University of Zurich[UZH]

Fully Fledged SDN in a LoRa Mesh

Florian Andreas Herzog
Zürich, Switzerland
Student ID: 18.916.874

Supervisor: Dr. Eryk Schiller
Date of Submission: September 1st 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

English Version

This bachelor-thesis tries to incorporate Software Defined Networking (SDN) mechanisms into a Long Range (LoRa) Internet of Things (IoT) mesh. Typically, devices used in LoRa-based wireless sensor networks (WSN) are limited in range. Therefore, in a scenario where a significant amount of nodes would be out of range of the LoRa Wide Area Network (LoRaWAN), deploying a mesh topology is a simple yet effective way to connect far away nodes using multi-hop communication. SDN, on the other hand, aims to improve network performance by analyzing the network and applying smart optimization algorithms. The hardware used in this thesis, are nodes being a Raspberry Pi, a popular choice inIoT, and the E32-868T20D LoRa Shield, a budget-friendly option to adapt LoRa technology. The software is implemented in Java, a programming language that promotes human-readability in code and benefits from decades of experience in practical software development. While LoRa networks with mesh topologies have already been subject of previous research using various devices and programming languages, the goal of this thesis is to test the effectiveness of SDN-based mechanisms in improving a LoRa mesh network and finally providing a user-friendly API as a service to other applications as a transmitter of data. Disclaimer: Neither the product, nor the analysis of SDN-based mechanisms have reached a state of success.

Deutsche Version

In dieser Bachelorarbeit wird versucht, Software Defined Networking (SDN) Mechanismen in ein Long Range (LoRa) Internet of Things (IoT) Mesh einzubauen. Normalerweise sind die in LoRa-basierten drahtlosen Sensornetzwerken (WSN) verwendeten Geräte in ihrer Reichweite begrenzt. Daher ist in einem Szenario, in dem sich eine beträchtliche Anzahl von Knoten außerhalb der Reichweite des LoRa Wide Area Network (LoRaWAN) befindet, der Einsatz einer Mesh-Topologie eine einfache, aber effektive Möglichkeit, weit entfernte Knoten mittels Multi-Hop-Kommunikation zu verbinden. SDN hingegen zielt darauf ab, die Netzwerkleistung durch die Analyse des Netzwerks und die Anwendung intelligenter Optimierungsalgorithmen zu verbessern. Bei der in dieser Arbeit verwendeten Hardware handelt es sich um einen Raspberry Pi, eine beliebte Wahl im Bereich IoT, und das E32-868T20D LoRa Shield, eine preisgünstige Option zur Anpassung der LoRa-Technologie. Die Software ist in Java implementiert, einer Programmiersprache, die die menschliche Lesbarkeit des Codes fördert und von jahrzehntelanger Erfahrung in der praktischen Softwareentwicklung profitiert. Während LoRa-Netzwerke mit Mesh-Topologien bereits Gegenstand früherer Forschungen mit verschiedenen Geräten und Programmiersprachen waren, besteht das Ziel dieser Arbeit darin, die Wirksamkeit von SDN-basierten Mechanismen zur Verbesserung eines LoRa-Mesh-Netzwerks zu testen und schließlich eine benutzerfreundliche API als Dienst für andere Anwendungen als Datenübermittler bereitzustellen. Hinweis: Weder das Produkt, noch die Analyse der SDN-basierten Mechanismen haben einen Erfolgsstatus erreicht.

Acknowledgements

At this point I would like to thank all parties who have contributed to the successful completion of this scientific work.

First of all I would like to thank Dr. Eryk Schiller, who supported the work as a supervisor with his profound knowledge and practical experience. His valuable suggestions at the regular project meetings accompanied the project status in a goal-oriented direction. I would also like to thank Dr. Schiller for providing the four Raspberry Pi devices with the E32 modules on behalf of the University of Zurich.

Furthermore I thank Prof. Dr. Stiller for the organizational facilitation and the administrative work, which made my work possible.

For correcting my work I would like to thank Dr. Chao Feng, who took care of me after Dr. Schiller left the university.

I would also like to thank the University of Zurich, which provided a conducive working environment through its institutional facilities.

Finally, I would like to thank my family and friends, as well as my current employer for their invaluable support throughout the course of this bachelor thesis.

Florian Herzog

Mettmenstetten, August 31st 2023

Contents

Terminology and Abbreviations

This chapter contains the definitions of all used terms and abbreviations. Future chapters will reference these definitions upon relevance.

WSN: Wireless Sensor Network

"WSN" stands for "Wireless Sensor Network." It refers to a network of spatially distributed autonomous sensors that are used to monitor physical or environmental conditions and collect data. These sensors are equipped with wireless communication capabilities, allowing them to communicate with each other and possibly a central base station or data collection point. [1]

WSNs are commonly used in various applications such as environmental monitoring[15], industrial automation[16], healthcare[27], agriculture[17], and more. They enable real-time data collection[26] from remote or inaccessible locations without the need for wired connections. Each sensor node typically consists of a sensing unit to gather data (such as temperature, humidity, light, etc.), processing capabilities to analyze data locally, and wireless communication components to transmit data within the network.

The data gathered by these networks can be used for analysis, decision-making, and improving various processes in different domains. However, there are challenges associated with WSNs, including energy efficiency, scalability, network management, and security due to the resource-constrained nature of individual sensor nodes and the dynamic nature of wireless communication. [20]

IoT: internet of Things

"IoT" stands for "Internet of Things." It refers to the network of interconnected physical devices, objects, or "things" that are embedded with sensors, software, and other technologies to collect and exchange data over the internet or other communication networks. These devices can range from everyday objects like household appliances, wearable devices, and vehicles to industrial equipment and infrastructure components.

The key idea behind the IoT is to enable these objects to gather and share data autonomously, often without requiring human intervention. This data can be used for various purposes, such as monitoring, control, automation, and analysis, leading to improved efficiency, convenience, and insights in various domains like smart homes, healthcare, transportation, manufacturing, agriculture, and more.[19]

IoT devices typically consist of sensors to collect data, processing units to analyze data, and communication components to transmit data to other devices or centralized systems. The connectivity and data exchange facilitated by IoT have the potential to revolutionize industries and everyday life by enabling smarter decision-making, better resource management, and enhanced user experiences. [3]

LoRa

"LoRa" stands for "Long Range." It is a wireless communication technology designed for low-power, long-range communication between devices, often used in Internet of Things (IoT) applications. LoRa technology enables devices to transmit data over considerable distances while consuming minimal power, making it suitable for applications where devices are spread out over a wide area and need to communicate with a central gateway or network.

LoRa technology operates in the unlicensed radio spectrum, and its unique modulation technique allows it to achieve long communication ranges with relatively low data rates. It's often used for applications like smart agriculture[22], smart cities[2] , industrial automation,[4] environmental monitoring,[23] and more. LoRaWAN (Long Range Wide Area Network) [**def-lora1**] is a protocol built on top of LoRa technology that defines the communication between IoT devices and gateways, facilitating efficient and scalable communication over long distances. [28]

LoRaWAN

The LoRaWAN is (as the name suggests) a Wide Area Network (WAN) using LoRa.

SDN

"SDN" stands for "Software-Defined Networking." It is an approach to network management and architecture that separates the control plane (which makes decisions about how data packets are forwarded) from the data plane (which actually forwards the packets). In SDN, the control plane is centralized and managed through software, allowing for more dynamic and programmable control over network behavior. [25]

SDN aims to simplify network management,[8] improve flexibility, and enable innovation by decoupling network control from the und erlying hardware. It allows network administrators to manage and configure the entire network infrastructure through software-based controllers, which can automate tasks, allocate resources, optimize traffic flow, and implement security policies more efficiently.

One of the key benefits of SDN is its ability to adapt to changing network requirements and traffic patterns without requiring extensive manual configuration. This flexibility makes SDN particularly useful in data centers, cloud computing environments, and large-scale networks where agility, scalability, and efficient resource utilization are important. [9]

Directed Diffusion

"Directed Diffusion" is a routing algorithm for wireless sensor networks. Nodes create interest gradients to guide data propagation. Instead of direct transmission, data spreads along these paths, enhancing energy efficiency and allowing data aggregation. This approach offers adaptability in routing, making it ideal for applications like environmental monitoring.[24] It optimizes data delivery, conserves energy, and supports dynamic communication in resource-constrained sensor networks. [29]

LQI

Link Quality Indicator (LQI) is a metric used in wireless communication systems to assess the quality of a radio link between two devices. It quantifies the reliability of received signals and estimates how well data can be transmitted over a particular link. LQI takes into account factors like signal strength, noise level, and packet error rates. Typically represented as a numerical value, higher LQI values indicate better link quality and a higher likelihood of successful data transmission. LQI is crucial in wireless networks, especially in scenarios like IoT and sensor networks, where data accuracy and reliable communication are essential for efficient operation. [6]

Network Infrastructure Components

Network infrastructure components are essential elements that form the foundation of a computer network. They include routers, switches, access points, and cables that facilitate data communication and resource sharing. Routers manage data traffic between different networks, switches connect devices within the same network, and access points enable wireless connectivity. Cabling, such as Ethernet, forms the physical connections. Firewalls and load balancers enhance security and optimize data distribution. Servers host resources like websites and files. Together, these components create a functional network, enabling efficient data transfer, communication, and access to services in various environments, from homes to enterprises. [13]

Node

In the context of computer networks and distributed systems, a node refers to an individual device, element, or point of connection within a network. It can be a computer, server, router, switch, or any other device that participates in data communication and information exchange. Nodes are interconnected to form a network topology, allowing them to share resources, exchange data, and collaborate. Each node possesses its own unique address, which aids in routing and identifying data destinations. The interactions and relationships between nodes define the structure and functionality of the network, enabling seamless communication and efficient resource utilization. [7]

Controller

A network controller is a central management entity in software-defined networking (SDN) that governs the behavior and configuration of network devices. It holds a global view of the network, allowing administrators to dynamically control and modify network policies, routing, and traffic flows through software interfaces. By decoupling the control plane from physical hardware, network controllers enable centralized management, agility, and automation. They communicate with SDN-enabled switches, routers, and other devices to enforce policies, optimize traffic, and respond to changing network conditions. Network controllers facilitate efficient resource allocation, security implementation, and network customization, empowering organizations to adapt their networks to evolving demands and efficiently manage complex network environments. [13]

PCE

Path Computation Element (PCE) is a crucial component in computer networks, particularly in the context of traffic engineering and routing. It's responsible for calculating optimal paths and routes for data traffic between network nodes. By offloading the complex path computation tasks from individual network devices, PCEs centralize and streamline the process, enhancing network efficiency, resource utilization, and responsiveness to changing conditions. [13] PCEs play a significant role in dynamic and efficient traffic management within modern networks, ensuring optimal data flow and effective utilization of network resources.[14]

Network Architecture Planes

Network architecture is organized into three distinct planes, each serving a specific purpose. Collectively, these planes work together to ensure the functionality, efficiency, and security of a network. Separating these planes allows for more organized and specialized handling of different aspects of network operations, making it easier to manage and troubleshoot complex networks. [13]

Data Plane

This plane deals with the actual movement of data packets within the network. Network devices like switches and routers operate within the data plane, forwarding packets based on predetermined rules and decisions. Data plane processing is characterized by actions such as packet filtering, switching, routing, and quality of service (QoS) classification. It's focused on efficient and accurate packet forwarding according to established protocols and configurations.

Control Plane

The control plane is responsible for network control and management. It includes processes that establish and maintain routing tables, handle network topology information, manage network resources, and make decisions about how data traffic should flow. Routing protocols like OSPF, BGP, and EIGRP operate in the control plane, determining the optimal paths for data packets through the network. Control plane functions are vital for ensuring proper network operation and facilitating communication between network devices.

Management Plane

The management plane encompasses administrative tasks related to the network. This includes activities like network monitoring, device configuration, security management, and policy enforcement. Network administrators use management plane tools and protocols to configure devices, track network performance, troubleshoot issues, and apply security measures. SNMP (Simple Network Management Protocol) is commonly used in this plane for monitoring and managing network devices.

0.1 EBNF

The Extended Backus Naur Form is a widely used syntax language that serves the purpose of formally defining a context-free grammar.[5]

1 Introduction

This bachelor thesis was written for the Communication Systems Group of the Department of Informatics at the University of Zurich over a period of six months.

The basis for it was an earlier bachelor thesis from the Department of Informatics by D. Reiss. It describes the design and implementation of an SDN based LoRa mesh using E32 modules on a Raspberry Pi. The goal was to show that SDN based mechanisms upon a custom mesh network on top of a LoRa mesh hardware stack can extend LoRa technology and provide it as extensible software.

Devices used in LoRa-based wireless sensor networks are often limited in range. Therefore, a LoRa network with mesh topology is a concept to increase the range of these devices through multi-hop communication.

Unfortunately, the final product was not able to set up the network dynamically. It was also criticized that the paper used OSPF as routing protocol. The goal of the paper at hand is now to use a more suitable routing technology.

To fix the shortcomings of the previous work, a routing protocol similar to Directed Diffusion was developed and adapted to the needs of IOT, particularly LoRa.

The system developed in this thesis was programmed in Java. For a definitive marketable product, the use of a low-level programming language such as C would be recommended in order to allow less powerful hardware than a Raspberry Pi to be equipped with the software. However, since the focus here is on the aspect of research, the choice fell on Java as a widespread high-level programming language whose successful history and experience as a programming language provides support for almost every thinkable use-case and promotes code in an often quite verbose, but sufficiently intuitive and human-readable style.

The original task included the investigation of the behavior between devices in a multi-hop manner in the real world. Unfortunately, unexpected difficulties were encountered during implementation of the production environment. However, a crude simulation showed that the system should work in theory. With this simulation, parameters of the network will be tested to find an optimal configuration.

2 Use Cases

This chapter will address two possible use-cases that might benefit from this work.

2.1 Use Case 1: Environmental Monitoring of an Underground Cave

In order to keep track of bio indicators in an underground cave such as temperature, oxygen content or humidity, it might be advantageous to use an IoT-based WSN to transport gathered data from where it was collected to the nearest exit, as equipping the whole cave with wire based infrastructure might either conflict with nature conservation regulations, or might simply not be economical.
Further, especially if the cave increases in size and depth, it is likely for a majority of the Nodes (IoT devices) to be unable to establish a connection to one central base station. Even though LoRa does provide highly reliable long-range communication, a complex underground cave could easily bring the conventional LoRa architecture to its limits.

However, when extending the conventional LoRa approach with a mesh network, far-flung parts of the cave could use multi-hop communication, letting intermediate nodes forward their data up to the central LoRa station. The nodes could then be equipped with sensors to generate the data to be collected. This data could be used as a warning system to predict when the cave's ecosystem might be endangered.

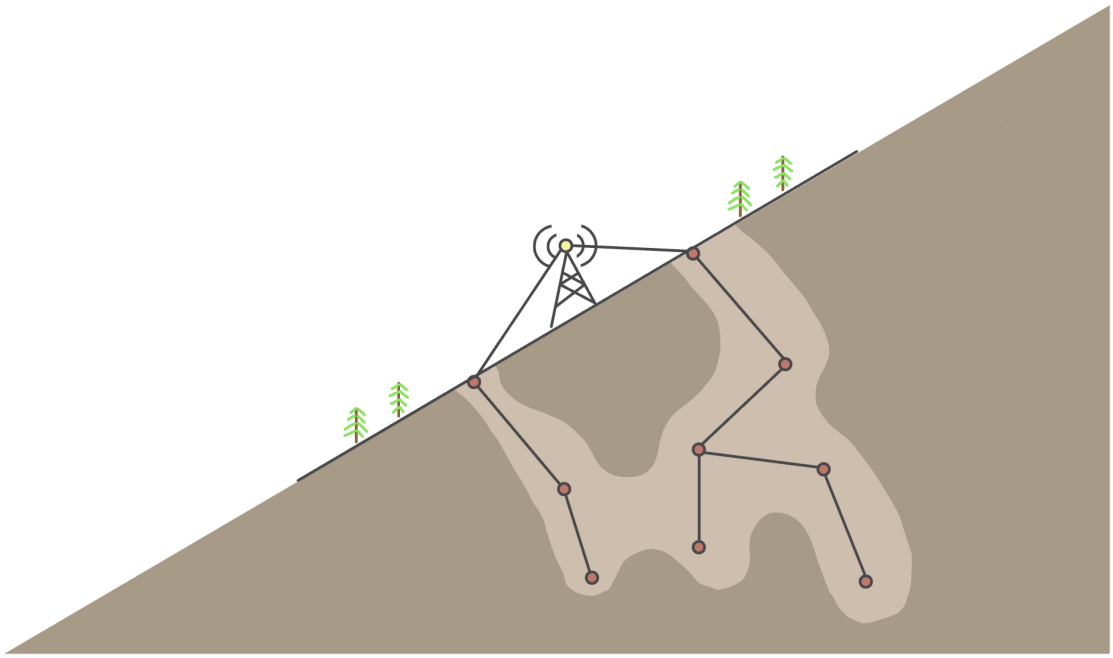

Figure 1: LoRa Mesh suitable for monitoring an underground cave (source: self-made illustration)

2.1.1 Target User Identification

Use case 1 would be targeted to geographers or biologists checking and maintaining environmental regulations.

2.1.2 Goal Specification

The deployed mesh should be able to feed data to a remote database and analysis tool.

2.1.3 Success Criteria

The following Criteria must be met for an implementation to be considered successful:

- All nodes must be able to successfully transmit data to a targeted client software in a multi-hop manner.

2.1.4 Functional Requirements

The following functional requirements must be met:

1. A node must be able to create a new mesh as the first mesh controller, if the LoRa base station is within reach and no existing mesh is within reach.
2. If an existing mesh is within reach, then the node needs to be able to join the mesh. The node might become a controller if the LoRa base station is within reach.
3. The mesh should be able to handle multiple-controller scenarios.
4. A freshly booted node needs a means to discover the mesh without knowing what channel the mesh is operating on.
5. After a freshly booted node discovered the mesh and found the corresponding channel, the node needs to have a procedure at hand to announce its presence to the mesh and request to join it.
6. When a node outside of the mesh wants to join the mesh, the node within the mesh who witnessed the joining node's presence needs to have a procedure at hand to forward that join request to the PCE.
7. When the PCE receives a join request of a node wanting to join the mesh, it needs a means to validate this request. A join request must be approved at most once per node.
8. Once the PCE approved a join request, it needs to be able to find an unused, unique mesh address to assign to the joining node.
9. A joining node whose join request was accepted needs to learn about its assigned mesh address eventually.
10. When a new node joined the mesh, according routing updates of established nodes need to be performed as part of the joining process, so that the joined node can use the mesh's capability of sending data in a multi-hop manner.

2.1.5 Non-Functional Requirements

The following non-functional requirements must be met:

1. A node must be equipped with a LoRa transmitter module that can be controlled by the software.
2. A node must be equipped with a battery that can supply it with power for days to weeks, before the battery needs to be replaced.
3. The mesh must provide an interface to feed the collected sensor data to. It must be feasible for a developer with limited experience with the LoRa mesh to implement an adapter software that extracts the necessary data from whichever service provided by the sensor, and feeds it to the mesh.
4. The data sink of the mesh must provide an interface that allows streaming collected data into an application to further process the data, or persist it (e.g. a database).
5. Each node must either be within reach of the LoRa base station directly, or within reach of a node that has already established a single- or multi-hop connection to the base station.

2.2 Use Case 2: Monitoring of Livestock Health on an Alpine Pasture

In the Swiss Alps, a farmer wants to monitor the health of his cows in a very poorly developed area. On the Alps, the animals can spread out over a wide area. For this purpose alone, setting up a central LoRa transmitting station would simply not be profitable. Instead, the range of the nearest station could be extended using the LoRa mesh, all the way to the cow pasture being used. Each cow would be equipped with a sensor that could measure hormones, body temperature, or the like. These sensors would then be able to send the data through the mesh to the central data collection point. The mesh would need to have multiple relay nodes between the transmitting station and the data collecting nodes in the case of a remote cow pasture, in case the path to the transmitter directly would be too far.

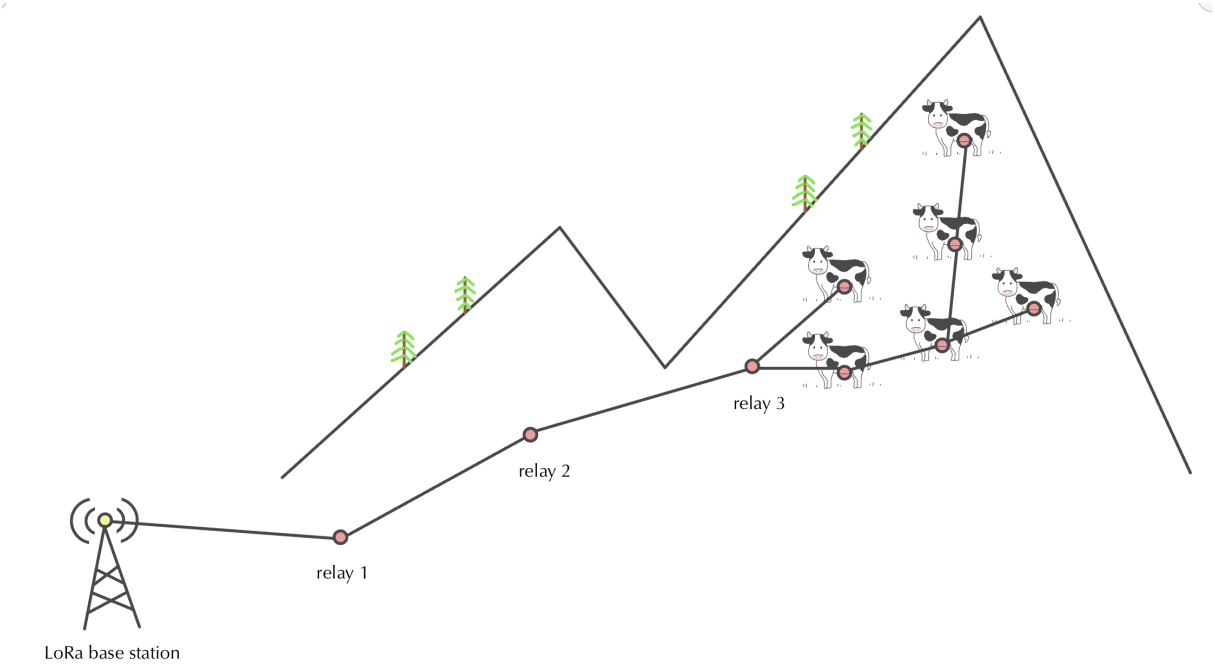

Figure 2: LoRa Mesh suitable for monitoring livestock on an alpine pasture (source: self-made illustration)

Another advantage of this would be that due to the increased mobility of the LoRa mesh nodes, moving the cow pasture would not be a problem.

2.2.1 Target User Identification

Use case 1 would be targeted to alpine farmers who are interested in revolutionizing their business with modern technology.

2.2.2 Goal Specification

The goal specification is equivalent to the one of Use Case 1.

2.2.3 Success Criteria

The success criteria are equivalent to the ones of Use Case 1.

2.2.4 Functional Requirements

The functional requirements extend to the ones of Use Case 1 with the following:

11. Since the livestock will be moving, the mesh needs to be able to dynamically adapt topology changes and discontinuous link reliabilities.

2.2.5 Non-Functional Requirements

The non-functional requirements extend to the ones of Use Case 1 with the following:

6. A node needs to come in a compact form that can be put on a cow in an animal-friendly manner.

3 Approach

Right at the beginning, the underlying work [18] was analyzed and all necessary procedures were extracted in order to get a minimalistic version of the mesh running. This resulted in a model with the most important processes that must run in a node, as well as the individual stages through which a single node must pass so that the network can fulfill its purpose. At the same time, the structure of a generic message was created, so that all types of messages needed in the future can correspond to this structure.

With this model as a basis, a simulation was written. The idea behind this was that certain nodes were difficult to access in the actual environment and could not accept a static IP address due to DHCP. This was not required, but offered a way of circumventing the tedious procedure of getting access to the individual nodes with every code change in order to test.

In the process of creating the simulation, an interface was developed, which must be presented by the individual network components. With the simulation ready, the implementation of the network components could be started according to the derived interface. While implementing these components, a lot of time and effort was spent particularly on informative logging. This would later be a critical tool when debugging the whole system. The main part of this part of the work was to debug the network components so that all components were able to run in parallel. The biggest issue there was thread-safe task management with components interacting with each other, which often raised circular dependencies. The solution was found in a central executor service and a consistent use of it, so that all procedures in a node are queued and scheduled asynchronously and individual components are strictly isolated from each other.

With the functioning of any individual simulated node, it was then possible to debug the coordination between the nodes. This includes message forwarding, routing calculations, correct accounting of sent and received messages, and again informative logging.

Now that the network components worked as hoped with the simulation as a targeted interface, the next step was to replace the simulation interface with the actual production interface. Unfortunately, this adaptation required more effort than expected. Consequently, due to time constraints, it was necessary to return to the simulation in order to be able to generate conclusions and insights despite the lack of a real-world implementation.

Finally, to generate data using the simulation, an additional statistics plugin had to be implemented that allows the extraction of relevant data from a test run.

4 Design

4.1 Requirements and Assumptions

The Usage of LoRa in an SDN-based mesh WSN projects a couple of requirements onto the design of this system. These, as well as assumptions taken in cases of unclear requirements, are listed in this section to provide a foundation for the design decisions taken in this chapter. Trivial requirements inherited by the general nature of a WSN are not listed.

4.1.1 Primary Metric: Reliability

The quality of the network with respect to available routing alternatives will be measured by the reliability of all paths. For path computation, this means that the algorithmic distance between two nodes u and v is derived from the inverse probability of a message being successfully transmitted from u to v.
Note that this reliability cannot be expected to be symmetric.

4.1.2 Requirement: Autonomous Deployment and Optimization

In order to be practically usable, the mesh should provide a shallow learning curve to a potential user / developer, requiring from them as little proprietary knowledge as possible.
Generally, this means that anyone using the mesh as a means to transport data should only have to worry about the input and output of said data on different ends of the network. In particular, this means that both the initial deployment and operational adaptions of the mesh should happen in complete autonomy. The only human interaction with respect to the network setup should be to plug in the devices and launch the software.

4.1.3 Requirement: Network Data Analysis Function

It must be possible for the mesh to autonomously measure link reliabilities, i.e. the probability for a message to be successfully sent from one node to another. Correspondingly, the Path Communication Element (PCE) of the network must receive this data and be able to use it for optimization.

4.1.4 Requirement: Information Density of Messages

LoRa imposes strict limitations on the temporal use of a channel. Consequently, messages should be as short and scarce as possible, implying a high information density on sent messages.

4.1.5 Assumption: Limited Amount of Nodes

Since scalability will be limited by LoRa's restrictions either way, it should be fair to assume that limiting the mesh size in terms of the amount of nodes, does not pose any significant disadvantages to the product. If scalability is a relevant requirement for a potential use-case, one may be better advised to stick to the standard usage of LoRa as a technology.

Applying this assumption and putting a limit on the amount of nodes, we can consequently restrict mesh addresses to a fixed number of bits, which helps us defining a compact and well-structured low-level data format to reference these addresses along with their corresponding nodes.

4.1.6 Assumption: No End-to-End Communication Between Regular Nodes

As the generic use case for the mesh is to collect data from its nodes in a self-sustaining manner, the following list of communication modes, if applied on each individual node, should suffice for the mesh to meets all requirements:

(1) (Multi-Hop Unicast) Node to Data Sink (feeding application data)

(2) (Multi-Hop Unicast) Node to PCE (feeding network data)

(3) (Multi-Hop Unicast) PCE to Node (feeding routing instructions)

(4) (Single-Hop Broadcast) Node to all Neighbors (announcing presence and allowing neighbors to measure LQI)

Analyzing these possibilities, we will notice (1), (2), and (3) all being Multi-Hop Unicast types between potentially different endpoints. We can use this to define the network paths [N-DataSink] for (1), [N-PCE] for (2), and [PCE-N] for (3) for each node N.
Regarding (4) as the exception, we will notice that it makes no sense to use the term "path" in the context of a WSN for a Single-Hop Broadcast Message, for the node will simply emit the message, and every other node who receives it can be considered a neighbor (hence we have a Single-Hop Broadcast). One can argue that because wireless data transmission is highly unreliable, not all neighbors will receive that message , putting the term broadcast in question, but we will explain how to use this issue to our advantage in 4.4. For now, let's just stick with the understanding that we can treat (4) as a "special case" that doesn't need to be considered for routing.
Combining this finding with the one that (1), (2), and (3) each produce at most one path per node, we can establish that if we restrict ourselves to only sending messages matching the list above, the number of paths in the network that need to be considered for routing equals 3 times the number of nodes in the network:

$$p = 3 \times n$$

where p equals the number of paths to consider, and n equals the number of nodes. This is different from regular networks, as for a network featuring node-to-node communication, the number of paths would be at least:

$$p \leq n \times (n - 1)$$

This comes with the following advantages:

- For any message, the path it is supposed to travel can be identified just by specifying (a) the communication mode (see list above), and (b) the associated node's mesh address.
- We can use the fact that the number of paths scale linearly with respect to the number of nodes, to reduce any message's header length.
- Routing can be simplified to a small set of Destination-Oriented Directed Acyclic Graphs (DODAGs).
- From a node's perspective, routing can be implemented just by keeping track of which paths the node is part of, i.e. for which paths the node should forward messages. We can use 4.1.5 to infer that this list of paths should be contained within an order of magnitude that can be handled by node hardware. If the PCE also keeps track of each node's current routing data in order to compute the state difference between routing updates, the PCE can also compress the data necessary for a routing update of a particular node, to fit into only a few messages.

4.2 Network Architecture

As suggested by [12], it is a widely used practice to divide networks into the following architectural planes:

- Data Plane:
- Control Plane
- Management Plane

This is said to greatly improve the modularity of a network, along with other benefits.
We will adapt this pattern and customize it a little to adapt it to our needs and advantages.

4.2.1 Data Plane

The Data Plane will have the following responsibilities:

1. Nodes sending collected application data to the Data Sink.
2. Nodes sending collected network data to the PCE.

Both responsibilities 1 and 2 can be resolved as a simple implementation of Multi-Hop Communication.

4.2.2 Control Plane

The Control Plane will have the following responsibilities:

1. Network Discovery: Detecting new nodes that want to join the mesh.
2. Link Quality Measurement: Collecting network data, i.e. estimating the reliability of Single-Hop transmissions between neighboring nodes.
3. Routing Control: PCE sending calculated routing updates to the according nodes, i.e. configuring routing policies.

Responsibility 1 will be discussed in section 4.6. Responsibility 2 will be discussed in section 4.4 Responsibility 3 can be resolved as a simple implementation of Multi-Hop Communication.

4.2.3 Management Plane

The Management Plane will have the following responsibility:

1. Node Configuration: Configuring Node addresses and node states.

This will be discussed partially in section 4.6, meaning the configuration of a joining node. The only other node status change happens autonomously when a node fails a periodic status check, which causes the node to enter the error-state, reboot and re-join the mesh.

4.2.4 Path Computation Element (PCE) and Data Sink

For the scope of the design chapter, the Path Computation Element (PCE) and the Data Sink (target interface for collecting application data) are assumed to be equivalent. This may be subject to change if demanded by stronger requirements. Further, the PCE / Data Sink are will be located on a web-service **outside** of the mesh.

As with the current implementation, the PCE / Data Sink is a Web API.

4.3 Multi-Hop Communication / Routing

As mentioned in 4.1.6, the only relevant paths are between individual nodes and the PCE / Data Sink. Therefore, assuming that

4.4 Network Data Analysis Function

In order to meet requirement for a Network Data Analysis Function (NDAF), nodes need a means to measure adjacent links, i.e. the distance to their neighbors.

In order to ensure a continuous and informative NDAF, each node will periodically send Hello-Messages to all of their neighbors for the purpose of measuring link reliabilities.

Since a node by itself has no way of noticing which neighbors received a sent message, it is easier to measure the link reliability on the receiving end. Therefore, we will use a Link Quality Identifier (LQI) to keep track of the ratio of received Hello-Messages.

Each Hello-Message will be part of its own series, meaning the series of Hello-Messages emitted by a particular node (see 4.7 for further specification). The receiving node will be able to identify this series, as well as the serial counter of the received message. The receiving node can then use this data to keep track of how many Hello-Messages it recently received from any other node, enabling it to calculate the corresponding LQI.

The map of calculated LQIs per neighbor will periodically be sent to the PCE, using Multi-Hop Communication. The data obtained in this way can then be used by the PCE to find possibilities of optimization.

4.5 Nodes Finding the Mesh's LoRa Channel

In order to allow new nodes to detect the mesh, each node within the mesh will periodically send the code of the mesh's LoRa channel on a special rendezvous-channel. A new node can hence listen on this rendezvous-channel in order to detect the mesh.

4.6 Nodes Joining the Mesh

Once a new node has found the correct channel, said node now needs to issue a request to join the mesh. This request must be verified by the PCE, and the node must receive a unique address within the mesh. Since the new node is unlikely to be able to communicate with the PCE directly, this communication must be taken care of by any neighbor of the node as a mediator.

The designed process is the following (visualized in figure):

1. New node announces presence, issuing a join-request with a volley of messages that contain the node's globally unique hardware-id as data.
2. One or more of the new node's neighbors receive enough of these join-messages to confidently consider themselves a suitable mediator and potential initial point of entry for the node. (The PCE might re-connect the new node as soon as more network data is available, but for now the new node just needs to find an initial connection)
3. The nodes which consider themselves a suitable mediator will inform the PCE about the waiting new node, sending a message that also includes the new node's globally unique hardware-id as data.
4. The PCE receives one or more of these mediator-join-requests, decides which one to approve, and assigns a mesh address to the new node which is unique within the mesh. The hardware-id attached to a join-request helps the PCE identify duplicates, such that the new node cannot be assigned multiple addresses.
5. The PCE sends a message back to the mediator. This message contains the assigned mesh address as data, as well as the new node's hardware-id (to avoid confusion if multiple join requests are active).
6. Each node on this message's path (including the mediator and every intermediate node) takes notice of the new node joining the mesh, and notes the assigned address in order to accept this node as an extension of this message's path. This will later enable the node to send messages over the same path using its own address.
7. The mediator node finally answers the new node's request and informs it about its new mesh address.
8. The new node accepts its new status and schedules the standard procedures:
    - Announcing the mesh channel on the rendezvous channel.
    - Announcing own presence to neighbors and allowing them to measure LQI.
    - Sending NDAF-insights to the PCE.
    - Sending application data to the Data Sink.
    - Functionality testing.

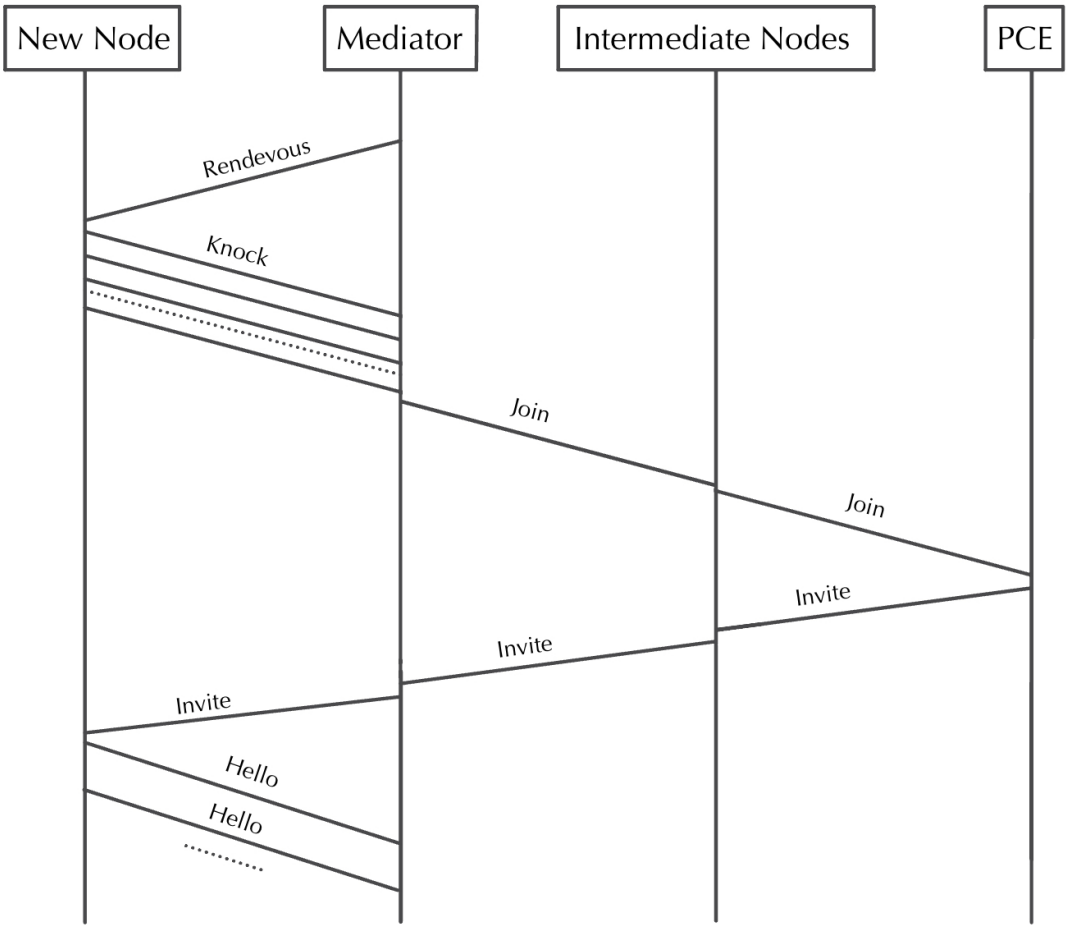

Figure 3: UML class diagram visualizing the LQI register of a node (source: self-made illustration)

4.7 Message Structure

From our previous insights we can gather that every message should contain the following data:

- A means of identification of the (directed) path which the message is traveling on.
- A sequence counter to allow the detection of missed messages.
- A message-type code to indicate how to interpret the message:
  - Type 'Rendezvous': Messages to announce the mesh's LoRa channel.
  - Type 'Data': Messages to send collected application data through the mesh for the end user.
  - Type 'Knock': Messages to indicate that a node outside the mesh has detected the mesh and wants to join.
  - Type 'Join': Messages informing the PCE about a node that would like to join the mesh.
  - Type 'Invite': Messages from the PCE to approve a join request and configure the joining node
  - Type 'Hello': Messages of nodes re-announcing their presence to neighbors and allowing them to measure the according link reliability (LQI)
  - Type 'NDAF': Messages to update the PCE about measured link reliabilities (LQI)
  - Type 'Routing': Messages to update a node's routing policies

Since we want to keep messages as short as possible, we will have to allocate a fixed amount of bits for each of those header parts. Here we will specify these exact numbers for the purpose of having an example, but keep in mind that these numbers are effectually parametrizable and can depend on the implementation.
That being said, let's define the message-type code to use 3 bits, as we derived the necessity of 7 different message types, implying that 8 possibilities are enough.
Further, let's define the Sequence Counter to use 5 bits, leaving 32 possibilities. With the sequence counter being a continuously incrementing integer overflowing on 32, message loss would only remain undetected after 32 subsequently lost messages. If we further bind the sequence counter to a semantic correspondence between sender and receiver such that the receiver can keep track of the sequence counters received by any sender, undetected message loss becomes highly unlikely in a decently connected network.
For the message's path, let's use an 8-bit integer with the following sub-structure:

- One bit indicating whether the message is sent by a node and directed to the PCE or Data Sink, or whether the message is sent by the PCE and directed to the node.
- seven bits to identify the associated node (sender or receiver, depending on the direction-bit)

With that, we define a message structure which will be used by every message sent through the mesh (pseudo-EBNF):

```
<message> <= <type-code> <sequence-counter> <path-id> <data>
<type-code> <= <3 bit>
<sequence-counter> <= <5 bit>
<path-id> <= <direction-bit> <node-address>
<direction-bit> <= <1 bit>
<node-address> <= <7 bit>
<data> <= {*any binary data*}
```

It is worth noting that the message structure as proposed here, will limit the amount of nodes in the mesh to:

$$126 = 2^7 - 2$$

We deduct two address possibilities: 0 for none (for joining nodes), and -1 for invalid (for exception cases).

4.8 Message Caching

In order to avoid messages constantly flooding the network in circles, it is important for a node to be aware of which messages it has already seen. The unified message header defined in 4.7 allows us to identify a message as duplicate if it was already seen. Especially the Sequence Counter helps with this, as the number increments between messages.

4.9 Message Tracing

Since the Sequence Counter defined in 4.7 allows us to detect lost messages and messages are cached by intermediate nodes on the message's path, we can also implement a mechanism to restore lost messages: If a node takes noticve of a skipped sequence counter, it can broadcast this notion by appending that sequence counter and the associated correspondence data (node address and path direction) to future Hello-Messages, informing its neighbors about the lost message. If any neighbor finds a match in its cache, it can then re-send it.

5 Implementation

5.1 Complexity Management

Despite often being independent from any visible results, complexity management is a crucial factor for the success of most software projects. Commonly used design Patterns such as Dependency Injection, or the Observer Pattern can contribute greatly to simplicity, maintainability, and extensibility of a software system.

5.2 Core Modules

Figure 4 shows a UML diagram visualizing the abstracted implementation of the core modules.

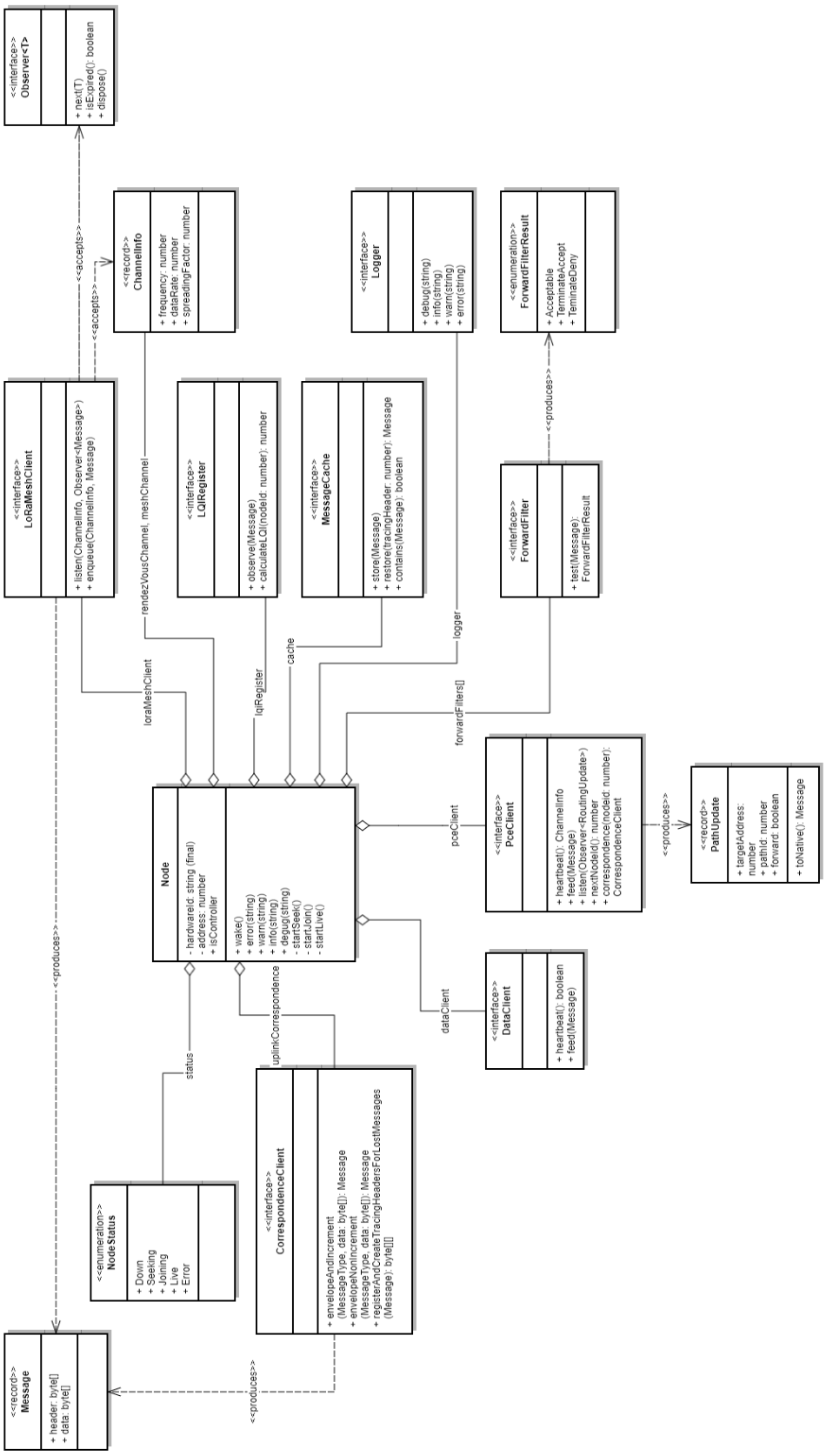

Figure 4: UML class diagram of a node and the core modules (source: self-made illustration)

5.2.1 LQI Register

The figure shows how the connection quality between individual nodes is determined. In the example, node A sends a Hello message, namely the fourteenth Hello message from the sequence of Hello messages from A. The neighboring nodes B and C receive the message, D does not. Nodes B and C both include the sequence number of the message in their register. B, which started counting at sequence number eleven but did not receive sequence number twelve, has thus received three out of four Hello messages from A. From this we can conclude that messages from A have about a fifty percent chance of being received by B. C already started counting at sequence nine but missed numbers ten, eleven, and thirteen. With three out of six Hello messages received, C's reception strength with respect to A is approximately fifty percent. D, which did not receive Hello number fourteen, counts accordingly only messages from ten to thirteen. If twelve was not received, the presumed reception probability is seventy-five percent.

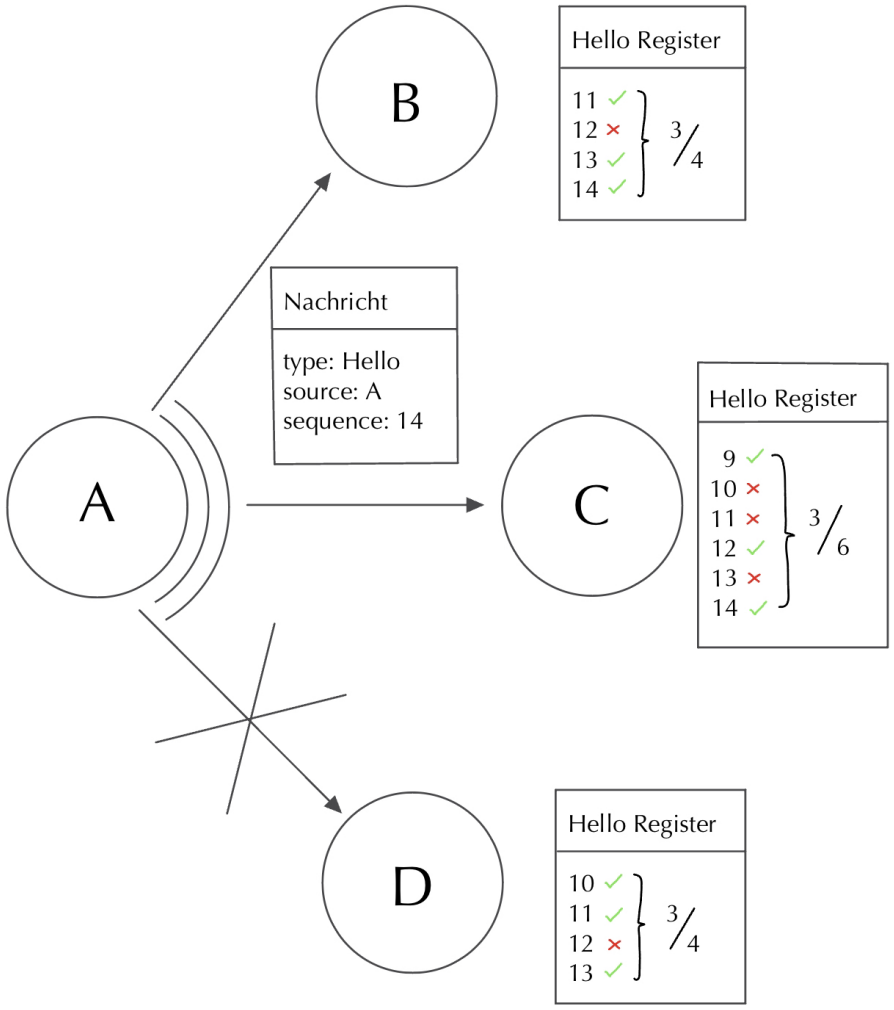

Figure 5: Visualization of the concept of the LQI register of a node (source: self-made illustration)

5.3 Production

As mentioned before, the production interface could not be implemented in time. The reason for this was ultimately the time required for the debugging process. Nevertheless, the peculiarities of the components important for the production interface are discussed in this section.

It is worth noting that the production interface was eventually almost completed, but because of the time running out, even with a running system there would have been too little time to run sophisticated tests in reality.

5.3.1 PCE as Web API

The PCE of the production interface differs only slightly from the simulated version in terms of the calculations, i.e. the actual functioning of the PCE as a network component. The significant effort of the new implementation of the PCE came with the fact that a complete Web API had to be set up and made accessible via the Internet. Likewise, an adaptation had to be implemented on the node side in order to execute the communication from the controller to the PCE via this API with HTML requests.

5.3.2 Raspberry Pi System Adaption

To make the system compatible with a Linux-based Raspberry Pi it is not only necessary to adapt system calls and filesystem I/O, but it must also be taken into account that not all frameworks, libraries and Java versions are compatible with the Raspberry Pi. The Java Runtime offered for the Raspberry Pi at this point in time only supports class file versions up to 55.0, which corresponds to Java 11 (currently Java 20 is the latest version). Accordingly, a lot of time was lost to downgrading large parts of the existing software to previous Java versions, and replacing frameworks and libraries with custom solutions.

5.3.3 LoRa Module Using Effecitve Hardware

The LoRa module used was an E32-868T20D. The manufacturer Ebyte offers a dedicated service for this module to control it. Via command line a command can be executed to either transmit on a specific channel, or to listen on a specific channel. The idea was to execute this command within the Java program and to redirect the input or output stream to a buffer that can be controlled in the program.

However, this is also associated with a certain complexity in Java, which ultimately exceeded the planned timeframe during the debugging process.

6 Evaluation

In this chapter we will run tests on the simulation, generating data output from different setups, and see how changing different parameters changes the results.

Specifically for this purpose, a special execution service was implemented which can work on virtual time to control (mainly accelerate) the time of the simulated scenario. Also, this new execution service can be programmed to halt the simulation after a fixed time period, making sure that the results of individual test runs are comparable with respect to fairness (time as a resource).

6.1 Analyzing impacts on the Autonomous Setup

In order to test the quality of the autonomous setup, we will run the tests with the following strategy:

- Each test configuration will be repeated five times. The comparative score of the results will be derived from the average time it took for a node to join the mesh.
- Test configurations will differ in the following properties:
  - Node Topology (there will be two topologies: uc1 resembling Use Case 1, and uc2 resembling Use Case 2 (without moving nodes however, since this is not an automatable feature of the simulator at this moment)
  - Parameter Join Timeout
  - Parameter Join Volley
  - Parameter Message Tracing

The parameters modified in the test configurations have the following meaning:

- **Join Timeout** defines how long a joining node waits for a join request to be answered, before considering the request failed and trying again.
- **Join Volley** defines how many messages are sent with a single join request. A larger volley increases the chance that enough messages are received by the mesh, and also allow the receiving node to better estimate the initial LQI to the joining node.
- **Message Tracing** can either be used, or disabled. It is a mechanism to try and recover lost messages, that comes with a certain overhead.

Test configurations will use the following code (EBNF):

```
<config> <= <topology>; <join_timeout>; <join_volley>; <message_tracing>;
<topology> <= uc1 | uc2
<join_timeout> <= long timeout | medium timeout | short timeout
<join_volley> <= large volley | medium volley | small volley
<message_tracing> <= tracing on | tracing off
```

Since the effect of the concrete values of these parameters are relative to the simulator and can not be set into relation with any standard, we will abstract the effective values of join-timeout and join-volley with the terms 'long' (12 seconds), 'medium' (6 seconds), and 'short' (2 seconds), or 'large' (50 messages), 'medium'(20 messages), and 'small'(10 messages) respectively.

6.1.1 Test 1: uc1; short timeout; small volley; tracing off

The first configuration to be tested will use topology uc1 (figure 6), a short join-timeout, a small volley of join messages and no message-tracing.

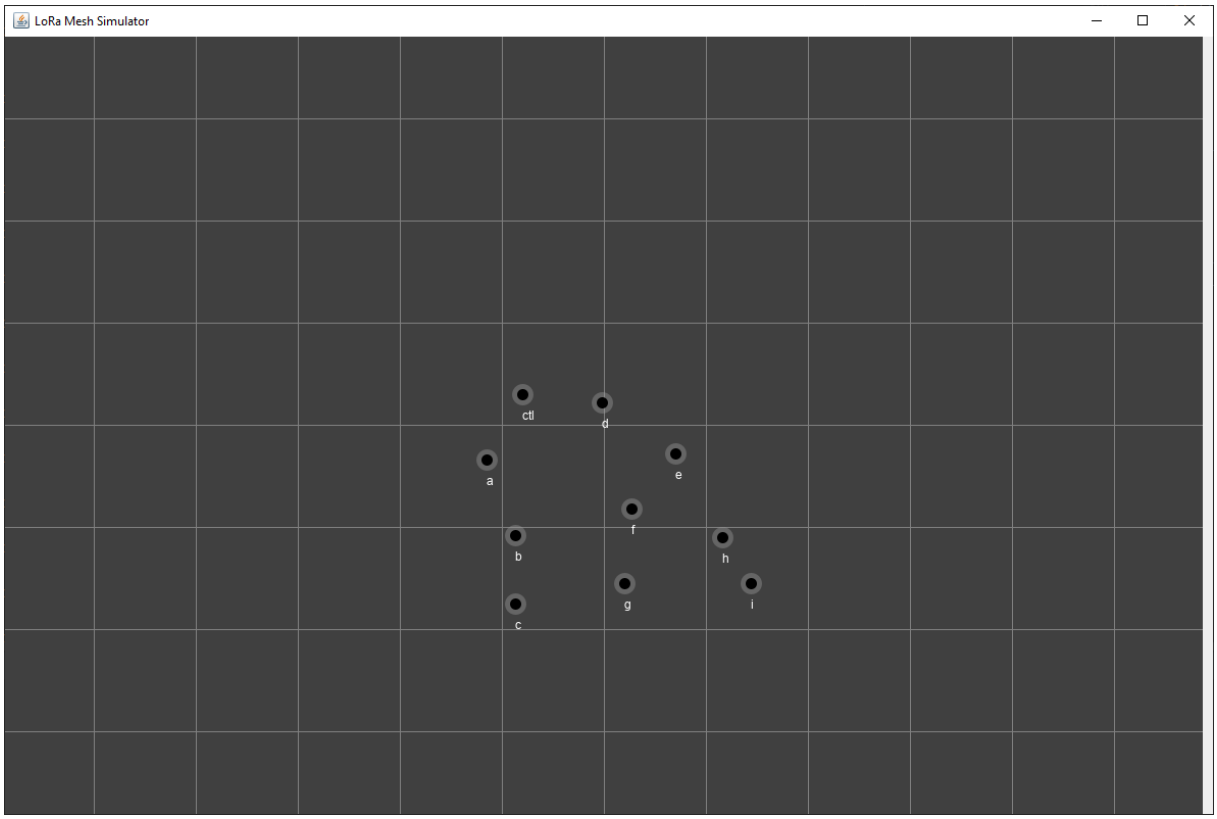

Figure 6: Node topology of uc1 (source: simulator)

The results are listed in table 6.1.1. The columns run <x:1..5> represent the results for rest run x. The rows represent a node. The left-most column, as well as the bottom row contain the arithmetic average of the corresponding row or column respectively. A cell contains the time unit at which the associated node successfully joined the mesh during the associated test run, or a dash (-) if the node was unable to join the mesh within the time given.

Looking at this data, we can observe that only a few nodes were able to join the mesh.
Also, test run 1 stands out as an outlier, having relatively long setup times. However, since it also follows the pattern of most nodes being unable to join, we will consider these numbers bad luck for the time being, and focus on why nodes are unable to join.

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	54539	2257	2227	2363	2368	12750.80
b	-	9779	3913	4952	10445	7272.25
ctl	-	-	-	-	-	-
d	54789	2498	2473	2685	2611	13011.20
e	55423	4862	3200	4586	3059	14226.00
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
i	-	-	-	-	-	-
average	41188.00	3879.40	2362.80	2917.40	3696.80	9452.25

Table 1: results data of test 1

6.1.2 Test 2: uc2; short timeout; small volley; tracing off

The second configuration to be tested will use topology uc2 (figure 7), and otherwise use the same parameters as the first test: A short join-timeout, a small volley of join messages and no message-tracing.

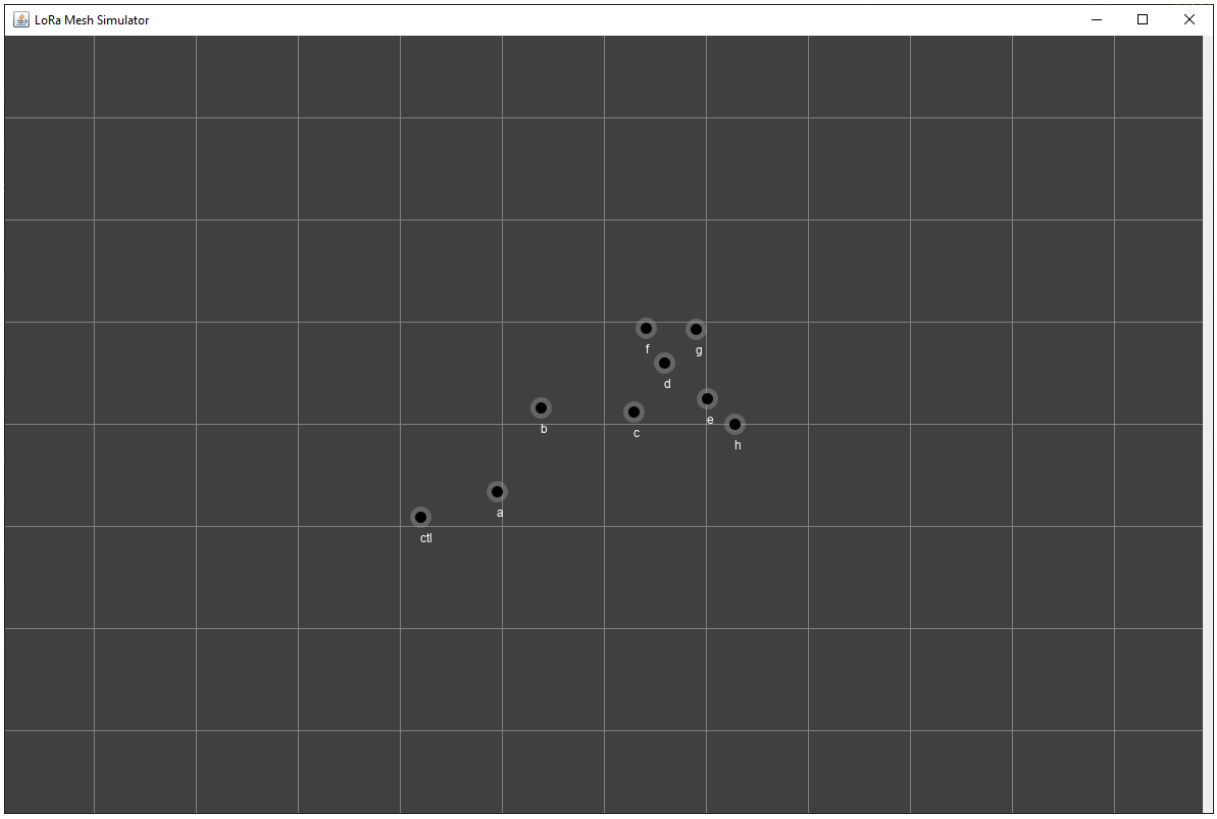

Figure 7: Node topology of uc2 (source: simulator)

The results are listed in table 6.1.2, using the same logical structure as before.

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	2186	2604	1885	2063	2022	2152.00
b	3496	2976	2192	2299	2522	2697.00
c	7550	-	-	4507	-	6028.50
d	-	-	-	-	-	-
e	-	-	-	-	-	-
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
average	3308.25	1860.33	1359.33	2217.50	1515.00	2719.63

Table 2: results data of test 2

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	2161	2529	3541	2307	2865	2680.60
b	3726	3249	5442	4413	11276	5621.20
c	-	-	-	-	-	-
d	2394	2861	3773	2569	3246	2968.60
e	4482	4399	7224	2946	7279	5266.00
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
i	-	-	-	-	-	-
average	2552.80	2607.80	3996.20	2447.20	4933.40	3307.48

Table 3: results data of test 3

Again, the number of nodes unable to join the mesh is significant. This topology looks even a bit worse with only 2-3 nodes being able to join, compared with the equivalent test using topology uc1, where 3-4 nodes succeeded.
It is worth noting that in both test cases, only nodes close to the controller were able to join.

6.1.3 Test 3: uc1; medium timeout; small volley; tracing off

For test number 3, we will go back to topology uc1, and increase the join-timeout paramete to mediumr. This means that a node will wait longer for a join-request to be answered before retrying. The other parameters stay untouched: small join-volley and message-tracing disabled.

The results are listed in table 6.1.3, using the same logical structure as before.

The obvious take-away from test number 3 is, that even if the average setup time improved in comparison to test 1, the number of successfully joining nodes did not change significantly (node b was also unable to join in test 1 run 1).

6.1.4 Test 4: uc1; medium timeout; medium volley; tracing off

For test 4, we want to try increasing the join-volley parameter to medium. This means that when issuing a join request, a node will send significantly more messages, increasing the chance of being heard and improving measurability of the initial LQI. We will stick to topology uc1 and also leave the other parameters untouched with respect to test 3: medium join-timeout and message-tracing disabled.

The results are listed in table 6.1.4, using the same logical structure as before.

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	4197	2165	2570	2219	26685	7567.20
b	9329	6197	12450	4661	31618	12851.00
c	-	-	-	-	-	-
d	3408	2548	2942	2609	27082	7717.80
e	3793	5664	5591	3760	29875	9736.60
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
i	-	-	-	-	-	-
average	4145.60	3315.00	4710.80	2650.00	23052.20	7574.72

Table 4: results data of test 4

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	2330	2022	2341	2945	2281	2383.80
b	8139	6359	-	19481	4308	9571.75
c	-	-	-	-	-	-
d	2733	2413	2740	3399	2689	2794.80
e	5582	3563	3296	4522	5094	4411.40
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
i	-	-	-	-	-	-
average	3757.00	2871.60	2094.50	6069.60	2874.60	3832.55

Table 5: results data of test 5

Looking at these results, there is no improvement with respect to the number of nodes being able to join the mesh. We can even observe a deterioration of the average setup time. However, this might also be owed to the fact that run 5 seems to be a significant outlier again.

6.1.5 Test 5: uc1; medium timeout; medium volley; tracing on

With message tracing being the only tested parameter that hasn't been touched so far, it is time to try and activate it. We will again leave the other parameters as in the previous test 6.1.4: Topology uc1, medium join-volley, and medium join-timeout.

The results are listed in table 6.1.5, using the same logical structure as before.

Not to the favor of the implemented message tracing mechanics, the results show that activating message tracing didn't help the ability of the mesh to set up autonomously. In run 3, there was even a node failing (node b), which used to join successfully in previous tests.

6.1.6 Test 6: uc1; long timeout; large volley; tracing on; extended simulation time

With no single parameter being able to get any of the nodes c, f, g, h, or i to successfully join in topology uc1, we will now try setting a long join-timeout, a large join-volley, active message-tracing, and also double the simulation timeout, leaving the mesh double the available time with respect to all previous tests, to try and set up its nodes.

The results are listed in table 6.1.6, using the same logical structure as before.

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	2198	2074	2492	2085	12949	4359.60
b	-	25062	12558	12126	34724	21117.50
c	-	-	-	-	-	-
d	3031	2913	3324	2921	13768	5191.40
e	5649	10276	4492	3884	20091	8878.40
f	-	-	-	-	-	-
g	-	-	-	-	-	-
h	-	-	-	-	-	-
i	-	-	-	-	-	-
average	2719.75	8065.20	4573.40	4203.40	16306.60	7909.58

Table 6: results data of test 6

Reading these results, we must acknowledge that even with double the available time the issue still remains. From this, we can only conclude that there must be a severe bug somewhere in the code that obstructs the mesh from working as intended, it makes little sense to test further parameters.

Also, since the successfully connected nodes only build a very small mesh, it seems pointless to try and analyze SDN-based metrics on the mesh.

6.1.7 Test 7: Multi-Hop Communication

Since none of the previous tests present a proof that multi-hop communication actually works, we set up a new simplified topology, only placing a few nodes in a straight line (figure 8) and overriding the link properties such that any two adjacent nodes have a 100% chance of successful message transmission, and

node	run 1	run 2	run 3	run 4	run 5	average setup time
ctl	1	1	1	1	1	1.00
a	966	5421	1011	913	935	1849.20
b	-	-	-	-	-	-
c	-	-	-	-	-	-
average	483.50	2711.00	506.00	457.00	468.00	925.10

Table 7: results data of test 7

non-adjacent nodes have a 0% chance of message transmission.

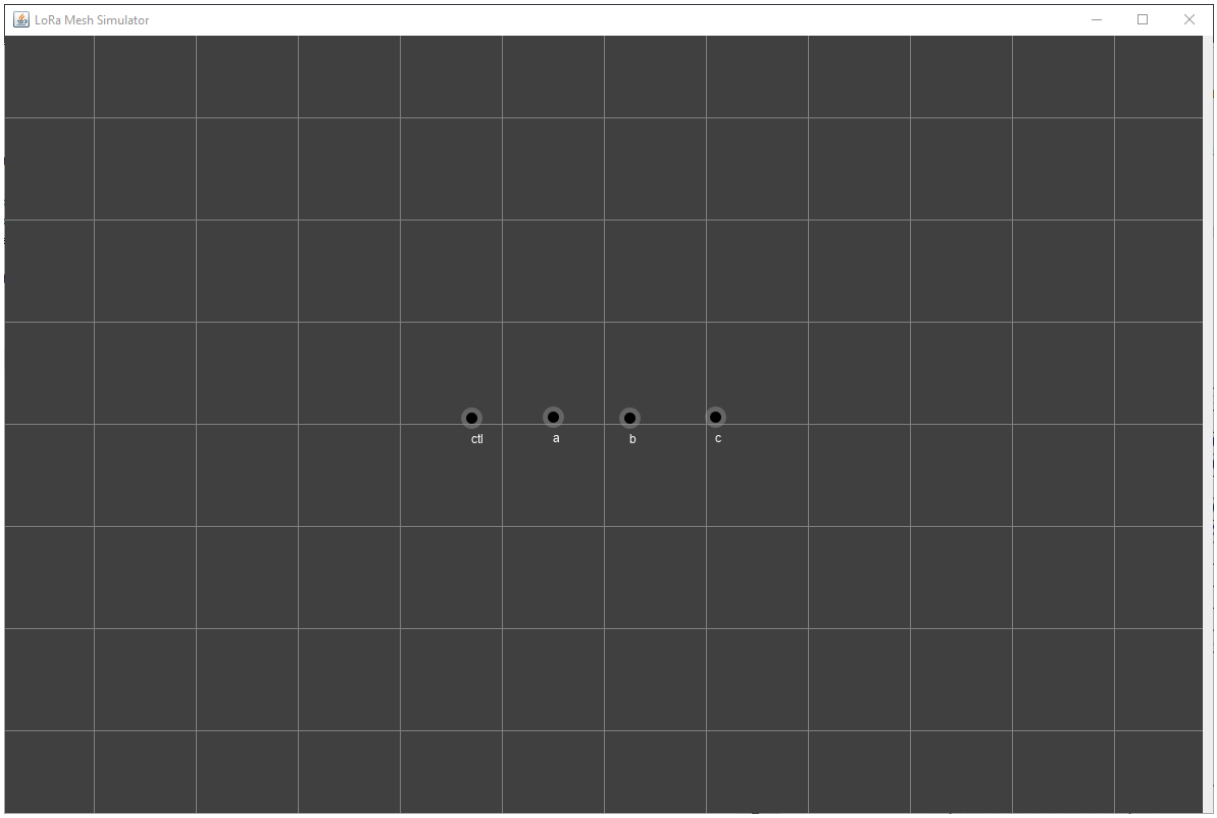

Figure 8: Simplified node topology before test run (source: simulator)

The results (table 6.1.7) now give us a very strong indication that multi-hop communication does in fact not work. This is also reflected in the resulting graphical view of the simulator (figure 9): Node a, which is adjacent to the controller, is able to join, node b, which is adjacent to node a, is able to detect the

mesh, and node c, which is adjacent to node b, keeps seeking the mesh.

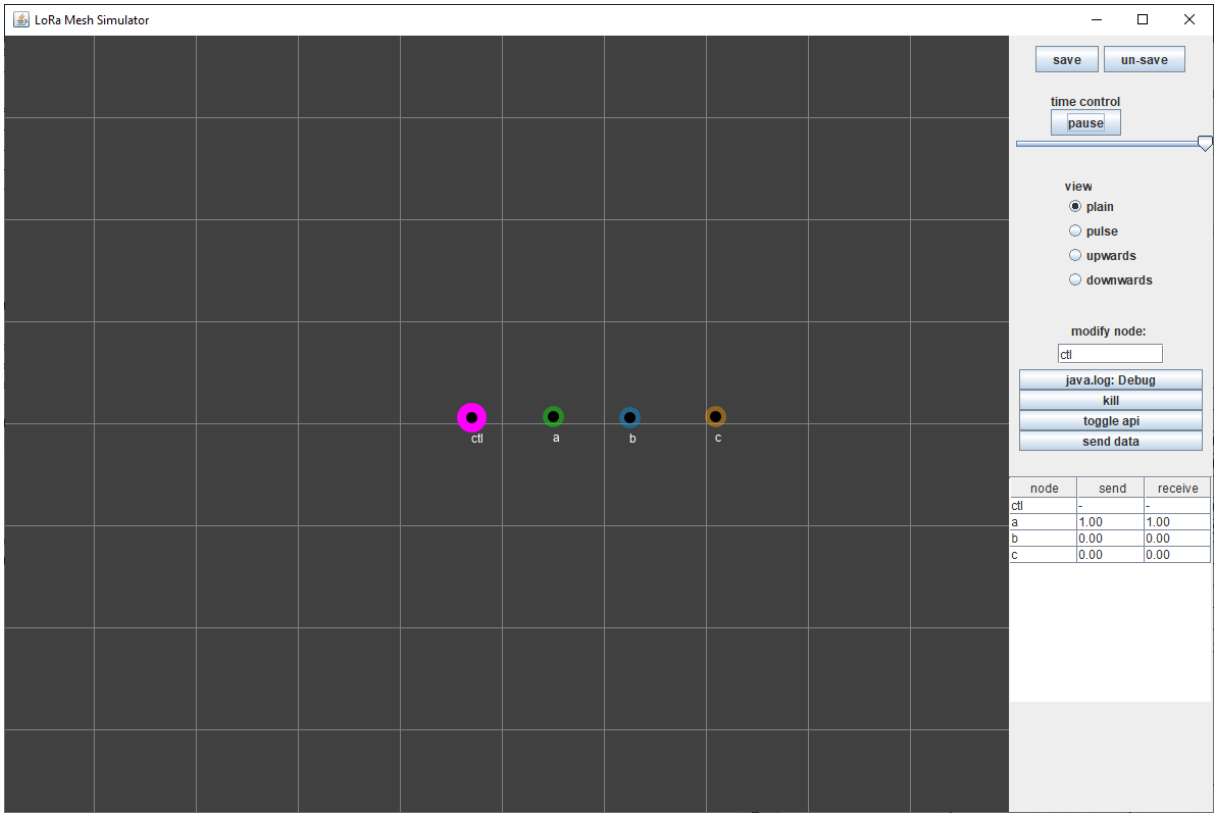

Figure 9: Simplified node topology after test run (source: simulator)

7 Conclusion

This chapter will focuses on subjective insights gained through the development of this thesis, addresses shortcomings, and proposes further steps to be taken on the topic.

7.1 Insights

Throughout the development of this bachelor thesis, software bugs were a significant issue. In the end, even with hundreds of hours of programming and the use of well-established design patterns, none of the attempted versions of the final product got to a point of running in a satisfying manner. For a developer with approximately half a dozen years of experience in Java, this produces the humbling insight that a sophisticated WSN carries a lot of complexity.

As for the topic of a fully fledged SDN-based LoRa mesh, the extension of LoRa with an SDN-based mesh, introducing multi-hop communication to extend the already impressive range of conventional LoRa, one might be able to cover use cases that would not be possible without. With IoT becoming evermore popular, research in this field is relevant without doubt.

7.2 Shortcomings

The lack of a functional product, along with the lack of informative data, is clearly a big shortcoming of this thesis. Maybe with more time the software could be fixed, allowing for the proper analysis that was initially planned.

Also, the simulator that was used as a tool for testing does clearly not qualify as a scientific instrument. While there are network simulation softwares available such as ns3 (used e.g. by [11], [10], or [21]), the computational model of the simulator implemented in this thesis is clearly too much of a simplification as to be the foundation of meaningful statements. In the defense of this thesis however, the original plan was to measure and run tests in the real world, which unfortunately was not possible in time. The simulator was originally planned as a debugging tool only.

7.3 Future Work

A future paper could use the software provided by this thesis as a basis to try again and test the LoRa mesh in the real world. It should certainly be possible to clear the remaining critical bugs and finally get the software running. All in all, the project still looks promising even after the second failed attempt.

List of Figures

List of Tables

References

[1] I. F. Akyildiz et al. "Wireless sensor networks: a survey". In: *Computer Networks* 38.4 (Mar. 2002), pp. 393–422. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(01)00302-4.

[2] Roberto Omar Andrade and Sang Guun Yoo. "A comprehensive study of the use of LoRa in the development of smart cities". In: *Applied Sciences* 9.22 (2019), p. 4753.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/J.COMNET.2010.05.010.

[4] Sezana Fahmida et al. "Real-Time Communication over LoRa Networks". In: *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE. 2022, pp. 14–27.

[5] Richard Feynman and Chapter Objectives. "Ebnf: A notation to describe syntax". In: *Cited on* (2016), p. 10.

[6] Ashish Gupta et al. "HybridLQI: Hybrid MultihopLQI for Improving Asymmetric Links in Wireless Sensor Networks". In: *2010 Sixth Advanced International Conference on Telecommunications*. 2010, pp. 298–305. DOI: 10.1109/AICT.2010.58.

[7] Johan Janssen and Henk Corporaal. "Controlled Node Splitting". In: *Compiler Construction*. Ed. by Tibor Gyimóthy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 44–58. ISBN: 978-3-540-49939-8.

[8] Emilia Rosa Jimson, Kashif Nisar, and Mohd Hanafi Ahmad Hijazi. "The state of the art of software defined networking (SDN) issues in current network architecture and a solution for network management using the SDN". In: *International Journal of Technology Diffusion (IJTD)* 10.3 (2019), pp. 33–48.

[9] Wolfgang Kellerer et al. "Adaptable and data-driven softwarized networks: Review, opportunities, and challenges". In: *Proceedings of the IEEE* 107.4 (2019), pp. 711–731.

[10] Sunil Kumar et al. "An NS3 Implementation of physical layer based on 802.11 for utility maximization of WSN". In: *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE. 2015, pp. 79–84.

[11] Ashadi Kurniawan, Prima Kristalina, and Moch Zen Samsono Hadi. "Performance analysis of routing protocols AODV, OLSR and DSDV on MANET using NS3". In: *2020 international electronics symposium (IES)*. IEEE. 2020, pp. 199–206.

[12] James F Kurose. *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[13] M. Tamer Özsu Ling Liu. *Encyclopedia of Database Systems*. Springer New York, NY, 2018.

[14] Raül Muñoz et al. "PCE: What is it, how does it work and what are its limitations?" In: *Journal of Lightwave Technology* 32.4 (2013), pp. 528–543.

[15] Luis ML Oliveira and Joel JPC Rodrigues. "Wireless Sensor Networks: A Survey on Environmental Monitoring." In: *J. Commun.* 6.2 (2011), pp. 143–151.

[16] Marko Paavola and Kauko Leiviska. "Wireless sensor networks in industrial automation". In: *Factory Automation*. IntechOpen, 2010.

[17] D Devi Kala Rathinam et al. "Modern agriculture using wireless sensor network (WSN)". In: *2019 5th international conference on advanced computing & communication Systems (ICACCS)*. IEEE. 2019, pp. 515–519.

[18] Daniel Reiss. "SDN-based LoRa Mesh". In: (2021).

[19] "IoT". In: *Encyclopedia of Wireless Networks*. Ed. by Xuemin (Sherman) Shen, Xiaodong Lin, and Kuan Zhang. Cham: Springer International Publishing, 2020, pp. 681–681. ISBN: 978-3-319-78262-1. DOI: 10.1007/978-3-319-78262-1_300308. URL: https://doi.org/10.1007/978-3-319-78262-1_300308.

[20] "WSN Management". In: *Encyclopedia of Wireless Networks*. Ed. by Xuemin (Sherman) Shen, Xiaodong Lin, and Kuan Zhang. Cham: Springer International Publishing, 2020, pp. 1515–1515. ISBN: 978-3-319-78262-1. DOI: 10.1007/978-3-319-78262-1_300748. URL: https://doi.org/10.1007/978-3-319-78262-1_300748.

[21] Md Abubakar Siddik et al. "Performance Evaluation of IEEE 802.11 for UAV-based Wireless Sensor Networks in NS-3". In: *arXiv preprint arXiv:2208.13268* (2022).

[22] SJ Suji Prasad et al. "An efficient LoRa-based smart agriculture management and monitoring system using wireless sensor networks". In: *International Journal of Ambient Energy* 43.1 (2022), pp. 5447–5450.

[23] Konstantinos Tzortzakis, Konstantinos Papafotis, and Paul P Sotiriadis. "Wireless self powered environmental monitoring system for smart cities based on LoRa". In: *2017 Panhellenic Conference on Electronics and Telecommunications (PACET)*. IEEE. 2017, pp. 1–4.

[24] Jian Wan et al. "An efficient gradient mechanism of directed diffusion in wireless sensor network". In: *2008 International Conference on Computational Intelligence and Security*. Vol. 1. IEEE. 2008, pp. 427–431.

[25] Siew Hong Wei et al. "Machine Learning as a Means to Adapt Requirement Changes for SDN Deployment Process in SDN Migration". In: *Advances in Computational Intelligence*. Ed. by Ignacio Rojas, Gonzalo Joya, and Andreu Catala. Cham: Springer International Publishing, 2019, pp. 629–639. ISBN: 978-3-030-20518-8.

[26] Yang Yu, Viktor K Prasanna, and Bhaskar Krishnamachari. "Energy minimization for real-time data gathering in wireless sensor networks". In: *IEEE Transactions on wireless communications* 5.11 (2006), pp. 3087–3096.

[27] Yuan Zhang et al. "Ubiquitous WSN for healthcare: Recent advances and future prospects". In: *IEEE Internet of Things Journal* 1.4 (2014), pp. 311–318.

[28] Longyu Zhou, Ning Yang, and Ke Zhang. "Positioning Improvement Algorithm Based on LoRa Wireless Networks". In: *Artificial Intelligence and Security*. Ed. by Xingming Sun, Zhaoqing Pan, and Elisa Bertino. Cham: Springer International Publishing, 2019, pp. 188–198. ISBN: 978-3-030-24271-8.

[29] Mengxia Zhu et al. *Example DSN Control Hierarchy*. Chapman and Hall/CRC, 2012.

A Hardware Setup

The instructions for hardware setup can be found in [18].

B Software Installation

The LoRa Mesh software can be used for the following purposes:

- Simulate a mesh topology on a single machine.
- Deploy a device as a node of your LoRa mesh.

The code is available on GitHub: https://github.com/Stud-FH/lora-mesh

Disclaimer: Since the software is not fully developed, it requires some experience to handle. For help, please contact florianandreas.herzog@uzh.ch

The code of the LoRa Mesh PCE API software can be used for the following purposes is also available on GitHub: https://github.com/Stud-FH/lora-mesh-api