home boldly pushes boundaries by accommodating up to 256 devices in a single mesh site.
Rigorous testing procedures are employed to evaluate the performance of the mesh network under various conditions, with a specific focus on RF parameters. A dedicated tool for analyzing mesh behavior is under development, offering insights into the intricate workings of the network.
Controlled by a gateway, which acts as a mesh node with LAN connectivity, Plejd-Home provides remote access via an app interface, facilitating seamless monitoring and management of the mesh devices.
The implementation of a live analysis tool allows for real-time monitoring of each node within Plejd-Home, enabling in-depth analysis of their behavior over time. The results obtained from the extensive testing scenarios conducted within this setup are detailed in the forthcoming sections, shedding light on key findings and implications for mesh network performance.

4.2 Version Cache Rule Set Verification

In this verification, we aim to measure the consistency and reliability of version numbers within the mesh network. A version cache, capable of accommodating 256 indexes as detailed in the methodology section 3.7.1, will be maintained. This cache is crucial for ensuring that the system can accurately track and confirm the consistency of version numbers for mesh packages across all nodes in the network.
To achieve this, the Python Live analysis tool will be deployed on 200 devices within the mesh site. This tool is important for its ability to provide real-time detection and visualization of any anomalies in version numbers. By continuously monitoring and plotting version numbers over time for each node on Grafana, the tool will facilitate immediate identification of discrepancies. This continuous measurement will allow us to understand the behavior of version numbers under various conditions, including both synthetic and natural environments.
The measurement of these version numbers and the subsequent analysis are intended to ensure the reliability and robustness of the versioning system. This process is critical for identifying and addressing potential issues early, thus contributing to the overall stability and security of the network.

4.2.1 First Test Case Evaluation: Version Behavior Analysis

During this evaluation, we monitored the version behavior of the lamp nodes installed in the mesh test setup over a 24-hour period. Separate queries were created for each node in the mesh, allowing us to visualize version numbers over time. As shown in Figure 4.2 subplot 2, ideally, a linear curve was observed, with the version number incrementing by 1 for each new mesh package sent. These incrementing packages prompted actions based on their commands when received by nodes in the mesh, while packages with the same version number or a significantly decremented version were ignored.
Figure 4.2 illustrates the offline visualization of the version behavior analysis. Subplot 1 shows the creation and execution time of the mesh packages, and Subplot 3 depicts the version status for the particular mesh package. These visualizations provide a detailed analysis of how version numbers change over time and their correlation with the mesh package's actions. It was noted that whenever a device in the mesh was powered off and then turned on again, its version number reset to one, a phenomenon that occurs only after a hard power reset. However, when the lamp nodes were powered down by mesh commands, the physical power was still on, which allowed the device to continue from the same version number buffer for incoming commands. This behavior follows the rule set outlined in section 3.7.1.
Additionally, a noteworthy irregularity was observed, wherein the version numbers of numerous nodes inside the mesh gradually decreased over time. This issue persisted throughout the testing period and was subsequently brought to the attention of the firmware team for further analysis.
For a real-time view of these results, refer to Figure 4.3. The Grafana visualization provides a comprehensive dashboard that complements the offline plots. This figure presents similar data as in Figure 4.2, but in an interactive format that allows for

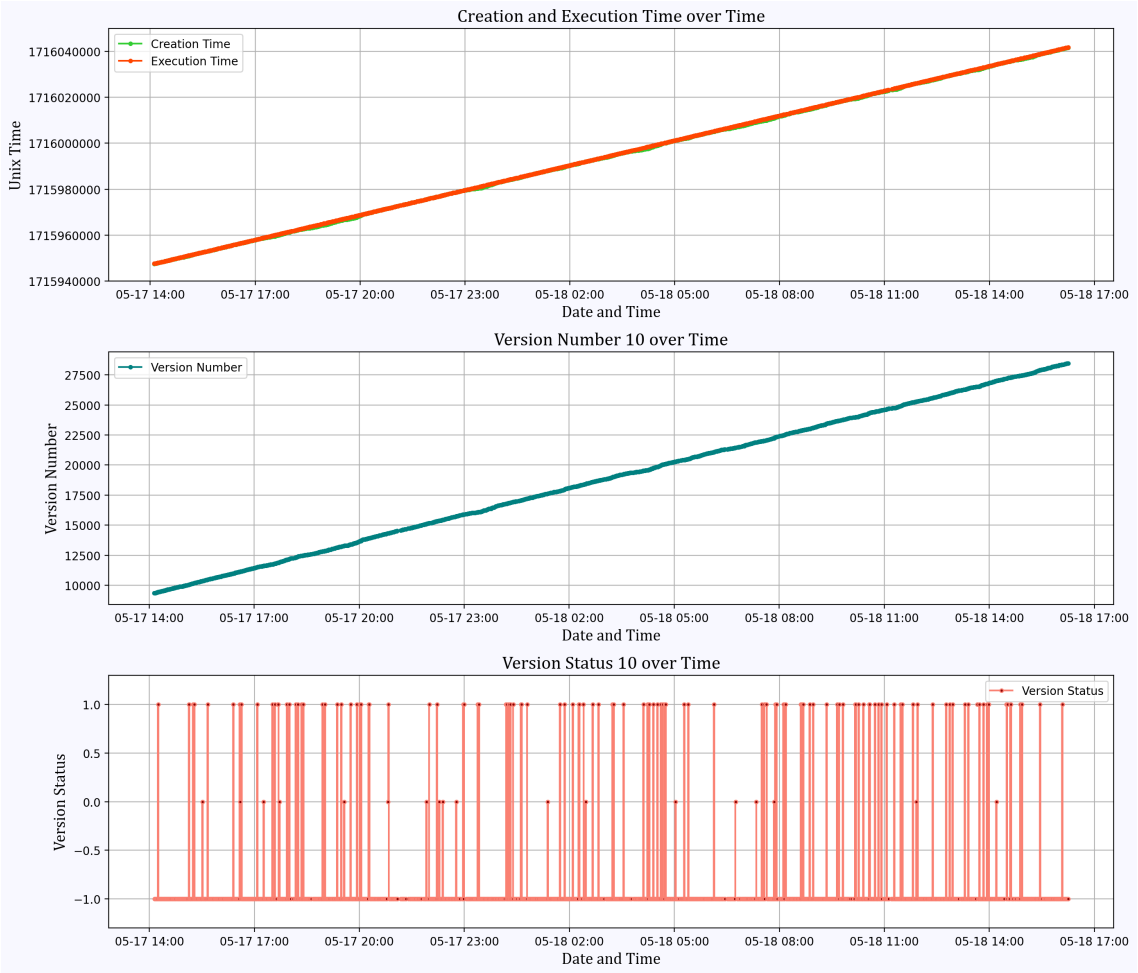


Figure 4.2: First Test Case Evaluation: Version Behavior Analysis offline visualization

real-time monitoring and analysis. Figure 4.3 boasts five distinct visualizations. Among these, two plots focus on metadata visualization: one showcases the total number of mesh packets analyzed after filtering out rebroadcast packages, while the other delineates the time response in seconds for executing mesh packages. This time response is computed as the disparity between packet creation and reception times and is presented in the form of a gauge and a statistical value. The remaining three plots delve into time series visualizations. The third plot delineates the creation versus execution time for packages received by node 21. Additionally, the fourth plot tracks version numbers over time for all nodes, depicting the behavior observed during the first test case evaluation while the fifth plot reveals the outcomes of version validation. Here, a value of +1 signifies a version incremented by more than 1, 0 denotes no increment, and -1 indicates a version decrement.

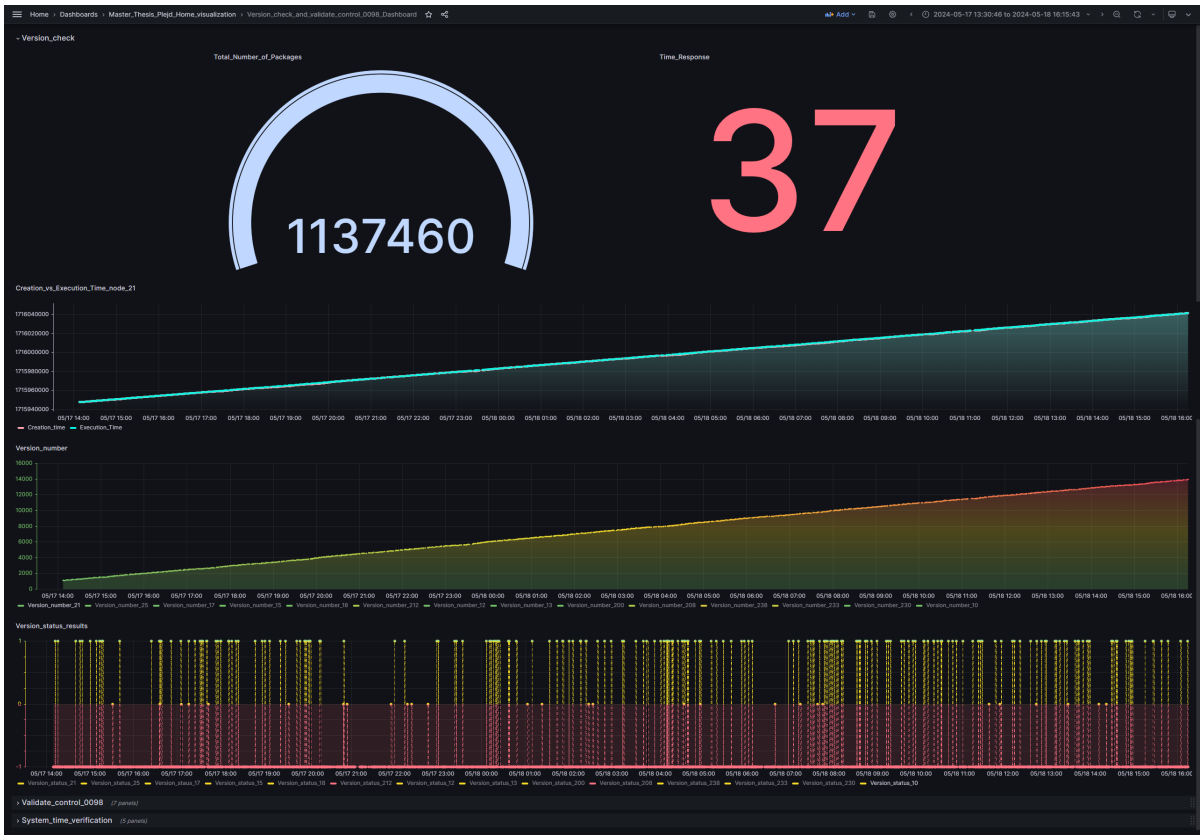


Figure 4.3: First Test Case Evaluation: Version Behavior Analysis

4.2.2 Second Test Case Evaluation: Version Invariance in Node 10

To evaluate version invariance, we modified the firmware of a lamp node in the mesh network to ensure its version number remains constant for all mesh packages sent from this node. This modification was applied to a single lamp node, numbered 10. Normally, each node in the mesh reacts only to messages with the latest version numbers. Therefore, with the version number fixed, packages sent from node 10 were discarded by every other node in the mesh, as these packages did not meet the requirement for an updated version number.

The results of this test are presented in Figure 4.4 which show the offline visualization of the version invariance in node 10. Subplot 1 illustrates the creation and execution time of the mesh packages, Subplot 2 displays the version numbers, and Subplot 3 indicates the version status of the particular mesh package. These visualizations help understand the fixed version number's effect on the communication within the mesh network. In Figure 4.5, the Grafana visualization highlights the real-time monitoring of node 10.

When node 10 was designated as the primary node connected to the app, all its outgoing packages were immediately rejected, resulting in no operations being performed. We kept node 10 in the mesh for a limited period and monitored the version behavior. It was observed that the version number of the packages it sent remained constant, as expected.This behavior is visualized in Figure 4.4 subplot 3.

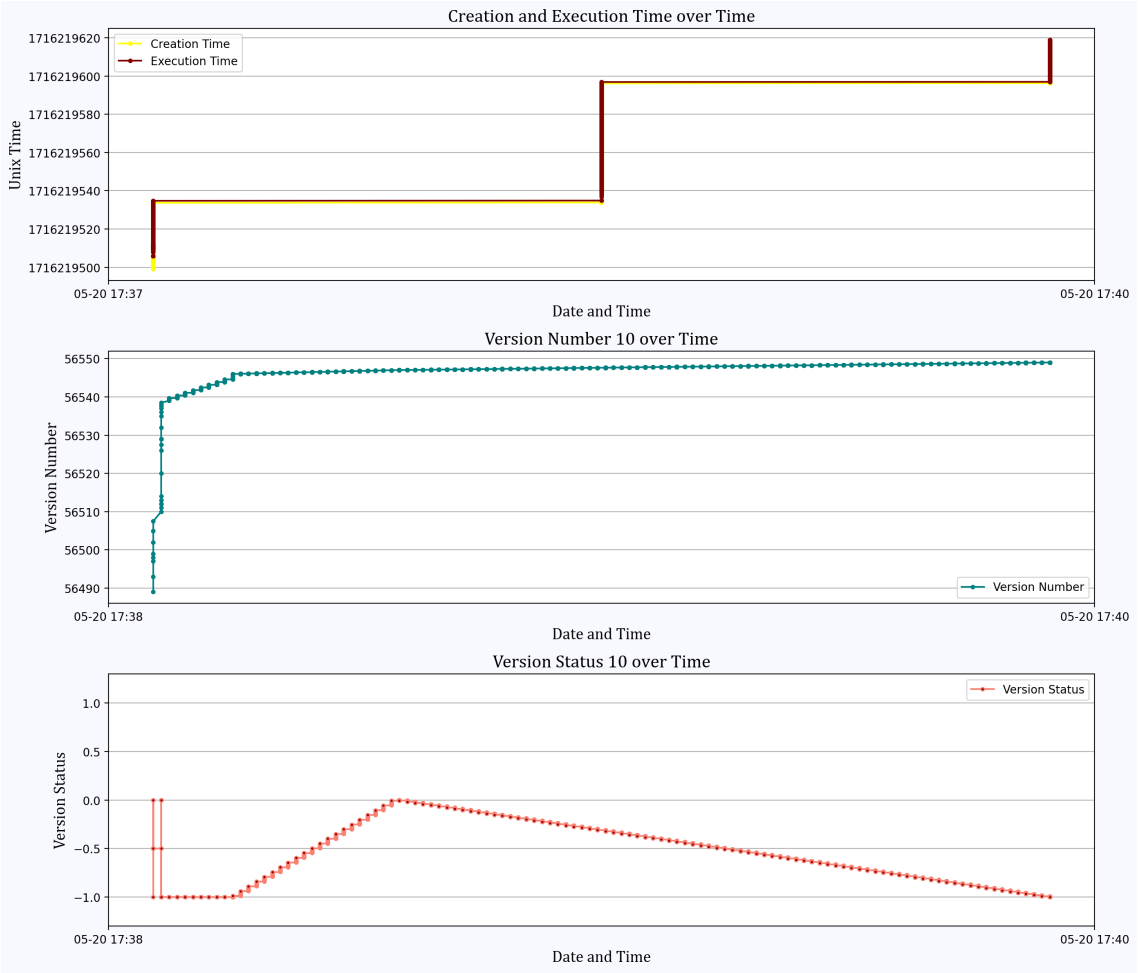


Figure 4.4: Second Test Case Evaluation: Version Invariance in Node 10 visualization offline visualization

These results demonstrate the impact of a fixed version number on mesh communication. The version number for node 10 remained the same throughout the test period, confirming our intentional creation of this condition and enabling us to visualize the node's behavior in real-time on Grafana. The evaluation successfully shows the robustness of the version handling rule set in real-world scenarios. The Grafana visualization (Figure 4.5) provides a comprehensive dashboard that complements the offline plots. Figure 4.5 features three distinct time series visualizations. The first plot delineates the creation versus execution time for packages received by node 10. The second plot tracks version numbers over time for all nodes, depicting the behavior observed during the first test case evaluation. The third plot reveals the outcomes of version validation, where a value of +1 signifies a version incremented by more than 1, 0 denotes no increment, and -1 indicates a version decrement.

This evaluation successfully demonstrated the impact of a fixed version number on mesh communication and verified the robustness of the version handling rule set in real-world scenarios.

Figure 4.5: Second Test Case Evaluation: Version Invariance in Node 10

4.3 Level Validation Rule Set Verification

The level validation procedure is essential for ensuring that the state and level imposed on a lamp node are applied accurately and consistently. This validation is crucial because it guarantees that the lighting system responds correctly to control signals, maintaining system reliability and user satisfaction. The procedure is detailed in the methodology section 3.8.1 and involves adding a control package that can be activated by various triggers such as an application, button clicks, or a sniffer console.
The core of this procedure lies in using the Node Light Table Look-Up Table to verify that the status and level specified in the control package are met whenever an output announce is received. Successful verification is indicated by matching values; any discrepancy is reported as a failure, potentially due to a timeout. This validation process is vital as it ensures the system's accuracy and responsiveness, which are key factors in maintaining the integrity and reliability of the mesh network.
To thoroughly evaluate this process, the Python live analysis tool was deployed across 200 devices within the PlejdHome mesh network. This tool continuously monitors and performs level validation as devices are triggered by external stimuli. Additionally, an external script periodically triggers individual lamps or groups of lamps to change their state and level, simulating real-world usage. The Python live analysis tool then validates these changes by checking the state and level for control packages activated by the script.
This extensive testing is conducted to monitor the lamps' behavior over time and assess the firmware's resilience. By emulating real-world conditions, the testing process aims to compare the actual performance of the lamp firmware with the specified parameters. The importance of this thorough verification process lies in its ability to confirm the correctness and dependability of the mesh network's level validation process, ensuring that the system functions as intended in practical scenarios.
The results from this validation process provide critical insights into the system's reliability and effectiveness, highlighting areas that may need improvement and confirming aspects that perform well. This comprehensive evaluation helps in refining the system to better meet user expectations and operational demands, ultimately enhancing the robustness and user satisfaction of the PlejdHome mesh network.

4.3.1 First Test Case Evaluation - 24-Hour Level Validation

To evaluate the level validation of every node in the mesh network, a Python script was executed for 24 hours during the first test case evaluation. This involved meticulously monitoring all theoretical validation conditions as outlined in the methodology section 3.8.1. Throughout the test, it was observed that whenever a group output status and level command was activated without an output announce, the packet was added to the data structure lookup table. The node light table was updated in accordance with the corresponding state and level, which were then displayed on Grafana. The metadata visualization in Figure 4.6 presents two plots: a bar gauge and a pie chart. Both plots display the same information, including the total number

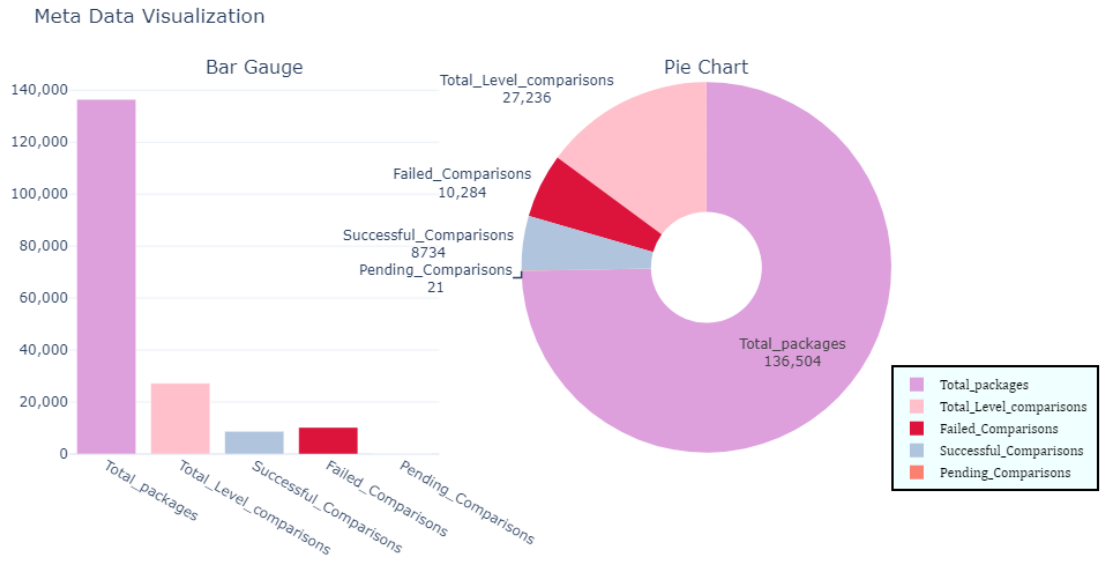


Figure 4.6: First Test Case Evaluation: Metadata visualization Offline Visualization

of mesh packages calculated after excluding rebroadcast packages, the total number of level validation packages, the number of successful level validations, pending validations, and failed validations. This provides an immediate overview of the system's validation status. The results of this evaluation are presented in Figure 4.7, which contains four subplots. Subplot 1 displays the node light table intensity level for node 21, Subplot 2 shows the level validation level for node 21 as added by the control package, Subplot 3 illustrates the state of node 21, and Subplot 4 indicates the result of the validation. The values in the node light table (Subplot 1), the node level (Subplot 2), and the node status (Subplot 3) frequently aligned as predicted during the testing period, indicating that the system was functioning as intended. However, during periods of high message volume, some validations failed, certain comparisons did not show up, and some output announces were not visualized. To address these issues, the implementation includes a mechanism to send a get request to the light node, allowing its level and current condition to be verified. Despite the challenges posed by message flooding, this additional step helped ensure accuracy in the validation process. For a real-time view of these results, refer to Figure 4.8. The Grafana visualization provides a comprehensive dashboard that complements the offline plots, enabling real-time monitoring and analysis of the level validation rule set. The Grafana dashboard consists of seven plots, categorized into three metadata visualizations and four time series plots. The metadata visualizations include a gauge for the total number of mesh packets analyzed and a digital bar gauge and pie chart for validation results, providing an overview of the system's validation status. The time series plots include the Node Light Table Intensity Levels, Level Comparison Rule Set Intensity Levels, Level Comparison Rule Set States, and Level Comparison Rule Set Results. These plots offer detailed insights into the intensity levels, state changes, and validation outcomes over time, ensuring the system's performance and

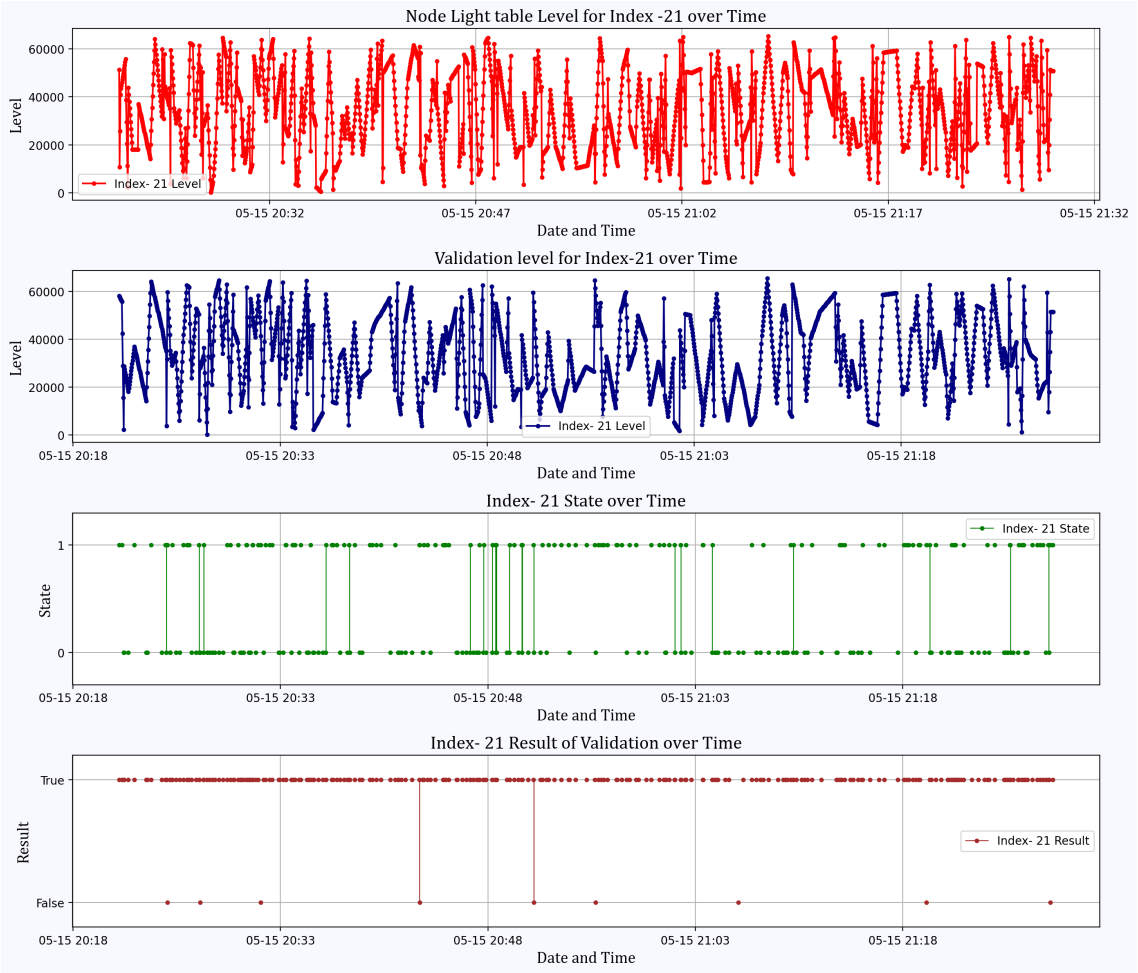


Figure 4.7: First Test Case Evaluation: Level Validation rule set verification by 24- hour monitoring and offline visualization

validation processes are accurately monitored and analyzed.

4.3.2 Second Test Case Evaluation: Handling Power Outage for Lamp Node

In the second test case evaluation, we focused on a specific lamp node (indexed as 248) within a mesh network of 200 devices. Using a continuous push button trigger, we intentionally reduced the lamp's intensity from 100% to 80% before abruptly cutting off the lamp's main power supply. Adjustments were made through the app to ensure that the lamp would remain off in the event of a voltage drop or power outage.
Throughout the continuous button trigger for lamp 248, control packages were consistently added to the level comparison list in accordance with the rule set implementation. Our approach ensured that only the most recent control package, specifying the final intensity level of 80%, was considered, even amidst a flood of control packages triggered by the constant input. To achieve this, we effectively removed overridden commands 3.6.

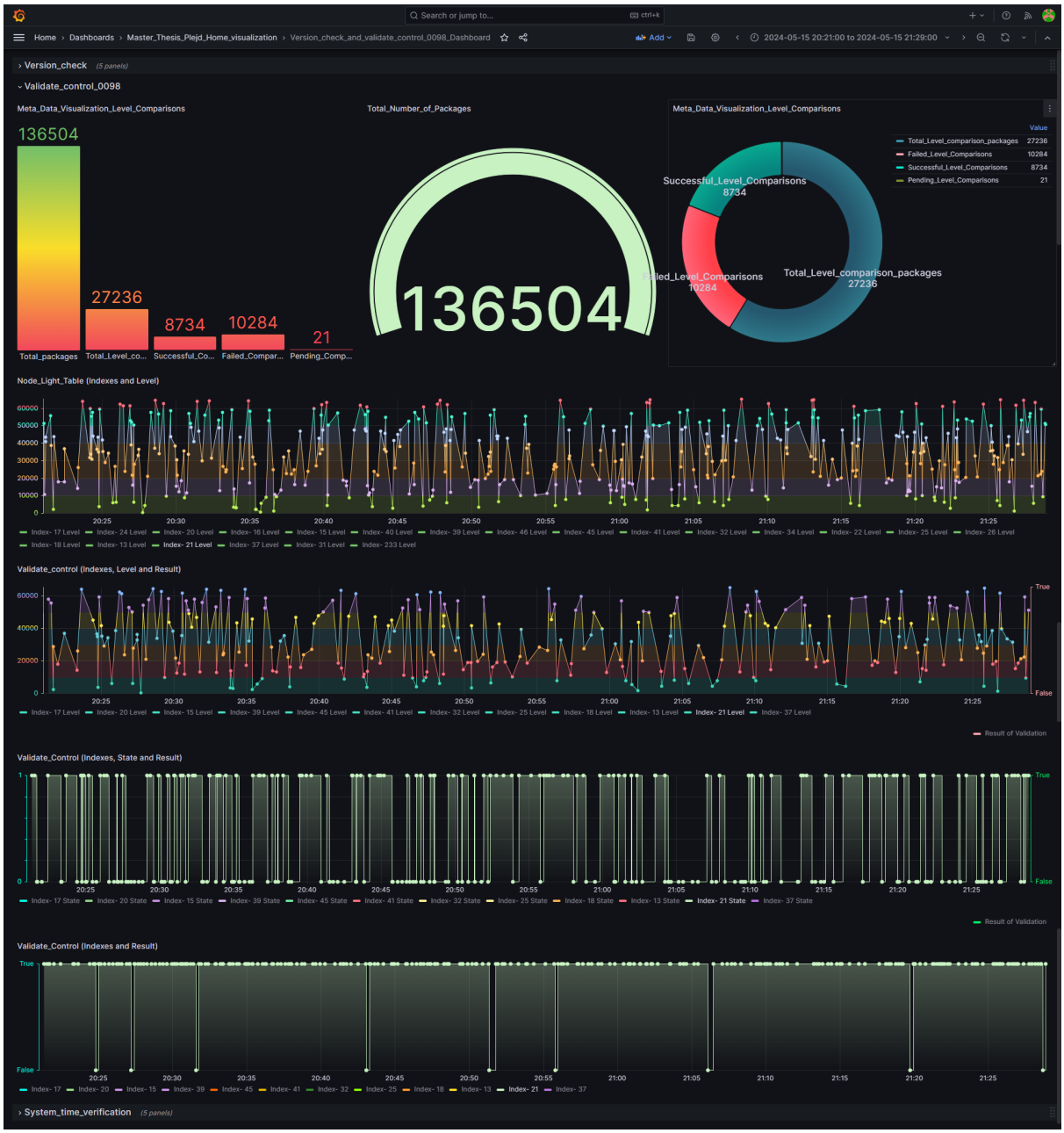


Figure 4.8: Level Validation rule set verification by 24- hour monitoring

Figure 4.9: Second Test Case Evaluation: Level Validation rule set verification by handling power outage for lamp node Offline Visualization

The results of this evaluation are presented in Figure 4.9, which contains four subplots. Subplot 1 displays the node light table intensity level for node 248, Subplot 2 shows the level validation level for node 248 as added by the control package, Subplot 3 illustrates the state of node 248, and Subplot 4 indicates the result of the validation.

Upon restarting the lamp device a few minutes after its power was cut, it remained in the off state as per the programmed parameters. Offline observations, depicted in Figure 4.9 subplots 2 and 3, confirmed that the state and level matched the intended final control package values. However, since the output announcement was not received, the validation process could not be successfully completed. Consequently, the validation result was marked as a failure,as reflected in the offline visualization shown in Figure 4.24 subplot 4.

Although the validation level for index - 248 indicated that the control package addition was successful, the absence of a group output state and level command with the output announce meant that the node light table remained unchanged. This discrepancy led to an inadequate representation of the lamp's state and level in the node light table, ultimately resulting in the validation failure. For a real-

Figure 4.10: Level Validation rule set verification by handling power outage for lamp node

time view of these results, refer to Figure 4.10. The Grafana visualization provides a comprehensive dashboard that complements the offline plots, enabling real-time monitoring and analysis of the level validation rule set. The Grafana dashboard consists of four plots, categorized into four time series plots. The time series plots include the Node Light Table Intensity Levels for node 248, Level Comparison Rule Set Intensity Levels for node 248, Level Comparison Rule Set States for node 248, and Level Comparison Rule Set Results for node 248. These plots offer detailed insights into the intensity levels, state changes, and validation outcomes over time, ensuring the system's performance and validation processes are accurately monitored and analyzed.

4.3.3 Third Test Case Evaluation: Validation of an Absent Node

In this test, we aimed to assess the system's response when a control package is directed to a non-existent node. Specifically, we targeted light node 255, which wasn't part of the PlejdHome mesh test configuration. We sought to understand how the system managed such scenarios by sending a level comparison package to this non-existent node via the app. As expected, since node 255 was absent in the mesh, no output announce was received, resulting in a discrepancy between the node light table and the control package. Consequently, the validation process failed.
The results of this evaluation are presented in Figure 4.11, which contains four subplots. Subplot 1 shows the level validation level for node 255 as added by the control package, Subplot 2 illustrates the state of node 255, and Subplot 3 indicates the result of the validation.
Repeated attempts to send control packages to the missing node consistently led

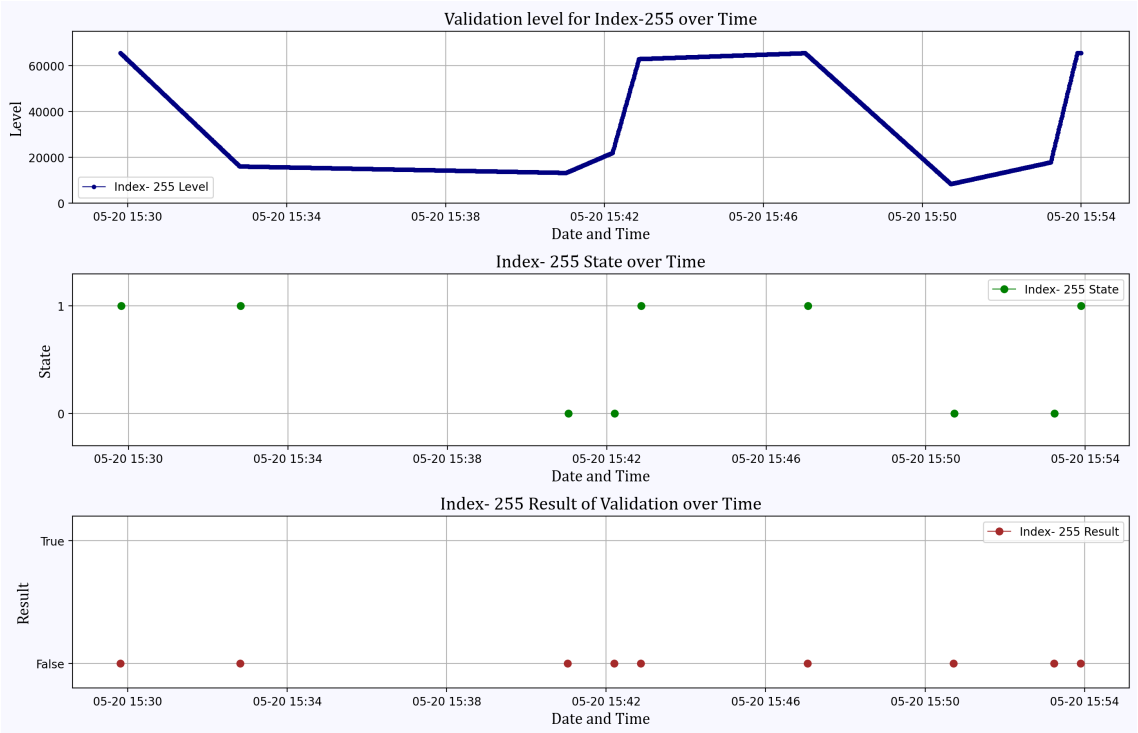


Figure 4.11: Third Test Case Evaluation: Validation of an Absent node offline visualization

to validation failures. Despite changes in the control package's state and level, the absence of an output announce prevented the update of the node light table, causing the validation to fail. This persistent failure, depicted in Subplot 3 of Figure 4.11, effectively validated the non-existence of lamp node 255 within the network.
For a real-time view of these results, refer to Figure 4.12. The Grafana visualization provides dynamic monitoring of the validation process. It consists of four plots: Node Light Table Intensity Levels (which remained unchanged due to the absent node), Level Comparison Rule Set Intensity Levels, Level Comparison Rule Set States, and Level Comparison Rule Set Results for node 255. These plots offer continuous insights into intensity levels, state changes, and validation outcomes, ensuring the system's robustness and accuracy in monitoring network status and validation processes.

4.4 System Time Rule Set Verification

In verifying the system time rule set, our focus was on establishing a comprehensive test setup to assess the behavior of 200 mesh devices, with a Gateway serving as a remote connection node (see Figure 4.1). We utilized commands from the Sniffer console, button inputs, and the mobile application to control mesh devices, enabling us to meticulously observe and analyze the system time behavior of each device using Python script.
Our verification approach entailed defining success criteria, with time deltas falling within the range of -30 to +30 seconds considered acceptable. We assessed the

Figure 4.12: Third Test Case Evaluation: Validation of an Absent node

accuracy of system time packages by parsing Time flags and categorizing them as precise, inexact, or unknown.
Furthermore, metadata analysis provided insights into the total number of system time packages received, relative to the total mesh packages, excluding re-broadcast messages. By recording counts of successful and failing validations, we gained a comprehensive understanding of the mesh network's system time performance.
Individual node searches allowed for detailed examination of time deltas and exactness, facilitating the identification of anomalies and ensuring consistency over time.
Moreover, the system facilitated the issuance of GET requests for system time from specific nodes, enabling the computation of time deltas and real-time evaluation of their current system time.
This verification process was crucial as it ensured that the system time rule set functioned reliably and adhered to specifications across various operational scenarios. By measuring time deltas and assessing the accuracy of system time packages, we could confidently confirm the system's ability to maintain precise time synchronization among mesh devices, which is fundamental for seamless operation and coordination within the network.

4.4.1 First Test Evaluation of System Time Rule Set under Normal Conditions

In the initial assessment of the System Time Rule Set under typical circumstances, a Python script monitored system time on every node within the mesh network over a 24-hour period. The aim was to confirm that the system time rule set operated as intended, adhering to the methodology outlined in Section 3.9.1.
The metadata visualization in Figure 4.13 provided a comprehensive overview of the

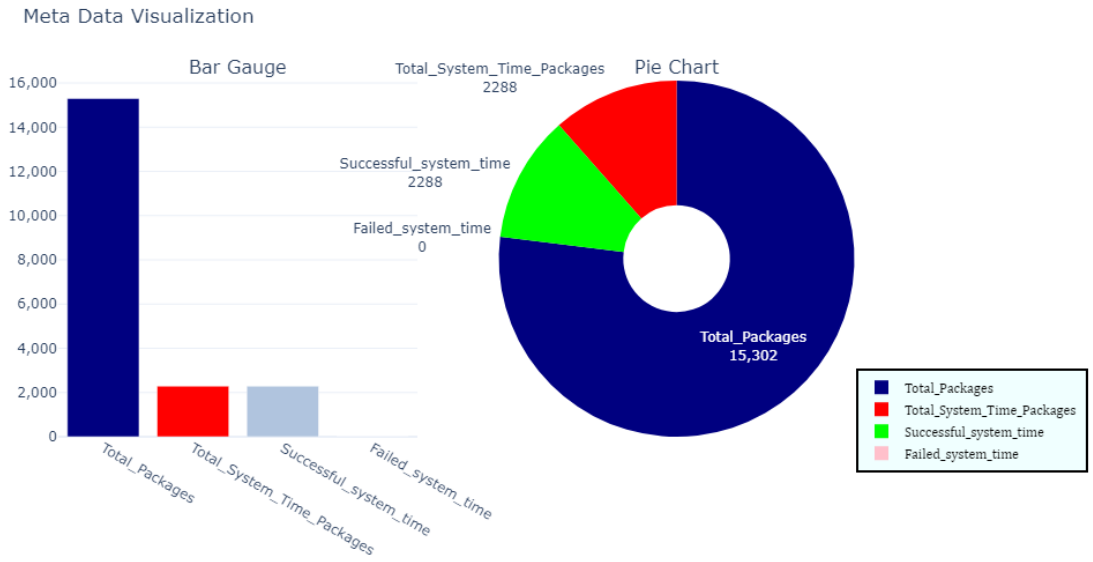


Figure 4.13: First Test Case Evaluation: Metadata offline visualization

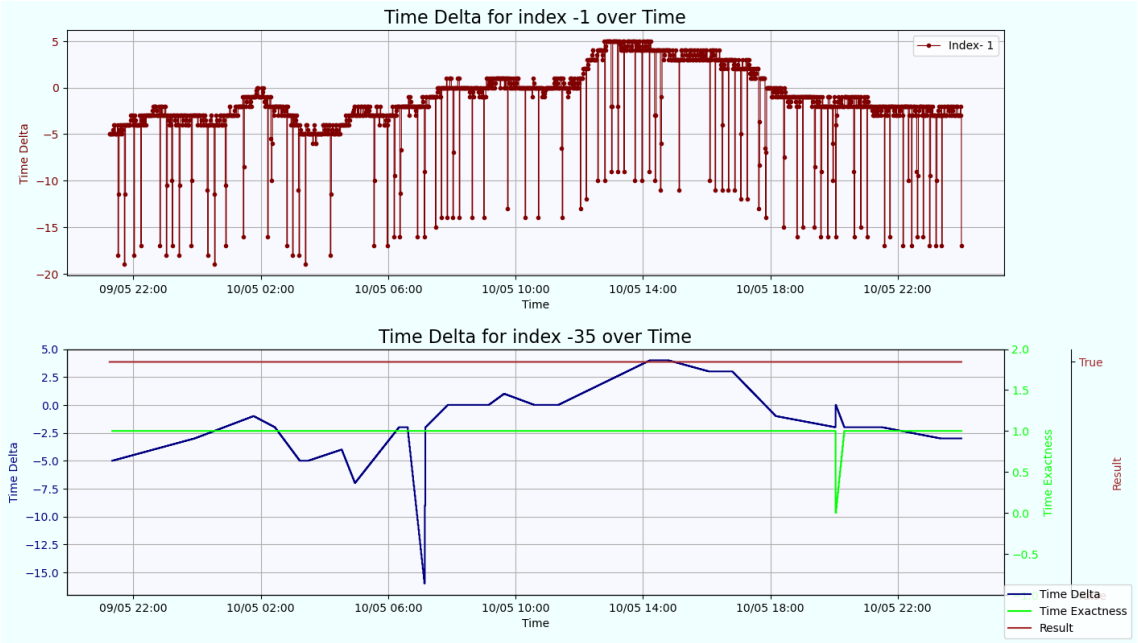


Figure 4.14: First Test Case Evaluation: System Time rule set Verification under normal condition offline visualization

system's time validation status through two plots: a bar gauge and a pie chart. These plots displayed key metrics such as total mesh packages (excluding rebroadcast packages), total system time packages, successful system time validations, pending system time validations, and failed system time validations.

Figure 4.14 depicted the results of the evaluation in three subplots. Subplot 1 illustrated the time delta of index-1 over time, representing the system time populated on index 1, which could resemble any node in the mesh. Subplot 2 showed the time

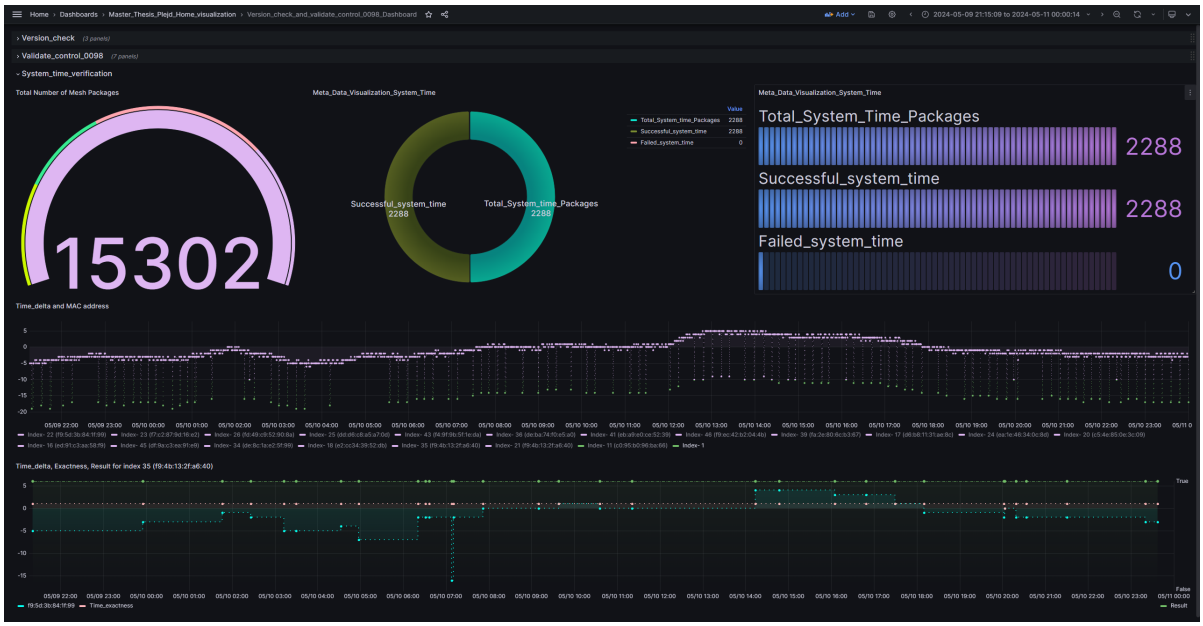


Figure 4.15: System Time rule set Verification under normal condition

delta for index-35, along with the status of the time exact flag and the result of validation.

According to the predetermined criterion, if the time delta exceeded 15 seconds compared to the announce time, the respective node accepted the new time and attempted to mitigate this discrepancy. The visualization confirmed that nodes received more precise time updates and progressively adjusted to reduce time differences over succeeding intervals. Notably, certain nodes exhibited sporadic time delta variations at random intervals, indicating potential anomalies or variability in system time synchronization.

This preliminary test evaluation affirmed the system's capability to dynamically change and synchronize node time throughout the mesh network. It provided valuable insights into the behavior of the system time rule set under typical operating configurations, highlighting its effectiveness in maintaining accurate time synchronization among mesh devices. The Grafana visualizations are shown in the Figure 4.15 which provide real- time insights of the offline visualization as seen in Figure 4.14. The Grafana plots 4.15 provide a comprehensive real-time dashboard for system time verification, consisting of a total of five distinct plots. These plots include three metadata visualizations and two time series plots, each serving specific purposes in the analysis of system time data. The Gauge plot displays the total number of mesh packets analyzed, excluding rebroadcast packages. It provides an overview of the volume of data processed in the system. The Pie chart breaks down the system time command packages into categories, providing a visual representation of: Total system time command packages, Successful system time validations, Pending system time validations, Failed system time validations. The Time series subplot 1 represents the time delta (Y-axis) over time (X-axis). Separate queries are made for every node in the PlejdHome mesh setup. It includes an index that shows which nodes in the mesh have announced the system time. This visualization

helps in tracking the time synchronization accuracy for each node individually and collectively. Plots 4.19 show the Time delta for the node 35 under the normal testing condition. The subplot 2 in Figure 4.15 focuses on the time delta, time exactness flag status, and the result of the verification specifically for node 35 in the PlejdHome mesh setup. It provides detailed insights into how accurately node 35 maintains system time and how often it synchronizes correctly with the time announcements.

4.4.2 Second Test Evaluation: Impact of Manipulating Timekeeping Mechanism on System Time Rule Set

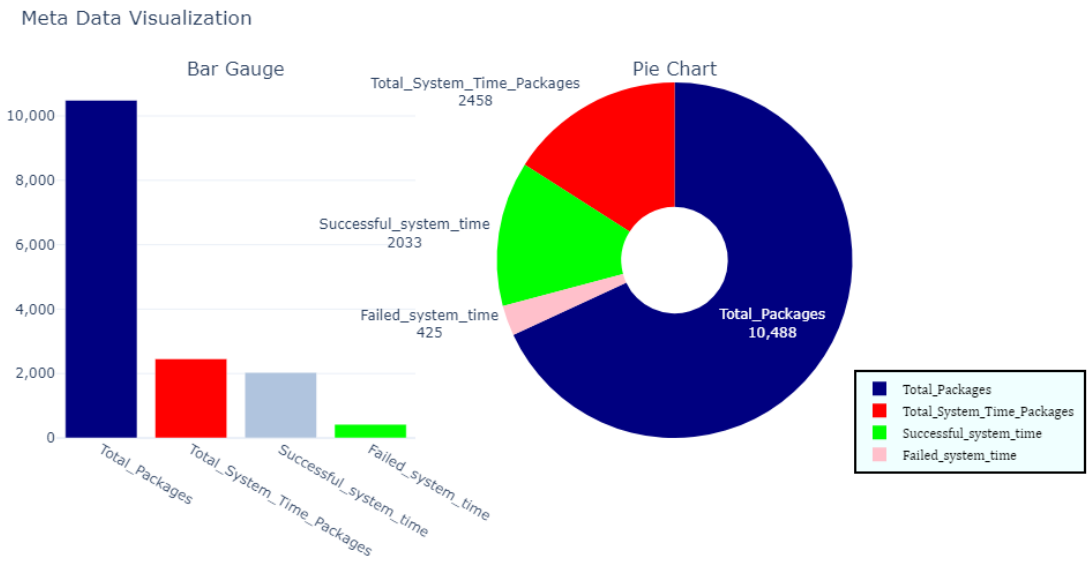


Figure 4.16: Second Test Case Evaluation: Metadata offline visualization

Embedded systems rely on precise timekeeping mechanisms to ensure accurate synchronization and operation. At system startup, timer peripherals are initialized, determining the resolution of time monitoring based on the frequency at which these clocks count. For instance, a timer configured at 50Hz increments its count value 50 times per second, by reading the internal count registers of the timers as they count, the elapsed time is determined. After that, the raw count value is translated into useful time intervals like seconds or minutes. For example, every 50 counts would equal one second if the timer was set to count at 50 Hz. Variables are updated based on timer counts to ensure exact time tracking and accurate time representation.
We deliberately changed the input frequency in our analysis, moving it from the typical 50 Hz to 60 Hz. The timekeeping procedure underwent a significant alteration as a result of this adjustment, which made the counter increase its count value 60 times every second. As a result, the way that raw count values were converted into meaningful time intervals was changed. Under the new frequency setting, every 60 counts was equivalent to one second.
The metadata visualization in Figure 4.16 provided a comprehensive overview of the system's time validation status through two plots: a bar gauge and a pie chart. These

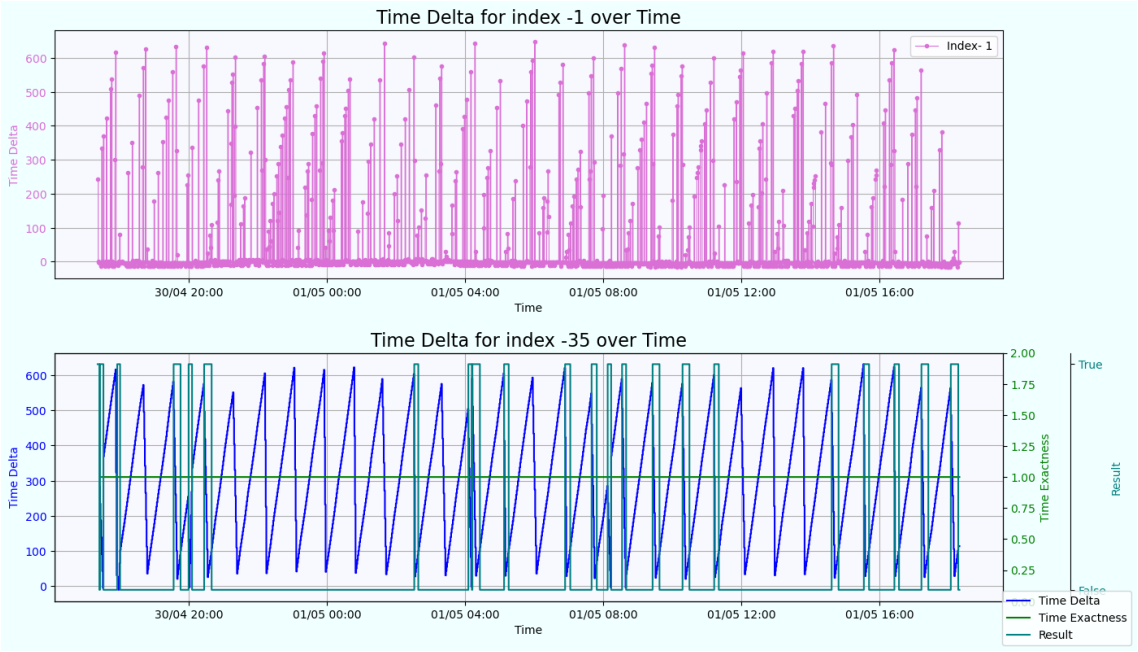


Figure 4.17: Second Test Case Evaluation: System Time rule set Verification by manipulating Timekeeping offline visualization

plots displayed key metrics such as total mesh packages (excluding rebroadcast packages), total system time packages, successful system time validations, pending system time validations, and failed system time validations.

Figure 4.17 depicted the results of the evaluation in three subplots. Subplot 1 illustrated the time delta of index-1 over time, representing the system time populated on index 1, which could resemble any node in the mesh. Subplot 2 showed the time delta for index-35, along with the status of the time exact flag and the result of validation.

The ramifications of this manipulation were significant. The system time mechanism became disrupted, leading to a divergence between the observed time delta and the actual time. Initially, the deviation was minimal, but over time, it escalated, posing a considerable challenge to maintaining accurate time representation within the system.

Importantly, this adjustment of frequency did not affect every node in the system. The localized impact of the frequency change was highlighted by the time delta that was maintained by those using the standard 50Hz frequency, which was near to the expected value.

Regarding the gateway node's behavior, an interesting finding was made. The gateway node communicated with an external NTP server by broadcasting a time announce message with an exact flag at regular intervals. The altered nodes received this update and changed their time to temporarily match the ideal representation of time. The manipulated nodes soon resumed their linear time divergence because of the changed count cycle, therefore this alignment was only temporary.

One of the main factors reducing the system's overall time deviation was the gateway node's existence. The controlled nodes would have continued to deviate from the real

Figure 4.18: System Time rule set Verification by manipulating Timekeeping

time without its regular time announcements and synchronization updates, resulting in a considerable departure over an extended period of time.
This evaluation underscores the fragility of time synchronization principles within embedded systems when subjected to intentional or inadvertent manipulation of the timekeeping mechanism. Even subtle alterations in input frequencies can have profound implications for time accuracy and system operation, emphasizing the critical importance of robust timekeeping protocols in embedded system design and implementation.
The Grafana visualizations are shown in the Figure 4.18 which provide real- time insights of the offline visualization as seen in Figure 4.17. The Grafana plots 4.15 provide a comprehensive real-time dashboard for system time verification, consisting of a total of five distinct plots. These plots include three metadata visualizations and two time series plots, each serving specific purposes in the analysis of system time data. The Gauge plot displays the total number of mesh packets analyzed, excluding rebroadcast packages. It provides an overview of the volume of data processed in the system. The Pie chart breaks down the system time command packages into categories, providing a visual representation of: Total system time command packages, Successful system time validations, Pending system time validations, Failed system time validations. The Time series subplot 1 represents the time delta (Y-axis) over time (X-axis). Separate queries are made for every node in the PlejdHome mesh setup. It includes an index that shows which nodes in the mesh have announced the system time. This visualization helps in tracking the time synchronization accuracy for each node individually and collectively. Plots 4.20 show the Time delta for the node 35 under the Timekeeping testing condition. The subplot 2 in Figure 4.18 focuses on the time delta, time exactness flag status, and the result of the verification specifically for node 35 in the PlejdHome mesh setup. It provides detailed insights into how accurately node 35 maintains system time and how often it synchronizes

correctly with the time announcements.

Figure 4.19: Time Delta for node 35 under normal condition

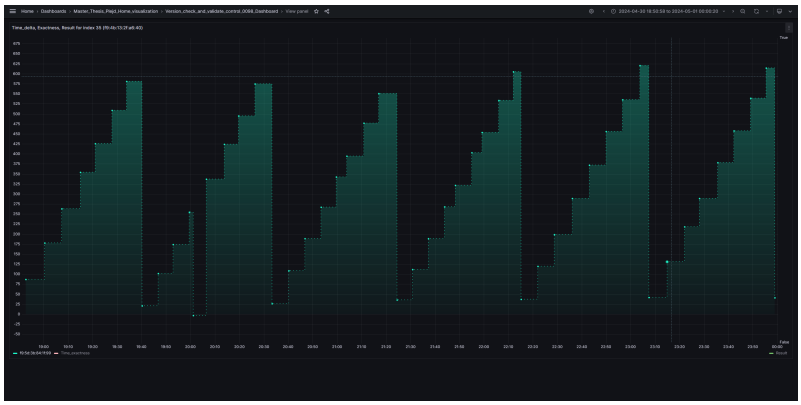


Figure 4.20: Time Delta for node 35 under Time keeping manipulation

4.5 Scene Validation Rule Set Verification

In our scene validation test, we aimed to measure the effectiveness and accuracy of the scene validation rule set as mentioned in 3.10.1 within the PlejdHome mesh network. Scenes, which are predetermined configurations applied to single or multiple devices in the network, can be activated through astronomical timings, button clicks, or the app interface. The importance of this measurement lies in ensuring that the network can reliably execute scene commands, maintain synchronization, and provide real-time feedback on the state and intensity levels of the nodes. Reliable scene validation is crucial for ensuring the overall functionality and user experience of the PlejdHome mesh network. For our test setup involving 200 devices, we configured a total of 13 scenes, including several astronomical events. These scenes were specifically set for certain lamp nodes in the mesh, allowing for both individual and group actions. Configurations were made via the app by selecting the appropriate lamp nodes and setting the desired timings for astronomical scenes. These configurations were then fetched into the Python live monitoring tool for real-time analysis. The scenes were designed to set all lamp nodes to 50% intensity, turn all lamp nodes

off, or turn all lamp nodes on. The validation of these scenes will occur within a 10-second window after the scene command was activated. Over a 48-hour period, scene verification will be conducte and results will be monitored on Grafana. This will include checking the states and intensity levels of the lamp nodes, validation results, and the specific scene numbers.

4.5.1 First Case Evaluation: 48-Hour Scene Verification Test

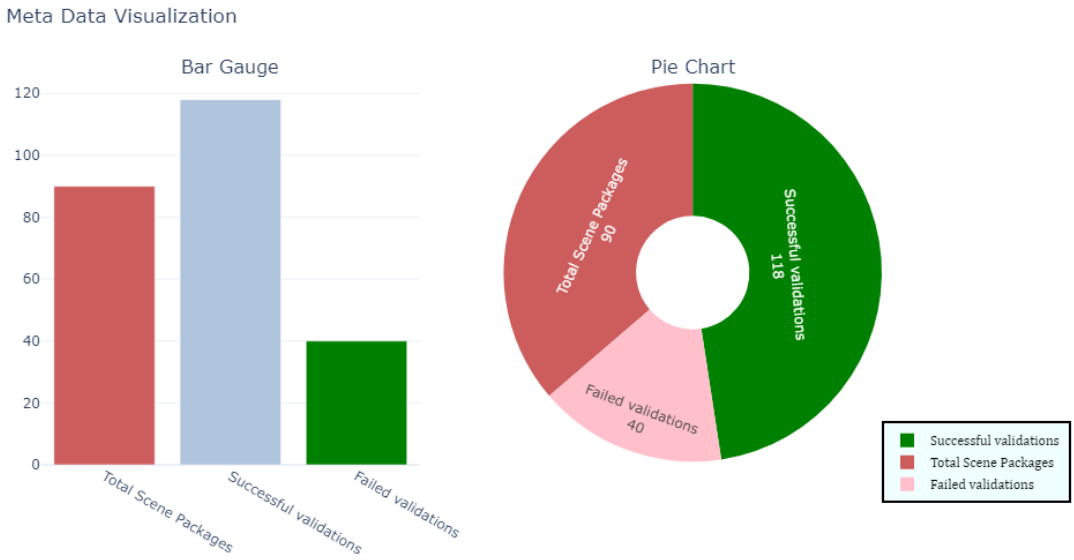


Figure 4.21: First Test Case Evaluation: Metadata offline visualization

In this test case, we conducted a comprehensive 48-hour evaluation of the scenes associated with the PlejdHome mesh test setup using the Live Analysis Tool. All the scenes, along with their respective scene numbers, associated nodes, and their state and level configurations, were fetched and stored in a dictionary within the tool. When a scene was triggered, the scene number was cross-checked against the pre-fetched data in the Live Analysis Tool, and the corresponding parameters such as the command, index, and scene number—were added to the scene lookup table. The state and level for that scene were derived from the pre-fetched data and incorporated into the lookup table. The results of this evaluation are presented in the following figures. Figure 4.21 provides a comprehensive overview of the scene validation status through two plots: a bar gauge and a pie chart. These plots display key metrics such as total scene packages, successful scene validations, and failed scene validations. Figure 4.22 depicts the results of the evaluation in five subplots. Subplot 1 illustrates the node light table intensity level for Index-40 over time. Subplot 2 depicts the intensity level associated with that particular scene number, the state triggered for that scene, and the scene number. Subplots 3 and 4 show the scene numbers and scene results over time, and Subplot 5 illustrates the scene numbers and the indexes associated with those scene numbers.
The 10-second validation window, outlined in 4.5 , posed a challenge because not all nodes sent out their current state and level within this timeframe. Consequently,

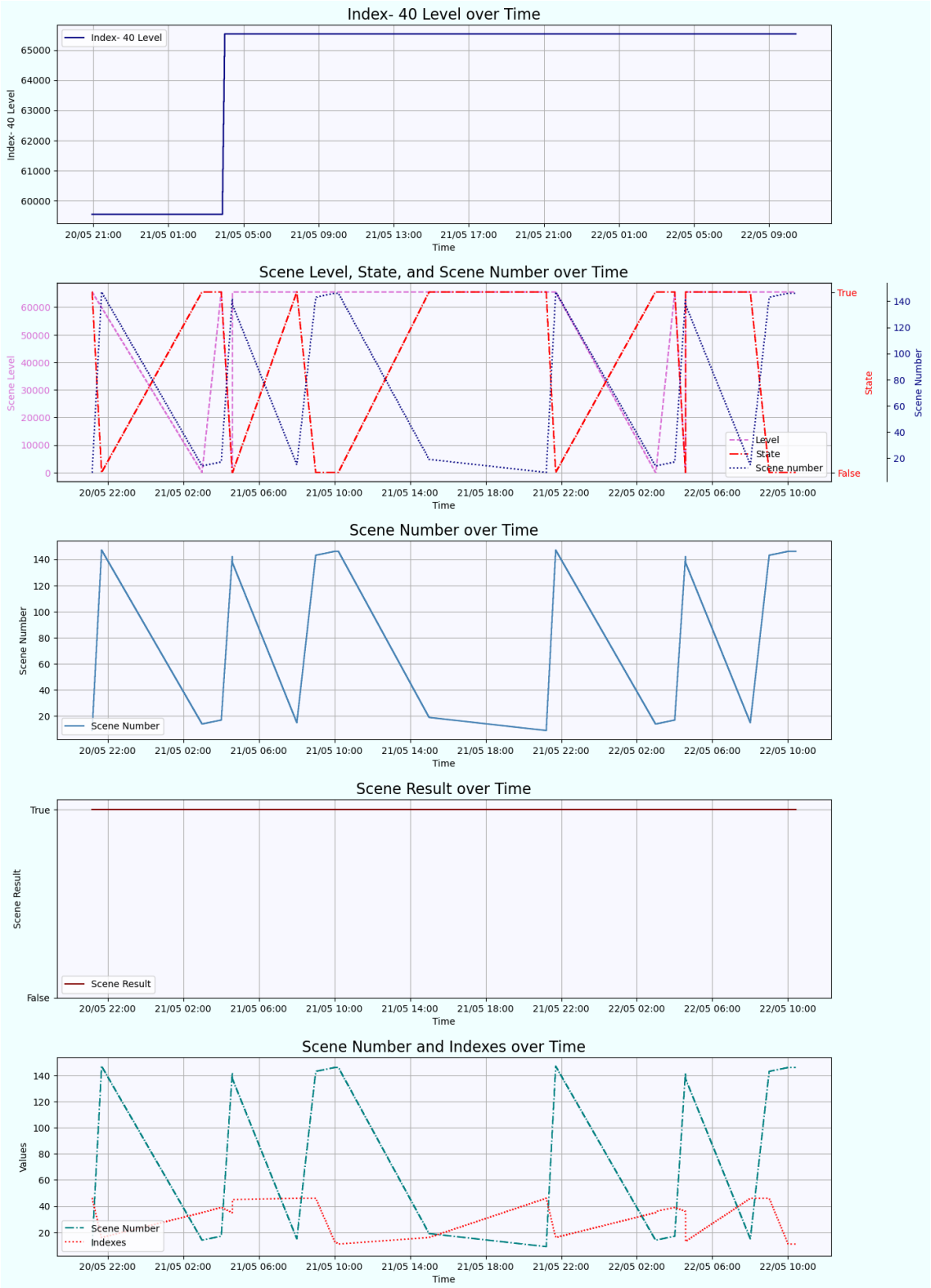


Figure 4.22: First Test Case Evaluation: 48-Hour Scene Verification Test offline visualization

scenes involving all nodes were only partially validated, focusing on a subset of the

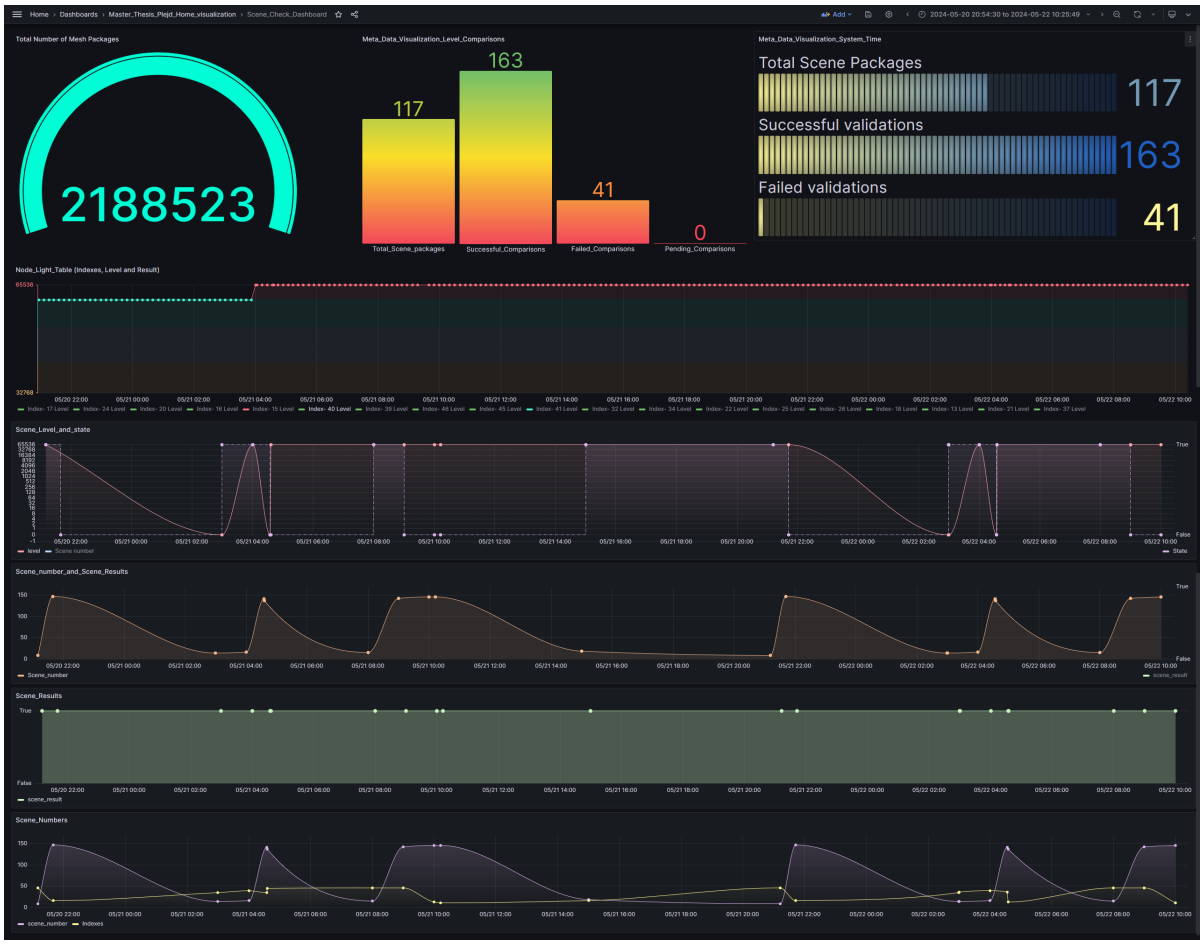


Figure 4.23: First Test Case Evaluation: 48-Hour Scene Verification Test

lamp nodes. Conversely, scenes with fewer bulb nodes and specific arrangements were fully validated. This part of the test highlighted the system's ability to handle a large number of nodes and provided insights into potential areas for improvement in synchronization and reporting mechanisms.
Every active scene in the mesh was watched throughout the course of the 48-hour period, and validations were carried out using the output announcements that were received. In order to get the output announce from the nodes connected to each scene number, a validation window of ten seconds was established. In the event that an output announce was not received within this time window, that node's scene validation was deemed unsuccessful.
The 10-second validation as outlined in 4.5 window posed a challenge because not all nodes sent out their current state and level within this timeframe. Consequently, scenes involving all nodes were only partially validated, focusing on a subset of the lamp nodes. Conversely, scenes with fewer bulb nodes and specific arrangements were fully validated. This part of the test highlighted the system's ability to handle a large number of nodes and provided insights into potential areas for improvement in synchronization and reporting mechanisms
The Grafana dashboard (Figure 4.23) for the Scene Validation Rule Set consists of eight unique plots: three metadata visualizations and five time series plots. The metadata visualizations provide important details such as the total number of scene validation packages, and the counts of successful and unsuccessful scene validations, displayed using a gauge, pie chart, and digital bar gauge for clarity and ease of comprehension. The first time series plot in Figure 4.23 depicts the Node light table intensity level for various lamp indexes over time, with separate queries made for each node in the PlejdHome mesh setup. The second time series plot shows the scene numbers, scene level, and state plotted against time, tracking when each scene is triggered. The third plot illustrates the results of the scene numbers. The fourth plot shows the scene results, indicating whether each validation was successful or failed. The fifth plot highlights the scene numbers and the specific lamp indexes affected by those scenes.
In summary, the metadata and time series plots collectively illustrate the performance and accuracy of the scene validation rule set over the 48-hour evaluation period. The results indicate that while the system is capable of managing a large number of nodes and validating scenes accurately, there are areas for improvement in synchronization and reporting mechanisms, particularly concerning the 10-second validation window.

4.5.2 Second Case Evaluation: Handling Crash or Power Outage

In this test case, we evaluated the system's handling of crashes or power outages by configuring a scene on lamp node 248 within the 200-device PlejdHome mesh test setup. The scene was designed to switch the lamp to an ON state and set its level to 100%, regardless of its previous state and level. The scene parameters were fetched into the Python Live Analysis Tool, and the scene was triggered via the mobile app. As soon as the scene was executed, we abruptly switched off the lamp node 248.

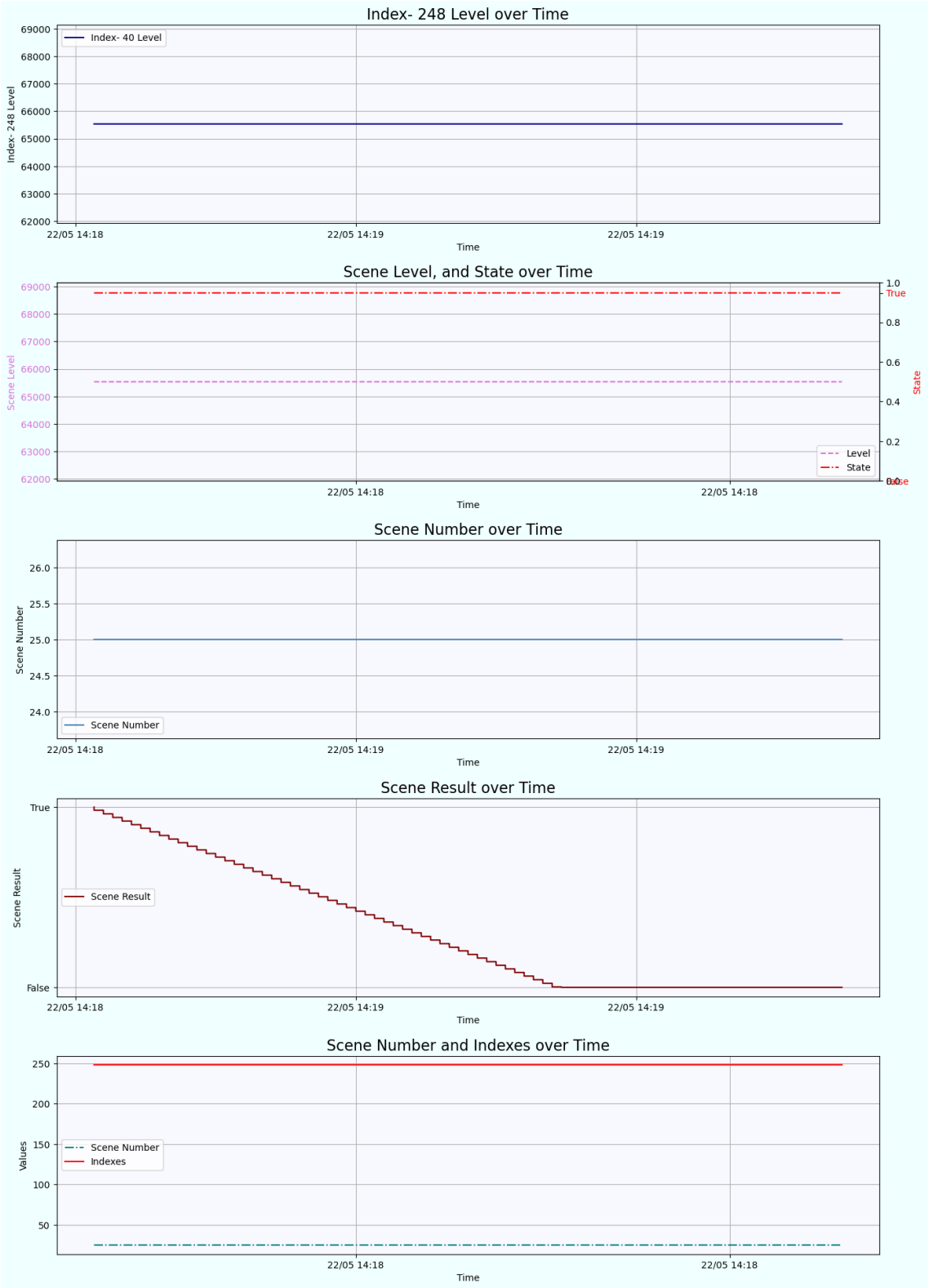


Figure 4.24: Second Test Case Evaluation: Handling Crash or Power Outage offline visualization

The results of the evaluation are depicted in Figure 4.24. This figure contains five

Figure 4.25: Second Test Case Evaluation: Handling Crash or Power Outage

subplots. Subplot 1 illustrates the node light table intensity level for Index-248 over time. Subplot 2 depicts the intensity level associated with that particular scene number, the state triggered for that scene, and the scene number, which is 25 in this case. Subplots 3 and 4 show the scene numbers and scene results over time, while Subplot 5 illustrates the scene numbers and the indexes associated with those scene numbers.

The scene parameters, which included the command, index, and scene number, were added to the scene lookup database once the scene was triggered. The scene number was then compared with those in the Live Analysis Tool. The pre-fetched data in the tool was also used to populate the scene's state and level. Next, a comparison was done for the particular index between the node light table and the scene lookup table. When the lamp node sends out an output announce, the node light table updates to reflect the lamp node's current level and status.

In this case, the light node was turned off after the scene was triggered, which stopped it from delivering the output announce. A validation failure resulted from not updating the node light table, which was seen live on Grafana. The figure 4.24 subplot 4 shows this failure; the node light table for index 248 stays unaltered, but the outcome is depicted as a failure. The plot makes it evident that the validation failed since there was no output announce, even though the scene command was executed and the state and level were modified in the scene lookup table.

The Grafana dashboard (Figure 4.25) for the Scene Validation Rule Set consists of five time series plots. The first time series plot in Figure 4.25 depicts the Node light table intensity level for various lamp indexes over time, with separate queries made for each node in the PlejdHome mesh setup. The second time series plot shows the scene numbers, scene level, and state plotted against time, tracking when each

scene is triggered. The third plot illustrates the results of the scene number which is 25 in this case. The fourth plot shows the scene results, indicating whether each validation was successful or failed. The fifth plot highlights the scene numbers and the specific lamp indexes affected by those scenes.
In summary, this evaluation demonstrated the system's response to a crash or power outage scenario. The results showed that the absence of an output announce from the lamp node after the scene was triggered led to a validation failure. This highlighted the system's dependency on real-time updates from the nodes to confirm the execution of scene commands and provided insights into potential areas for improving robustness against such disruptions.

5

Discussion

The purpose of this chapter is to provide an in-depth analysis and interpretation of the findings presented in the previous chapters. We ~~will~~ examine how the different rule sets and real-time analysis tools contribute to enhancing data integrity, network stability, and overall performance in IoT mesh networks. By categorizing our discussion, we can systematically address the implications of each rule set and tool, identify potential areas for improvement, and explore the broader impact of these technologies on IoT deployments. This chapter aims to synthesize the results, offer insights into the practical applications of our methods 3, and suggest directions for future research and development.

5.1 Version Cache Rule Set Impact

The version cache rule set is discussed, highlighting its crucial function in preserving the accuracy and dependability of data in the mesh network environment. This rule set functions as a reliable technique to detect anomalies and aberrant behaviors, serving as an early warning system for possible network problems by methodically comparing incoming mesh packet version numbers against those kept in the cache. Operators can quickly identify and resolve new issues by utilizing Grafana's real-time viewing and analysis features. This allows for proactive intervention to reduce risks and maintain network stability. Additionally, only validated and current data is handled due to the rule set's strict enforcement of data consistency, protecting against disruptions and data corruption that might compromise network performance. Its adaptability to the dynamic nature of network conditions enables seamless synchronization and coherence among nodes, fostering optimal network operation. Continuous evaluation and optimization efforts further enhance its efficacy across a spectrum of operational scenarios, reaffirming its pivotal role in bolstering data integrity, communication efficiency, and overall network resilience in IoT deployments.

5.2 Level Validation Rule Set Impact

The real-time analysis of the level comparison rule set using Grafana offers indispensable support for troubleshooting and implementing preventive actions within the IoT mesh network. By continuously monitoring the level comparison data in real-time, the team can swiftly identify discrepancies or anomalies in the state and level information received from different nodes. This capability facilitates the prompt

identification of potential issues such as communication errors, node malfunctions, or environmental disturbances. Grafana's real-time monitoring features enable the early detection of issues, allowing for proactive intervention and resolution through the setup of alerts based on predefined thresholds or patterns in the level comparison data.

Additionally, the historical data stored in Grafana serves as a valuable resource for conducting root cause analysis of past incidents. This data provides insights into the underlying causes of previous issues, enabling the implementation of targeted measures to prevent their recurrence. By analyzing the level comparison data over time, the team can identify nodes or areas of the network prone to recurring issues or performance degradation. This analysis allows them to prioritize maintenance activities, such as firmware updates, node replacements, or network optimization efforts, thereby improving overall reliability and stability.

Furthermore, the insights gained from the level comparison analysis enable the development of proactive mitigation strategies to prevent potential issues from occurring. These strategies may include implementing redundancy measures, adjusting network configurations, or enhancing node diagnostics to minimize the impact of potential disruptions.

5.3 System Time Rule Set Impact

IoT mesh network debugging and preventive action implementation are made possible with Grafana's analysis of the System Time Rule Set. The team can spot possible synchronization problems and take preventative action to preserve network stability by keeping an eye on nodes' adherence to the rule set. Grafana's visualization features make it possible to monitor time deltas between nodes in real time, which facilitates the rapid identification of deviations from the intended synchronization pattern.

Grafana may notify the team when disparities occur, enabling them to look into the matter and deal with the source of the issue as soon as possible. For instance, it can be a sign of underlying synchronization problems or network congestion if a node keeps rescheduling its announcement because it is frequently experiencing time delta violations. The team can proactively execute preventive steps by identifying patterns of synchronization abnormalities through the analysis of previous data in Grafana.

Preventive actions may include optimizing network parameters to reduce congestion, upgrading hardware or firmware to improve timekeeping accuracy, or implementing redundancy mechanisms to mitigate the impact of synchronization failures. Additionally, Grafana can facilitate trend analysis to predict potential synchronization issues before they escalate, allowing for preemptive maintenance and ensuring uninterrupted network operation.

All things considered, using Grafana for real-time System Time Rule Set analysis enables the team to efficiently diagnose synchronization problems and put preventative measures in place to improve the dependability and efficiency of IoT mesh networks.

5.4 Scene Validation Rule Set Impact

The real-time live analysis facilitated by the Scene Validation Rule Set offers invaluable assistance to the team in troubleshooting and maintaining the stability of the IoT mesh network. By continuously monitoring scene validation metrics and trends in real-time, the team gains immediate visibility into the network's performance. This real-time monitoring allows for the early detection of anomalies or deviations from expected behavior, enabling prompt intervention to address potential issues before they escalate.

One key preventive action that can be implemented is the setup of proactive alerts based on predefined thresholds or conditions. By configuring alerts to trigger when certain scene validation metrics exceed acceptable limits, the team can receive instant notifications of any deviations from normal operation. This proactive alerting mechanism enables swift corrective actions to be taken, minimizing downtime and preventing disruptions in network operations.

Furthermore, the real-time analysis of scene validation data allows the team to conduct timely investigations into the root causes of any issues that arise. By identifying the underlying factors contributing to network anomalies, the team can implement targeted corrective measures to address the root causes effectively. This data-driven approach to troubleshooting ensures that problems are resolved efficiently, leading to improved network reliability and performance.

Additionally, the continuous monitoring of scene validation metrics enables the team to proactively identify areas for optimization and enhancement within the network. By analyzing trends and patterns in real-time data, the team can identify opportunities to streamline processes, optimize resource utilization, and enhance overall network efficiency. This proactive approach to network management helps to prevent issues from occurring in the first place, leading to a more resilient and robust IoT mesh network.

5.5 Exploring Real-Time Analysis for Enhanced Testing Efficiency

Traditional testing methodologies often face challenges in complex product environments, where weak points are less defined, and stress factors are multifaceted. As product complexity increases, the need for iterative and agile testing models becomes more pronounced, making conventional pre-established testing plans less feasible.

This thesis delves into the potential of real-time analysis of incoming data from Plejd's mesh network to identify contradictions indicative of underlying problems. By continuously monitoring the mesh network, test engineers can swiftly detect discrepancies that may signal fault conditions, enabling rapid identification and reporting of issues. Estimations based on internal Plejd data suggest that such scenarios could be detected within three working days, although time frames for fault detection can vary significantly in testing environments.

To evaluate the efficacy of real-time analysis, an experiment was conducted involving

four test engineers at Plejd. Unknown faults were intentionally inserted into the Plejd home setup, and engineers utilized the Live analysis tool to identify and report the issues. The experiment aimed to assess whether the tool provided adequate information for detailed testing reports and to compare its effectiveness against conventional testing methods.
Interviews conducted post-experiment sought insights into the engineers' perceptions of the tool's efficiency compared to traditional testing approaches. Questions focused on whether existing methodologies would have detected the issues faster, estimated time frames for issue identification, and the adequacy of information provided by the tool for comprehensive testing reports. Through this investigation, the thesis sheds light on the potential of real-time analysis tools to augment traditional testing practices and improve overall testing efficiency and effectiveness.

6

Conclusion and Future Work

6.1 Conclusion

Bibliography

[1] InfluxDB, "Architecture of influxdb." `https://www.influxdata.com/time-series-platform/`.

[2] Grafana Docs, "Grafana." `https://grafana.com/docs/`.

[3] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: A survey," *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2016.

[4] J.-Y. Liao and Y.-H. Lin, "Smart lighting control using wireless sensor networks," *IEEE Sensors Journal*, vol. 15, no. 9, pp. 5552–5561, 2015.

[5] R. Majumder, "Energy management system for microgrid including phev and ev utilizing smes," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 2002–2010, 2017.

[6] P. Asghari, A.-M. Rahmani, and H. H. Javadi, "Internet of things applications: A systematic review," *Computer Networks*, vol. 148, pp. 241–261, 2019.

[7] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[8] Plejd AB, "Plejd website." `https://plejd.com/sv-se`.

[9] S. Raza, M. Shafique, and T. Chung, "A survey of iot cloud platforms," *Journal of King Saud University - Computer and Information Sciences*, 2017.

[10] X. Li and et al., "Advancements in iot mesh networks: A comprehensive survey," 2022.

[11] Y. Zhang, H. Wang, and Z. Guo, "Streaming analytics for real-time iot network monitoring," *ACM Transactions on Internet Technology*, vol. 20, no. 3, pp. 1–21, 2020.

[12] S. Jiang, X. Li, and Y. Wang, "Enhancing fault tolerance in iot networks with redundancy techniques," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 1234–1243, 2019.

[13] A. Kumar, S. Gupta, V. Singh, and M. Kaur, "Iot: A comprehensive survey on applications and its security issues," *Journal of Network and Computer Applications*, vol. 202, p. 103302, 2023.

[14] M. J. Farooq, A. Iqbal, and M. Naeem, "Internet of things: Architectures, protocols, and applications," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 4, pp. 437–461, 2021.

[15] M. Li, J. Zhang, X. Kang, J. Guo, and L. Zhang, "Communication topology analysis and optimization of industrial iot networks," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5293–5304, 2021.

[16] L. R. Oliveira, J. A. Peças Lopes, C. L. Moreira, and A. G. Madureira, "Evaluation of coap-based communication in iot environments," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 3963–3971, 2021.

[17] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[18] Norwegian University of Science and Technology (NTNU), "nrf openmesh: Bluetooth low energy based rebroadcasting mesh implementation on the nrf5x." Created in collaboration with The Norwegian University of Science and Technology (NTNU) as part of a master's thesis pre-study, 2016.

[19] Nordic Semicondictors, "nrf softdevice." `https://infocenter.nordicsemi.com/topic/struct_nrf51/struct/s110.html`.

[20] P. Levis, T. H. Clausen, O. Gnawali, J. Hui, and J. Ko, "The Trickle Algorithm." RFC 6206, Mar. 2011.

[21] J. K. Nurminen and et al., "Bluetooth low energy: An overview," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 178–183, 2011.

[22] D. Perkins and S. Park, "Bluetooth mesh networking: An overview," *IEEE Communications Magazine*, vol. 58, no. 2, pp. 104–109, 2020.

[23] T. Lepp"anen, T. D. Hamalainen, M. Pettissalo, O. L"ahetkangas, and J. Riekki, "Bluetooth mesh networking: An evaluation of the specification," in *2019 IEEE 20th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–6, IEEE, 2019.

[24] Nordic Semicondictors, "nrf microcontrollers." `https://www.nordicsemi.com/Products`.

[25] Nordic Semicondictors, "nrf52840 dongle." `https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle`.

[26] U. Raza, M. A. Salahuddin, S. A. Hassan, M. B. Khan, and S. H. Ahmed, "Performance analysis of version number based routing metrics for rpl," *IEEE Access*, vol. 7, pp. 59811–59822, 2019.

A

Appendix 1

A.1 Pseudo Code

```
import sniffer
import influxdb

# Initialize Real-Time Mesh Packet Analysis Tool
def initialize_tool():
    sniffer.connect()  # Connect to Sniffer Hardware
    influx_client = influxdb.connect()  # Establish connection to
   InfluxDB
    return influx_client

# Main function
def main():
    influx_client = initialize_tool()
    while True:
        packet = sniffer.receive_packet()  # Receive mesh packet
   from sniffer
        preprocessed_data = preprocess_packet(packet)  # Preprocess
    packet data
        parsed_data = parse_packet(preprocessed_data)  # Parse
   packet based on mesh commands
        store_data(influx_client, parsed_data)  # Store parsed data
    in structured format
        analyze_metadata(parsed_data)  # Analyze metadata (package
   counts, validation statuses)
        calculate_latency(parsed_data)  # Calculate latency
        visualize_data(parsed_data)  # Visualize data on Grafana in
    real-time

if __name__ == "__main__":
    main()
```

Listing A.1: Real-Time Mesh Packet Analysis Tool Pseudo Code

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY